

THESE

Présentation des travaux en vue de l'obtention du

Doctorat de l'Université Toulouse III

Spécialité : Informatique

par

Sandra BASNYAT

A Generic Integrated Modelling Framework for the Analysis, Design and Validation of Interactive Safety- critical Error-tolerant Systems

Soutenue le mardi 19 decembre 2006 devant le jury composé de :

- | | |
|------------------------|---|
| D. JAVAUX | Examineur <i>Human Factors, consultant, Liège (Belgium)</i> |
| C. JOHNSON | Co-Director <i>Professeur de l'Université de Glasgow (Royaume-Uni)</i> |
| P. JOHNSON | Rapporteur <i>Professeur de l'Université de Bath (Royaume-Uni)</i> |
| J.C. LAPRIE | Examineur <i>Directeur de recherche, Laboratoire d'Analyse et d'Architecture des Systèmes, CNRS (France)</i> |
| P. PALANQUE | Director <i>Professeur de l'Université Toulouse 3(France)</i> |
| F. VANDERHAEGEN | Rapporteur <i>Professeur de l'Université de Valenciennes (France)</i> |

Dedication

To my beautiful little princess, Leila Steere

Acknowledgements

I would like to thank my supervisor, Philippe Palanque who has been truly inspirational each day I saw him for the past three years. Philippe has dedicated so much time to ensure I fully understood what I was doing, to discuss papers we were writing, powerpoint slides we were preparing and so many more things even when he was “totally at the streets”! As well as inspiring me on an academic level, I have learnt so many other things from Philippe! About the reason of life for example! Which has of course been modelled using Petri nets by Philippe! He’s even explained that if you break a task model down enough you will end up explaining that it’s all about survival of the fittest gene! I’ve learnt how not to make travel arrangements with Philippe, how to miss flights! It’s all be fun, even the parts when he saw me vomiting ☺ I really feel lucky to have had a supervisor that I can also call a friend, someone who I was able to laugh with even when the thesis and everything else around me was driving me crazy.

Also, I would like to thank my co-Director and mentor, Chris Johnson who agreed to co-supervise me and allowed me to spend time in his group in Glasgow to work with his student Nick Chozos and also supported Nick in his visit to Toulouse for us to work together. I’d also like to thank Chris for his supersonic email replies which were always greatly appreciated since we (I won’t say I) were usually in a rush! I would therefore also like to thank Nick for sharing his perspectives with me and for discussing the fatal mining accident with me. I’m glad that while we were working together we had fun in the various countries we met up in! I’m glad I got to see your country too! Without these visits the incident and accident investigation perspective of the thesis would not have been developed.

I am grateful and thank my reviewers, Frédéric Vanderheagen and Peter Johnson for taking the time to read my thesis, for evaluating it and providing insightful comments. Also, my two other jury members, Jean-clause Laprie, Denis Javaux, who I thank for agreeing to be part of the jury.

I would like to thank the European Commission Research Training Network (RTN) for financing the 39 month ADVISES project and my PhD. I hope the results of the project will prove useful and beneficial to the EC.

I would like to thank the rest of the team in Toulouse who I can also call my friends. They helped me when I first arrived in France, especially Marco Winckler who helped me find a flat and gave me guided tours. The team have always been there to help me with whatever I needed, lifts in their cars, help with writing things in French! I thank Eric Barboni, AKA Didier, for being such a good friend, helping me with Petshop and introducing and getting me addicted to World of Warcraft (though that wasn’t such a great idea after all!). I’d like to thank David Navarre, who equally helped me with some of my models. Even though Xavier Lacaze is no longer part of the team, I’d like to thank him for all the help and time he gave me while he was here, and even after he left, when I felt totally alone in the office! Xavier was and is always ready to help with work and anything else I need (like lending me his kitchen to make food for the defence pot actually indeed or! ☺). I’d also like to thank Florence, for our little moments of madness in the office, for being a good friend and for helping me with the immense French administration! There was a lot of it! And Syrine also, for being such a great office buddy. I wish you both all the very best in finalising your PhDs.

To my friends IRL and IG in World of Warcraft, Grand guerrier de defence et chef de guild Fkai, the friendly giant for babysitting me in the early days! Grand guerrier de fury, Rhtaar for protecting me against all evil with his armur kipik! Tolchuk, Tölchuk and Tolbank, l’anglais dans une guild française, for always having the time to chat, being my taxi and a Chinese farmer to help pay for me my epic horse. I’m glad you’ve learnt a little bit of French in any case!. And also Nhougrod, un petit chat malicieux aux allures d’ours féroce for helping me whenever I needed. We have had and will continue to have lots of fun ;) Je vous soignera toujours!

Finally, I would like to thank my family, for helping me out whenever I needed (mostly financially!). My mum, who came to Toulouse within hours if I was ill and coming with me to different conferences! Ahh, the perks of a PhD! Thanks to you all for coming and sharing my defence day with me. And thank you to my beautiful little niece, my princess Leila, for giving me hours of pure pleasure, for making me laugh and smile without even trying.

Table of Contents

INTRODUCTION.....	1
1.1 <i>Motivations</i>	2
1.2 <i>Contribution</i>	2
1.3 <i>Structure of the thesis</i>	4
CHAPTER 1	7
1 INTRODUCTION.....	7
2 MODEL BASED DESIGN	8
2.1 <i>UML</i>	9
2.2 <i>RUP</i>	10
2.2.1 UML and RUP Section Summary	11
2.3 <i>Design Models</i>	12
2.4 <i>Model-Based Design Section Summary</i>	14
3 USER CENTRED DESIGN	14
3.1 <i>Task Analysis</i>	15
3.2 <i>Modelling tasks</i>	15
3.2.1 Collaborative Task Modelling.....	17
3.2.2 Task Modelling and Human Error.....	21
3.2.3 Task patterns	21
3.3 <i>Usability Analysis</i>	23
3.4 <i>Stakeholder Analysis</i>	24
4 MODEL BASED INTERACTIVE SYSTEMS ENGINEERING	25
4.1 <i>System Modelling & Notations</i>	26
4.1.1 Formal specification techniques for systems modelling	27
4.2 <i>Formal Analysis Techniques</i>	32
4.2.1 Marking Graph Tool Support	34
5 INTERACTIVE COOPERATIVE OBJECTS (ICOS) & PESHOP.....	36
5.1 <i>Petshop Environment</i>	38
5.2 <i>Scalability of ICOs and Petshop</i>	40
5.3 <i>Integrating Human Factors with ICOs</i>	41
6 MODEL BASED SAFETY CRITICAL INTERACTIVE SYSTEMS DESIGN	43
6.1 <i>Formal approaches for describing interactive systems and human-computer interaction</i>	44
6.1.1 Examples of FSTs for Interactive Applications Design.....	45
6.1.2 MUSE*/JSD.....	46
6.1.3 MEFISTO.....	47
7 SPECIFICITIES OF SAFETY-CRITICAL SYSTEMS: THE CERTIFICATION PHASE	48
8 CONCLUSION	49
CHAPTER 2	51
1 INTRODUCTION.....	51
2 BACKGROUND OF RESILIENCE.....	51
2.1 <i>System Safety</i>	53
2.1.1 System Definition.....	53
2.1.2 Safety Definitions.....	53
2.1.3 Safe Systems	54
2.2 <i>Resilience in organisations</i>	55
3 ACHIEVING INTERACTIVE SYSTEMS SAFETY BY RESILIENCE ENGINEERING	57
3.1 <i>Learning from Experience</i>	57
3.1.1 Incident and Accident Analysis.....	58
3.1.2 Overview of Safety Analysis Methods	59
3.1.3 Formal Methods for Safety Analysis.....	62
3.2 <i>Anticipating Failure & Vulnerabilities</i>	65
3.2.1 Hardware Failures	65
3.2.2 Software Failures.....	68
3.3 <i>Means to argue safety: safety cases</i>	74
3.3.1 Representing safety cases.....	75
3.4 <i>Human Factors in the Design of Safety Critical Interactive Systems</i>	76
3.5 <i>Human Error</i>	76

Table of contents

3.5.1	Human Error Theories	77
3.5.2	Error Management	82
3.5.3	Error Analysis Approaches	82
3.5.4	Accident Models	85
3.6	<i>Tasks and user errors</i>	89
4	CERTIFICATION PHASE	91
5	CONCLUSION	92
PART 1 CONCLUSION		93
PART 2 – THE APPROACH		95
CHAPTER 3.....		97
1	INTRODUCTION	97
1.1	<i>Gathering Information</i>	99
1.2	<i>Formalising Information</i>	99
2	MULTI-TYPE DATA	99
2.1	<i>Pre-design data</i>	99
2.2	<i>Post-design data</i>	101
3	MULTI SOURCE DATA	101
4	ULTIMATE GOALS	102
4.1	<i>Consistency</i>	102
4.1.1	Model Consistency	102
5	A GENERIC FRAMEWORK FOR ENSURING CONSISTENCY	103
5.1	<i>Design Process Centred on Formal Models</i>	104
5.2	<i>Generic Integrated Modelling Framework</i>	104
6	SUPPORTING INTEGRATION.....	111
7	CONCLUSION	111
CHAPTER 4.....		113
1	INTRODUCTION	113
2	TASK MODELLING.....	115
3	ERROR MANAGEMENT.....	116
3.1	<i>Designing with error in mind</i>	117
3.2	<i>Human Error Reference Tables</i>	118
4	MODEL PATTERNS FOR ERROR TOLERANCE.....	121
4.1	<i>Relating types of user “error” to model patterns</i>	121
4.1.1	Identification of possible deviations	122
4.1.2	Including model patterns for error tolerance in task models	123
4.1.3	Error modelling.....	124
4.2	<i>“Plugging in” model patterns for error tolerance</i>	128
5	SUPPORTING INTEGRATION.....	130
6	CONCLUSION	130
CHAPTER 5.....		133
1	INTRODUCTION	133
2	MODELLING INTERACTIVE SYSTEMS ON THE ATM EXAMPLE	134
2.1	<i>Behaviour</i>	135
2.2	<i>Interaction</i>	137
2.2.1	Activation Function - Event-based	138
2.2.2	Rendering Function	138
3	ERROR TOLERANCE	139
3.1	<i>Relating HERTs Task Analysis to System Modelling & Error Management</i>	139
3.2	<i>Handling of data entry errors</i>	143
3.3	<i>Handling of sequence errors</i>	146
3.4	<i>Explicit handling of timing errors</i>	147
4	MODEL VERIFICATION	148
4.1	<i>Verification of properties</i>	148
5	SUPPORTING INTEGRATION.....	151
6	CONCLUSION	152
CHAPTER 6.....		153

1	INTRODUCTION.....	153
2	INCIDENT AND ACCIDENT INVESTIGATION	154
2.1	<i>A Multidisciplinary Approach to Incident & Accident Investigation to Inform.....</i>	155
2.2	<i>Cash Machine Failure Scenario.....</i>	158
3	APPLICATION OF THE APPROACH.....	159
3.1	<i>Safety Cases.....</i>	159
3.2	<i>Events and Causal Factors Analysis</i>	162
3.3	<i>Relating Safety Analysis to System Model Validation</i>	162
3.3.1	Using Flawed Safety Cases to Inform the System Model	163
3.3.2	Modification 1: Entering/Exiting the Airlock.....	165
3.3.3	Modification 2: Fire, Alarm and Trapping Feature	166
3.3.4	Modification 3: Card In Machine Place.....	166
3.3.5	Relating the ECFA to the System Model Including Hazardous States	167
3.3.6	Marking Graph to Identify Further Scenarios Leading to the Same Incident	168
3.3.7	Improving the System Model to Avoid the Hazardous State.....	170
4	SUPPORTING INTEGRATION	170
5	CONCLUSION	171
CHAPTER 7		173
1	INTRODUCTION.....	173
1.1	<i>Methods for Identifying Barriers (pre and post incident/accident).....</i>	175
2	POST-INCIDENT BARRIER IDENTIFICATION.....	175
2.1	<i>Using Safety Cases to Identify Barriers.....</i>	176
2.2	<i>From the Incident/Accident Report</i>	177
3	APPROACH TO POST-INCIDENT/ACCIDENT BARRIER ANALYSIS AND MODELLING.....	177
3.1	<i>Safety Modelling Language</i>	179
4	PRE-INCIDENT SOFTWARE BARRIER ANALYSIS AND MODELLING	180
4.1	<i>Approach to software barrier analysis and modelling</i>	181
4.1.1	Interaction Hazard Analysis/Unintentional poor design	182
4.1.2	Software Barrier Identification.....	186
4.1.3	Software Barrier Modelling.....	188
4.2	<i>Barrier and System Model Integration.....</i>	190
5	SUPPORTING INTEGRATION	190
6	CONCLUSION	191
PART 2 CONCLUSION.....		193
6.1	<i>Dealing with the cost/benefit issue</i>	193
6.1.1	Classical development processes.....	193
6.1.2	Usability	194
6.1.3	Safety	195
6.1.4	Reliability.....	195
6.1.5	Methodological based cost-reducing techniques	196
6.1.6	Tool based cost-reducing techniques.....	197
6.2	<i>Summary.....</i>	197
CHAPTER 8		201
1	INTRODUCTION.....	201
2	THE CASE STUDY AND THE GENERIC INTEGRATED MODELLING FRAMEWORK.....	202
3	INTRODUCTION TO THE CASE STUDY	203
3.1	<i>Personnel.....</i>	206
3.2	<i>Engineering and the Case Study.....</i>	206
3.3	<i>Events Leading to the Accident.....</i>	208
4	INFORMATION PROPAGATION THROUGH MODELS.....	208
5	TASK MODELS OF OPERATORS.....	209
6	HUMAN ERROR ANALYSIS	212
7	MODEL PATTERNS FOR ERROR TOLERANCE.....	214
8	SAFETY CASE ANALYSIS.....	217
8.1.1	Water-Hammer Effect.....	217
8.1.2	Manual Handling.....	219
8.1.3	Failure of the Command and Control System	220
8.2	<i>Barrier Analysis based on Safety Case Analysis</i>	221
9	COMMUNICATION SEQUENCE.....	222

10	EVENTS AND CAUSAL FACTORS ANALYSIS	223
11	SYSTEM MODEL OF WFDS	225
12	MAPPING ERRONEOUS EVENTS TO SYSTEM MODEL	230
12.1	<i>Relating ECF and System Modelling</i>	230
12.2	<i>Remodelling the System Exploiting Safety Cases</i>	231
12.3	<i>Adapting System Model to Prevent the Accident</i>	234
12.3.1	Marking Graph Analysis.....	234
13	BARRIER ANALYSIS	238
13.1	<i>Step 1: H-B-T analysis</i>	239
13.2	<i>Step 2: Barrier modelling using Petri Nets & ICO formalism: Pump Priming Procedure</i>	241
13.2.1	ICO modelling of PB1: Pump Priming Procedure	241
13.2.2	ICO modelling of PB2 Auto-Shutdown of Pumps	243
13.3	<i>Step 3: Connecting technical and human barriers in the system model</i>	244
14	CONCLUSION	249
CHAPTER 9.....		251
1	INTRODUCTION	251
1.1	<i>Contributions on this case study</i>	252
2	THE CASE STUDY AND THE GENERIC INTEGRATED MODELLING FRAMEWORK	253
3	INTRODUCTION TO THE CASE STUDY	254
4	ARINC 661	257
4.1	<i>Informal overview of ARINC 661</i>	257
5	SYSTEM MODEL OF THE MPIA APPLICATION.....	259
5.1	<i>Formal Description of the MPIA application: WXR Page (weather radar)</i>	259
5.1.1	Behaviour	259
5.1.2	Activation Function	260
5.1.3	Rendering Function	261
6	HAZARDOUS INTERACTION ANALYSIS.....	261
7	SOFTWARE BARRIER IDENTIFICATION	266
8	BARRIER CLASSIFICATION	267
8.1.1	Barrier Classification	267
9	SOFTWARE BARRIER MODELLING	268
9.1.1	An example of Barrier Modelling	268
10	CONCLUSION	269
CONCLUSIONS AND PERSPECTIVES		271
PERSONAL PUBLICATIONS		275
BIBLIOGRAPHY.....		277
LIST OF FIGURES.....		294
LIST OF TABLES.....		301
PART 4 – ANNEX		303
SKILL-BASED HUMAN ERROR REFERENCE TABLE.....		304
SAFETY ANALYSIS METHODS		309
INDEX		311

Introduction

Human-computer interaction engineering is a discipline situated at the intersection of several disciplines: software development, hardware design and interface design. Software engineering offers a set of processes, methods and tools targeting at the design and development of reliable software. However, the lack of explicit consideration for the final user makes these processes and methods inadequate for the design of interactive systems. One of the main characteristics of interactive systems is the fact that the user is deeply involved in the operation of such systems. This means, for instance, that system output must be both perceived and correctly interpreted by the users. Taking into account such constraints requires knowledge about the users that cannot be gathered using techniques and methods in software engineering.

In contrast, human-computer interaction is user-centred and emphasizes the importance of utility and usability. This research area advocates the use of a highly iterative development process, based on the successive generation of prototypes, evaluated with the final user. It also investigates the way to increase bandwidth between the human and system, by providing new input and output devices and new interaction techniques.

Usability problems might appear in different forms in the user interface, reducing the user performance with the device, increasing the number of errors or having users being reluctant to use the device due to uncomfortable/unpleasant interaction. In the last decades, the Human-Computer Interaction (HCI) community has developed several methods to support sound and rigorous identification of usability problems. Many approaches have been proposed to cope with the diversity of contexts in which evaluation has to take place. Such methods may involve observation of users' activity, inspection by a usability specialist, simulation and/or prediction of usability problems based on models describing users' expected activity like, for instance, task models. Model-based approaches provide better support for the design of interactive systems, for example by identifying usability problems in the early phases of the development process thus reducing the time and development costs.

Such development approaches are beneficial for the systems the majority of the population interacts with on an almost daily basis. However, in the field of safety-critical interactive systems, ensuring usability, as well as the more straightforward requirements such as reliability and safety is crucial.



Railway Systems



Aviation Domain



Railway Systems



Cockpits

Figure 1-1. Safety-Critical Interactive Systems

A safety-critical system is one in which any failure or design error has the potential to lead to loss of life or, in other terms, that failure outweighs the cost of development. Safety-critical interactive systems add the human dimension to a software system by putting control into the hands of a human operator. Examples of such systems include nuclear power plants, railways systems, airplane cockpits, and military systems.

1.1 Motivations

When adding a human to the control of such complex safety critical interactive systems, the unpredictable nature of humans can unfortunately bring in human unreliability issues, in addition to technical unreliability. An incident or accident is almost never directly the fault of an individual human, though we often read and hear about “human error” as the blame and that more than 80% of accidents in aviation are attributed to human error (Johnson, 2006).

The operators of such systems may have heavy responsibilities, such as human lives. Small interaction problems, because of poor design, could contribute to an incident or accident. The complexities of such safety-critical interactive system accidents can be seen for example in the Kegworth air disaster (Figure 1-2). On the 8th of January 1989, a British Midland Airways (BMA) Boeing 737-400 aircraft crashed into the embankment of the M1 motorway near Kegworth, resulting in the loss of 47 lives. The pilot and co-pilot throttled back their one working engine rather than the failed engine during the Kegworth air crash. The Kegworth investigation concluded that in-board systems failed to prevent pilots from shutting down a healthy engine (Air Accidents Investigations Branch 1990). This could be seen as human error, in that the pilot failed to shut down the faulty engine, however, from a User-Centred Design perspective, this accident is partly attributed to a poor system design and interaction failure. The pilots struggled with the system for many minutes until they realised their mistake, by which time it was too late to take corrective action. The interaction failure occurred because “the crew believed that their objective was achievable by following a particular course of action, but the actions they took closed down the space of future interaction possibilities, and the feedback they received did not alert them to their misunderstandings until too late” (Blandford et al. 2003).



Figure 1-2. Kegworth Air Disaster

The design of a usable, reliable, safe and error-tolerant safety-critical interactive system is a goal that is hard (actually impossible, because we cannot guarantee either of these characteristics) to achieve because of the unpredictability of the humans involved, but can be more closely attainable by taking into account information from previous known situations. One such usually available and particularly pertinent source is the outcome of an incident or accident investigation.

With the objective of increasing safety, in safety-critical interactive systems, previous research in the field aimed at trying to eliminate the error completely by identifying its source. It has now been widely accepted however that human ‘errors’ are inevitable due to the idiosyncratic nature of humans and we must instead try to manage errors. The perspective of blame has also changed from isolating an individual operator to having a wider outlook on the organisation as a whole. However, the broader the perspective, the more information has to be gathered and thus making it more complex not only to organise it but also to reason about it.

The design and implementation of such systems requires the definition of specific methods, able to deal with usability, reliability and safety. For many years, research in human-computer interaction engineering intended to offer such methods for systems involving simple interaction techniques. In extension to these UCD approaches and in order to deal with these complex combinations of factors and systems, we promote the use of formal methods and model based design as a way of representing these components and their interrelations whether it is design, construction or investigation.

1.2 Contribution

This thesis proposes a multidisciplinary generic integrated modelling framework for the design of safety critical interactive systems. The goal is to propose means (such as model based design techniques, notations, tools,

methods) that allow taking into account and managing erroneous human and system behaviour. While designing interactive systems, the use of a formal specification technique is of great help because it provides non-ambiguous, complete and concise notations. The advantages of using such a formalism is widened if it is provided by formal analysis techniques that allow to prove properties about the design, thus giving an early verification to the designer before the application is actually implemented.

This is not a new goal for the field, and, in order to tackle these issues, modelling processes and techniques have been defined and applied widely in the field of safety critical systems. Model-based development (MBD) is a developing trend in the domain of software engineering (MDA Guide version 1.0.1 2003) advocating the specification and design of software systems from declarative models (Puerta 1998). It relies on the use of explicit models and provides the ability to represent and simulate diverse abstract views that together make up a 'system', without the need to fulfil its implementation. It is widely accepted within the community that models are needed for the design of safety critical interactive systems; this is to be able to understand issues such as safety and resilience and to think about how safety can be ensured, maintained, and improved (Hollnagel and Woods 2006b).

Our research focuses mainly on task and system modelling, which like other models are generally developed from an error free perspective, that is to say, without taking into account human or system-related "errors". Task models often uncover deep-rooted problems, for instance, in workload allocation across many different aspects of an interactive control system. What is more, task modelling and system modelling are often performed by experts with different competences. It is unlikely that a human factors specialist will develop the task model and continue to develop the system model. A computer programmer will also in general be incapable of collecting the necessary information for task modelling.

We have developed an approach that supports the integration of information relating to human and system-related erroneous behaviour in models (principally the task and system models) by feeding models with information from human error analyses, incident and accident investigation analyses and barrier analysis. Incorporating such data significantly increases the size of the models, thus we also define model patterns for error tolerance as a means of dealing with these complexities. Furthermore, we use existing tools for the task modelling and system modelling to support our approach. We believe this perspective extends the general boundaries of model based development (MBD), by taking into account additional information relating to previous experiences of failure. The ultimate goal is improvement of the design process with the aim of producing safer, more reliable and more usable safety-critical interactive systems.

We advocate a design practice where task and system models are tightly integrated and produced together within an iterative process. The design starts either with some initial model of the system (which may originate from the existing situation, e.g. analysis of the paper documents in an information system analysis) or with an initial task model which provides the end-user requirements. Task models and system models are then evolved incrementally. After each modification the complexity of task models is quantitatively assessed. The system designers propose modifications in the system models, in order to allow building simpler and more efficient task models. Task models are built in accordance with the new system, and analysed once again. This iteration will continue until satisfactory task models are produced. In our approach task and system models are not deduced from one another, but on the contrary are evolved in synergy and are checked for compliance with each other.

We refer, as a paradigm, to Diderot and d'Alembert's (Diderot and d'Alembert 1751-1772) description of the necessary tools for building a barrel, which includes many forms of an axe (see Figure 1-3). The model-based development process we propose for a safety critical interactive system requires dedicated methods, notations, tools and processes. Even though methods such as task modelling, or system modelling appear to be standard and applicable to many systems, we suggest that there are in fact many forms of task modelling and system modelling, and that they must be adapted and tailored to suit the system under design. This is also why we do not propose only one notation that might fit all our needs but instead several notations that provide dedicated mechanisms for each model that has to be built.

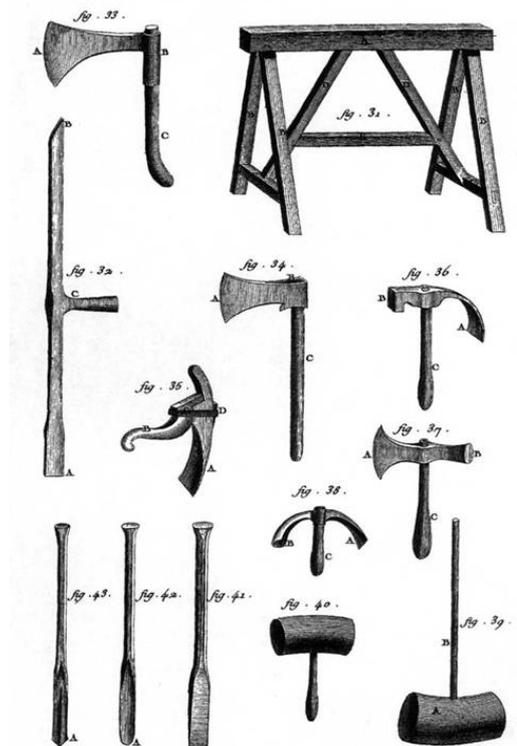


Figure 1-3. Types of axes needed to build a barrel (taken from (Diderot and d'Alembert 1751-1772))

1.3 Structure of the thesis

Chapters 1 and 2 form part 1 of this thesis which presents an overview of the state of the art relating to the various domains relevant to this research, including model-based design, resilience engineering and human error analysis techniques.

Chapter 3 begins part 2 of the thesis, which introduces the complexities surrounding the design of safety critical interactive systems and presents the generic integrated modelling framework for the first time. The subsequent chapters present the various phases of the approach, separated as task modelling and error management, system modelling and error management, safety modelling and barrier modelling. All of the proposed model-based design techniques are exemplified on an ATM Cash Machine example. Each of these chapters contains a dedicated section describing how the phase connects with other phases and how data is shared between phases.

We also show how generic integrated modelling framework can be used to support the design of systems while considering safety reliability and usability by encompassing error tolerance in the models. Finally, we illustrate our contribution through two case studies. The first case study is a fatal mining accident, while the second is a software based cockpit application that has currently not attributed to any accident.

PART 1 – STATE OF THE ART

Chapter 1

Design Processes and Methods

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.” (Albert Einstein, 1879 - 1955)

Chapter Summary

This chapter proposes a global overview of current pertinent research in model-based design with particular focus on safety-critical interactive systems and formal specification techniques for describing such systems. The aims of this chapter are twofold: firstly to survey the existing literature with respect to the aims of this thesis, and secondly to provide background for concepts on which we build in subsequent chapters. Although exact classification is impossible, we structure the review according to work generally concerned with modelling, reasoning, and specification in safety-critical interactive systems in particular, and also review expressiveness of formal specification techniques from wider catchments, including object orientation. Reference is made, where appropriate, to any pragmatic concerns. The advantages and disadvantages of current approaches will be discussed concluding with a justified need for improved methods.

1 Introduction

Human Computer Interaction (HCI) is the interdisciplinary study of interaction between systems and humans with a goal of improving user experience. A system that provides good user experience is referred to as user-friendly or usable. Interaction between humans and systems occurs via an interface which can be physical hardware, or a software application for example. In addition to computer science, typical disciplines that contribute to the field of HCI include anthropology, cognitive science, design, ergonomics, and psychology.

An interactive system, presented in Figure 1-1 is a reactive system, which means a system that responds to external stimuli and the control flow relies on the user. An interactive application will remain idle while waiting for input from a user, following which it will execute and be influenced by internal and external states. These states will have an effect on the way the system will react. Such a system results from the fusion of a functional core from the application which carries out calculations, and an interface, together provide the means of entry and exit proposed to the user to obtain the execution of a work on behalf of the system. One of the main characteristics of interactive systems is the fact that the user is deeply involved in the operation of such systems.

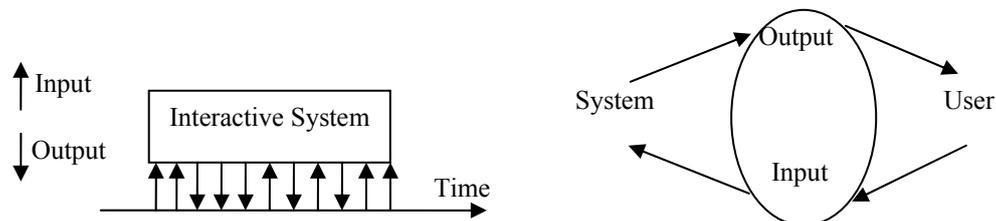


Figure 1-1. Representation of an Interactive System

The complexity of the development of an interface is variable because an interface is also a command line as well as a group of windows and menus that are manipulated with a mouse (known as WIMP, Windows Icon Menu and Pointer). More complex interfaces are known as post-WIMP (van Dam 1997), those involving visualisation, 3D manipulation, voice commands as data entry etc.

Even though the classical development processes are employed for the development of an interactive system, it is not simple. The limits of such processes lie in their cost in terms of time and money, in terms of new iterations of the same process due to inadequacies between the produced system and the real requirements.

The objective of this chapter is to exhaustively present current research in the field of design methods for interactive systems and more specifically, of safety-critical interactive systems (SCIS) from a User Centred Design (UCD) perspective. Although design methods are applied in most fields for most systems, a UCD approach is indispensable for the design of an interactive system because of the importance the role the end user has with the product. If the end user cannot use the designed system, because of interaction problems, physical and software related usability problems, design ignorance to user disabilities and user capabilities, the system is almost worthless. Taking this idea one step further, if any of the above types of problems occur within a safety critical context, the impact of the problem is exemplified and resulting consequences can be catastrophic.

The evolution of UCD methods for interactive systems and for SCISs is presented. It can be seen that these compel a focus on model-based design. The chapter contains references of the domain of UCD methods for interactive and SCISs necessary for obtaining and understanding of the domain.

Since this thesis is multidisciplinary, several domains must be touched on before the core of the thesis can be discussed. Figure 1-2 illustrates this intersection of theories presented in this chapter lying in UCD, Systems Engineering, Human Factors, Formal Specification and Model Based Design which are presented in this chapter.

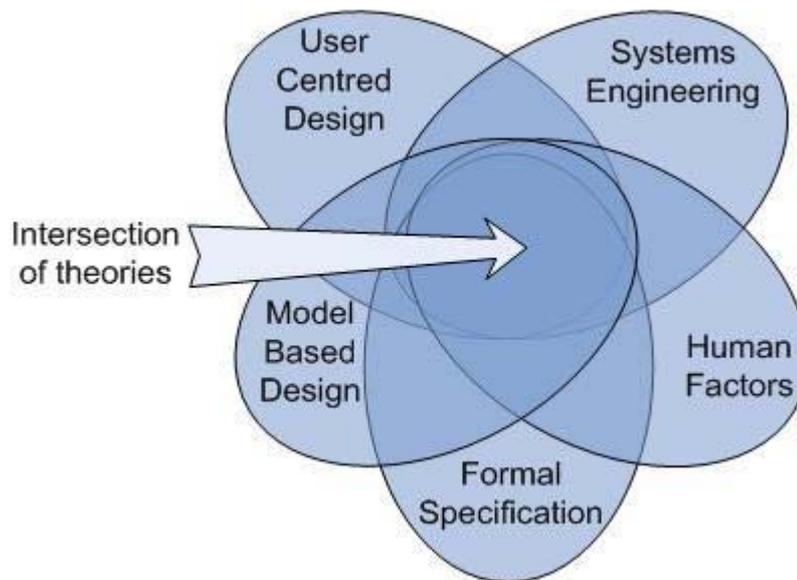


Figure 1-2. Intersection of Chapter Theories

2 Model Based Design

This section details model based design methods for designing interactive systems. Hollnagel and Woods (Hollnagel and Woods 2006a) affirm that models are needed to be able to understand issues such as safety and resilience and can be used as a way to think about how safety can be ensured, maintained and improved. According to the authors, a model should fulfil two requirements:

1. It should provide an explanation or bring about an understanding of an event such that effective mitigating actions can be devised
2. It can be used with reasonable intellectual effort, as well as time and cost efforts

This leads to the typical modelling problem that a model should be detailed enough to include all necessary information but not too detailed so that it becomes unclear.

The methods presented in this section are currently used in industry. We first describe UML, a technique used for modelling applications (that is not a design method) followed by RUP project management method which uses UML. UML is currently a standard for modelling objects. We then tie together model based design approaches with safety-critical systems analysis to illustrate how formal notations can be used to enhance the design of such systems. We describe the development process associated to UML and to RUP as well as the advantages and disadvantages of each method.

2.1 UML

UML (Unified Modeling Language) (Booch et al. 1999) is a semi-formal language for specifying interactive systems. The language has been normalised by the Object Management Group (OMG) in order to provide universal communication support that is independent of the application domain and programming language.

The authors recommend the use of UML for the following reasons; it is iterative and incremental, it is guided by the needs of the system users and is centred on the architecture of the software.

UML provides several points of view of a system with a number of levels of abstraction. These different viewpoints are represented by different types of diagrams including:

Structural diagrams (static)	Activity diagrams
Class diagrams	Collaboration diagrams
Object diagrams	State transition diagrams
Component diagrams	Module management diagrams
Deployment diagrams	Packages
Behavioural diagrams (dynamic)	Sub systems
Use case diagrams	Models
Sequence diagrams	

The class diagram models the conceptual architecture of a system by specifying the classes used the relationships between those classes. The relationships are increased with cardinality and/or with the particular behaviour between classes, for example, the heritage relationship.

The object diagram is an example of a particular class diagram, it is a class instance. This type of diagram specifies objects and relationships between these objects.

The component diagrams model the physical architecture of the application and describe the modules and relationships between modules. A module models a source file or an imported library. The component diagram highlights the dependence between files that allow the identification of future constraints related to the compilation.

The deployment diagram is closely related to the modelling of resource materials imposed in the system for example a computer, model, server etc. A node characterises a resource. The relationships between node describe what communicates with what and how the communication is supported.

Use cases model the needs of the client by characterising the interactions between the actors and the system. The actors are different types of users who will eventually use the system. The actors can visualise the information proposed by the system and can modify the state of the system, in which case, the system must provide a response to the modification. This “view” is abstract, allowing the identification of necessary functionalities of the system for the users to realise their tasks. Use cases provide an understanding of the functioning of the global system.

The sequence diagrams model the temporal elements of actions between objects and actors while the chronology of the actions are traced. These diagrams illustrate the scenarios of use by adding a temporal dimension (represented with a vertical line). The actions between objects and the actors are called messages. The messages can have different syntaxes and are represented graphically.

UML identifies five types of messages, simple messages, synchronous messages, asynchronous messages, dropped messages and timeout messages.

Activity diagrams are a representation of the behaviour of a method or the execution of a scenario of use. This formalism provides a way to express the sequence, synchronisation, parallelism and conditions like the “if, then, else” can also be modelled between activities.

Collaboration diagrams are related to the modelling of interactions between instances of classes (i.e. objects) and the actors. These diagrams are very precise since it is possible to define a value for an attribute. The relationships between elements of a diagram are represented by the messages characterised by their rating, their send conditions etc.

State transition diagrams are used to model state changes of an object following an action imposed by an actor and/or of another object. States and transitions are used to model these changes. A state represents the values of attributes of a group of objects and of a given instance. The transitions are triggered by an event or automatically. The model the change from one state to another. The transitions can be modified to represent behaviour such as “if then else”. These diagrams are representation of finite state machines.

To conclude, UML is a standardised language used to model in an object oriented approach. Though there is no proof that UML respects the concepts of object oriented modelling. The multiple views are nevertheless complementary to the problem as well as the different levels of abstract which allow us to understand the behaviour and the dependencies of the system

2.2 RUP

RUP (Rational Unified Process) (Kruchten 2000), based on UP (Unified Process) is a design method based on UML which allows the production of quality products, meeting user requirements. The RUP approach is guided by three basis and six rules of good design. RUP is an iterative and incremental process centred on the architecture of a system and is driven by use cases.

Iterative and incremental: the project is divided into iterations of short length, at the end of each iteration an executable version of the application is produced, these versions are incremented during each iteration

Architecture centred: the architecture should be modelled

RUP defines a group of good practices:

1. Develop the application iteratively
2. Manage the needs
3. Use an architecture based on components
4. Use visual models
5. Verifier the quality of the application
6. Manage/control the changes of the application

Figure 1-3 presents a view of the process decomposed in two dimensions. The horizontal axis represents the time and the deployment of the lifecycle. The vertical axis represents the disciplines, the people involved and the artefacts implicated in the process.

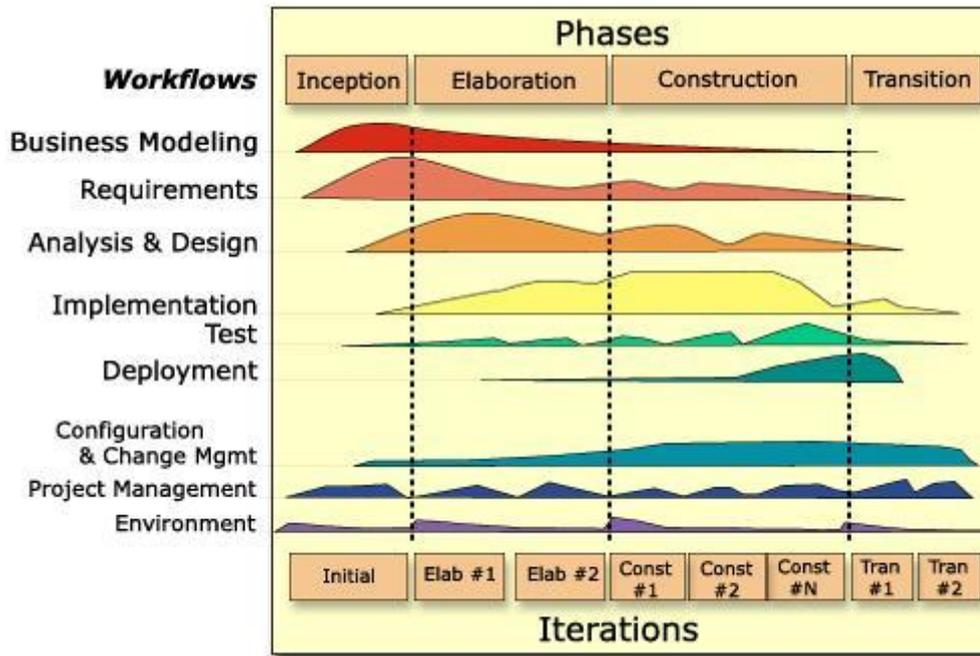


Figure 1-3. The RUP Lifecycle (from (Kruchten 2000))

The first dimension (the horizontal axis) expresses the dynamic aspect of the process in terms of cycles, phases and iterations. The four major cycles are; the initialisation (inception), the elaboration, the construction and the

transition phase. The application, in RUP, is built following a succession of incremental iterations. The second dimension (the vertical axis) on the contrary, represents the static aspect of the process defined in terms of activities.

Each phase must be terminated before starting a new phase. Boundaries are defined by RUP to be able to verify if the phase is finished or not. If design criteria are met, the phase is terminated, if not, the phase is reiterated.

The initialisation phase (inception) aims to identify the actors that will use the application as well as the interactions that will occur. This phase is supported by UML use cases. The results of the inception phase are a document proposing a general view of the project, draft versions of use cases (approximately 10% to 20% complete), an evaluation of time dedicated to each phase and to one or several prototypes.

The elaboration phase produces a description of the architecture of the software application, use cases that are approximately 80% complete, an executable prototype as well as detailed plans of criteria to evaluate each iteration. At the end of each phase, the software architecture must be fixed and will no longer be modifiable. This phase is the most delicate of the four because the software architecture is decided and defined prematurely in the method, and the choice of the architecture will bring constraints to future design choices.

During the construction phase the application is developed. For large projects, several phases of construction can occur in parallel. The application is integrated into the product following which testing is performed. After a successful test, the application is delivered and integrated with the required platform, a user manual and a description of the delivered version.

The final version, the transition phase is the delivery of the product to the client. This phase requires the development of new characteristics and the correction of certain errors following beta testing. The transition phase also includes training of users and of staff who will maintain the application. User feedback provides information on how to resolve installation usability issues and potential configuration problems

The static part of the process is based on the following four elements; the actors, the activities, the artefacts and the undertaking of operations. The actors define their responsibilities and the role of each. An activity is a task requested by an individual, this could be for example to test the application, or to find scenarios of use. An artefact is an element of information uses during the process; it is an input or output of an activity. An artefact can be, for example the source code, a model or a document. The operations resume the activities of each person and the sequence as well as the dependencies between the activities (which activity should be terminated before being able to start another). The undertaking of operations is modelled in UML by the collaboration diagram and/or the activity diagram.

The various activities carried out are characterised by nine processes represented in Figure 1-3, these are:

1. Business modelling
2. Requirements
3. Analysis and design
4. Implementation
5. Test
6. Deployment
7. Project management
8. Configuration and change management
9. Environment

RUP is an « off the shelf » method proposed by Rational who provide the method and support tools for the method. The tools provide the foundations for projects, sharing of documents between stakeholders, as well as support for the reuse of models. The approach is highly iterative. RUP is applicable to teams with more than ten people.

However the approach is related to Rational tools, and these tools which are almost essential have a significant cost. RUP is based on a process that decreases the quality of the technical aspects, i.e the code.

2.2.1 UML and RUP Section Summary

This section has presented the UML and RUP model-based design techniques. In UML, User Centred Design is not a central aim though the majority of existing systems are interactive, requiring some form of user interaction to function. This also means that UML is not suitable for designing safety-critical interactive systems, those we are interested in this thesis. Furthermore, UML gravely lacks formalisation, meaning underspecified, ambiguous and vague specifications. These are drawbacks of the UML. Although RUP promotes iterative design practices,

there is also still a need to improve the approach with methods and roles for User Centred Design principles as well as support for formal methods.

2.3 Design Models

The design of a usable, reliable and error-tolerant interactive safety-critical system is based on a mass of data of multiple natures from multiple domains. This section discusses the use of models to improve the gathering and refining of data for safety design of safety-critical interactive systems.

We promote the use of formal notations so that we can verify the properties of interactive safety-critical systems. Without such notations there is no means for designers to address reliability. However, formal notations may not be adequate for recording information that is idiosyncratically fuzzy and incomplete such as information gathered in the very early phases of the development process. Besides, it is important to note that in most cases, each model will be created by a different person with a different background within a different specialist domain which is likely to influence the kind notation they are able to master. Even for a system that is not safety-critical, it is still necessary to ensure the system's efficiency and reliability but this kind of issue is more salient for this type of system. To be effective, the models and analysis tools must consider the hardware, software, and human components in these [complex] systems (Leveson et al. 1997a). The models and approach proposed in this thesis aims to cover hardware, software and human aspects of the design process.

The following paragraphs provide an overview of the multiple models used in User Centred Design (UCD) approaches (see section 3). A number of which can be supported using the UML (Booch et al. 1999). For example the domain model is supported by class and object diagrams, and the application model which includes the commands and data for the application providers, are the main focus of UML. Some models are only partially accounted for. Task models and scenarios can be described informally and incompletely using UML use cases. Other models are not at all considered in UML for example, user model, platform model and presentation model (Bastide and Palanque 2003).

We hereafter present more precise information about some particularly relevant models for interactive systems design and highlight those which are particularly useful for safety-critical interactive systems design. Please note that since the core models of this thesis are the task model and system model, each are described in more detail in dedicated sections. Section 3.1 discusses task analysis while section 4.1 presents system modelling and notations.

Requirements Model: The functional and non-functional requirements of a system are defined in the requirements model. Requirements describe in a declarative way what a system is supposed to do. The description of a requirement model using a precise and un-ambiguous (i.e. formal) notion allows analysing the model and identifying errors or inconsistencies. In addition, tools can generate tests from the requirement models useful for verifying that a system behaves as the original requirements prescribe [(Palanque 1997) and (Campos and Harrison 1997)].

Task Model: A task model is a representation of user tasks (in order to reach a certain goal) often involving some form of interaction with a system, influenced by its contextual environment (Baber and Stanton 2004). Task models are used for planning and during various phases of user interface development for example. The models are usually developed by human factor's specialists following an extensive task analysis phase. For the design of interactive safety critical systems, task models can be advantageous for checking the properties of the future system.

User Model: A user model is a collection of information about a user and is a key component for providing flexibility and adaptation. They can incorporate generic information (valid over a wide range of potential users) such as (Card et al. 1983), (Fitts 1954; Barnard and May 1994) and represent information about perception, cognition or interaction. Other user models are aimed at representing information for specific users such as PUMA, (Blandford and Good 1997) and OSM (Blandford and Connell 2003) This information can be for instance, fed into a system model in the design phase in order to improve flexibility or in the evaluation phase in order to compute predictive performance evaluation (Palanque and Bastide 1997).

Environmental Model: An environmental or contextual model is developed by inspecting aspects of the environment of a current or future system. Information is gathered using techniques such as observation, documentation analysis or interviews. Examples of elements to be studied include location, temperature, artefacts, duration, social aspects and cultural ethics. The model can be used to identify causes of human

behaviour. Clearly, this can be beneficial for the development of an interactive safety critical system since contextual factors are one of the only ways to provide useful adaptation of the system to environmental changes.

Platform Model: A platform model includes a description of the platform and some platform specific characteristics. These models contain information regarding constraints placed on the UI by the platform such as the type of input and output devices available and computation capabilities. The model contains an element for each platform that is supported, and has attributes belonging to each element describing the features and constraints. Although this type of model is particularly useful for ensuring cross-platform compatibility of systems, they are critical when a given system is expected to be made available to several users working with different software and hardware environments.

System Model: The system model is, by far, the one that has been studied the most as it is the main raw material of system construction. In the field of interactive systems, most contributions come from the field of software engineering and have been more or less successfully adapted to the specificities of this kind of systems. Since the mid 80s several formalisms have been proposed that were addressing system modelling either at a very high level of abstraction (Dix and Runciman 1985), (Harrison and Dix 1990) (such as trying to capture the essence of interaction) or at a lower level in order to provide detailed modelling in order to support development activities (Paterno and Faconti 1992), (Palanque and Bastide 1990). Specific issues raised by interactive systems modelling include, system state, system actions, concurrency, both quantitative and qualitative temporal evolution, input device management, rendering, interaction techniques ...

Presentation Model: A presentation model details the static characteristics of a user interface, its visual appearance. The model contains a collection of hierarchically-ordered presentation elements such as sliders, windows and list boxes as far as WIMP user interfaces are concerned. For post-WIMP interfaces such graphical elements include icons, instruments etc. (Beaudouin-Lafon 2000) and (van Dam 1997). Current state of the art in the field of safety critical interactive systems is also addressing these issues. For instance, ARINC 661 specification [(ARINC 661a 2002)] provides a detailed description of interactive components and their underlying presentation platform for new generation interactive cockpits.

Architectural Model: An architectural model is a high level model of the application which describes the basic building blocks of the application. Examples of established architectural models are Seeheim model (Green 1985) which makes explicit the user interface part of the application and the Arch model [(Bass et al. 1991a)] which is an extension of the Seeheim model putting even more emphasis on the UI part. The Arch model divides all user interface software into the following functional categories, Functional Core, Functional Core Adapter, Dialogue, Logical Interaction and Presentation. From a modelling point of view, these components are usually dealt with individually. Various modelling techniques are applied to deal with these components and the following section address some of them i.e. domain model (related to functional core modelling) dialogue model and device model (a sub-part of the presentation component).

Domain Model: A domain model is an explicit representation of the common and the variable properties of the systems in a domain and the dependencies between the variable properties. (Czarnecki and Eisenecker 2000). The model is created by data collection, analysis, classification and evaluation. The term domain covers a wide range of interpretations, for example, the problem domain, business domain and the system/product domain. These models are necessary to understand the domain in which the future system will be built. In the field of safety critical systems the various domains involved (such as ATC and military systems) have already received a lot of attention. Domain models are readily available and are meant to be exploited before dealing with any system within that domain.

Dialogue Model: A dialogue model is a collection of hierarchically-ordered user-initiated commands that define the procedural characteristics of the human-computer dialogue in an interface model (Puerta 2002). Dialogue modelling has been regarded as a particularly hard issue to tackle. A lot of work has been devoted to it and the notations used have evolved in conjunction with interaction techniques. For instance, early work focussed on modal interaction techniques (Parnas 1969a) and evolved to WIMP interaction styles (Esteban et al. 1995) to reach recent and more demanding interaction techniques as in (Navarre et al. 2004) for multimodal interaction.

Device Model: Input and output devices are a critical part of the interactive systems as they represent the bottleneck via which the interaction between users and system takes place. Their behaviour is sometimes very complex even though it may be perceived as simple by the users. This complexity may lie in the device itself (as for haptic devices such as the Phantom (Massie and Salisbury 1994)) or in the transducers in charge of extending the behaviours of the devices (such as extending the behaviour of a mouse to cope with double or triple clicks

that embed temporal constraints) (Buxton and Myers 1986), (Accot et al. 1996). Device models can also be viewed as a person's understanding of how a device works (Satchwell 1997). In the field of safety critical systems describing the behaviour of such devices is critical as it makes precise the interaction techniques.

2.4 Model-Based Design Section Summary

In this section we have discussed the benefits and drawbacks of using model-based design. Though model-based design requires a significant degree of effort in terms of expertise for abstracting relevant information, employing dedicated notations and tools and understanding which model is most suitable for the representation of certain information, problems associated to inter-compatibility of design tools and additional time required to get started, its advantages outweigh these difficulties.

Models highlight important information, help to manage complexity and are a useful means for supporting methods. Furthermore, MBD advocates design reuse, verification and validation advantages at every stage of development and provides a means for separate subsystems to be

We have seen several significant models for HCI in this section, notably the task model, user model, domain model, context model, presentation model and dialogue model. This thesis will focus on the task model and system model, since we are interested in incorporating human factors in the model based design of safety-critical interactive systems.

The following section continues the theme of model-based design though will focus on a particular kind of model based design, directly related to user centred design. An approach which places the user at the centre of the process in order to achieve an interactive system

3 User Centred Design

Human-Computer Interaction and related disciplines have argued since the early days, that interactive systems design requires the embedding of knowledge, practices and experience from various sources. For instance, User Centred Design (UCD) (Norman 1988) advocates the involvement of human factors specialists, computer scientists, psychologist, designers etc in order to design useful and usable systems. UCD is a term applied to describe design processes in which end users greatly influence the evolution of the design. The end users are involved in some way throughout the design procedure to ensure that the end product is more user friendly to those it is intended to be used by. Norman's suggestions clearly place the user at the centre of the design process. Since Norman's introduction of the term UCD, further descriptions have been provided, more recently, Karat;

“For me, UCD is an iterative process whose goal is the development of usable systems, achieved through involvement of potential users of a system in system design”(Karat 1996).“I suggest we consider UCD an adequate label under which to continue to gather our knowledge of how to develop usable systems. It captures a commitment the usability community supports—that you must involve users in system design—while leaving fairly open how this is accomplished” (Karat 1997).

Furthermore UCD has been precisely defined and more information about it can be found in (ISO 13407 1999). This standard provides guidance on human-centred design activities throughout the life cycle of interactive computer-based systems. It states that UCD is an approach to interactive system development that focuses specifically on making systems usable. It is a multidisciplinary activity.

The following three principles, defined by Gould and Lewis (Gould and Lewis 1975) are the cornerstone of UCD: (i) early focus on user and tasks, (ii) empirical measurement (usability evaluation) and (iii) iterative design.

Methods for including the user throughout the design process include (in approximate order of use within the development process) surveys and questionnaires, field studies, focus groups, card sorting, mock-up interface concepts, usability testing and heuristic evaluations

Since we are primarily interested in model based design of safety-critical interactive systems, the following section presents task analysis and task modelling, a key procedure of UCD for capturing end users potentials tasks, actions and goals. Following this we briefly present usability analysis and stakeholder analysis which are relevant to our approach, Firstly because we believe usability together with reliability can increase safety.

Secondly, because information such as rules and regulations cannot be obtained from the users directly. A number of other stakeholders within the organization must define these.

3.1 Task Analysis

Within a model-based UI development methodology, the creation of the task model is a commonly agreed-upon starting point (Vanderdonckt and Puerta 1999). Tasks analysis is a central element of user centred design approaches. For this reason a lot of work has been devoted to it and to its integration in the development process of interactive systems. Analysing the tasks undertaken by current operators and the tasks to be accomplished by operators of a future system is necessary to ensure client requirements are met and satisfy the end users.

This section first provides an informal presentation of task modelling. A summary of task modelling techniques is then provided including descriptions of established notations and tools for task modelling. We then present the notation of task patterns.

Tasks should be analyzed before the interface is designed in order to allow the designers to make critical decisions about what actions the system should support.

A task model is a representation of user tasks often involving some form of interaction with a system influenced by its contextual environment. We use the word “influenced” (as opposed to “driven” by the environment) to highlight our thoughts on user’s having an underlying goal and hence plan in their mind before attempting to perform a task. This contrasts Suchman’s (Suchman 1987) theory of situation action. This theory analyses user behaviour through emergent actions of users during a particular activity. Situation action dismisses the role of predetermined intentions and goals of a user as part of the analysis. There is no intentionality in situation action since what happens is always developing ad-hoc out of the current situation. She proposes that plans are representations of situated actions produced in the course of an action. Therefore they become resources for the work rather than determine its course. This kind of assumption cannot help the design of a system because it is almost impossible to design a system without knowing the goals of users and their planned activities. While Suchman’s view may be true for gaming and leisure activities it is clear that in the application domain considered in this thesis, training goals, performance and task efficiency are critical to the correct and safe operation of the system.

Users perform tasks, which are structured sets of activities (1994) in order to achieve higher-level goals. Tasks can be further decomposed corresponding to lower level sub goals. This notion of decomposition naturally results in tree-like structures and thus hierarchical representation of the model. The typical characteristics of a task model include its hierarchical structure, the task decomposition and sometimes the temporal relationship between elements.

Task models are used for planning and during various phases of user interface development. The models are usually developed by human factor’s specialists following an extensive task analysis phase. For the design of interactive safety critical systems, task models can be advantageous for checking the properties of the future system. The following section presents a number of established task modelling techniques, including their notations and support tools.

3.2 Modelling tasks

The most dated task analysis technique is known as Hierarchical Task Analysis (HTA) (Annett and Duncan 1967). Models can be represented textually or graphically with structured numbering corresponding to the various goals and decomposed actions. Plans can then be drawn up indicating the order in which tasks are to be performed.

The **Goals, Operators, Methods and Selection rules** (GOMS) model was developed as a model for predicting human performance while interacting with a system. The original GOMS model, referred to as CMN-GOMS (Card et al. 1983), is the ancestor of a family of GOMS models that were refinements of the original model (Baumeister et al. 2000). Four popular versions of GOMS are the classic CMN-GOMS, the simplest Keystroke-Level Model (KLM), Natural GOMS Language (NGOMSL), and Cognitive-Perceptual-Motor GOMS (CPM-GOMS).

Méthode Analytique de Description de tâches (MAD) (Scapin and Pierret-Golbreich 1989) combines structured interviewing and modelling techniques focusing explicitly on the hierarchical relation of tasks. The main concepts of MAD include the task, action and structure.

Task Knowledge Structure (TKS) (Johnson and Johnson 1989) is based on the assumption that people possess knowledge structures in their memory that relate to tasks. The technique represents this conceptual knowledge to describe the roles, goals, plans and procedures comprised of actions and objects.

GroupWare Task Analysis (GTA) (van der Veer et al. 1996) was developed for the modelling of complex tasks in a co-operative environment. The foundations of the GTA approach are a combination of both ethnography (Jordan 1996) and activity theory (1995) adopting a clear distinction between tasks and actions.

User Action Notation (UAN) (1993) further developed in (1992) was designed to formalise the communication between the User Interface (UI) designers and the development team when specifying and analysing details of technology. The central concept is the abstraction of task, including hierarchical structure, and sequencing. User actions, interface feedback and state changes are used to build up the task description, which may subsequently be used as an action at higher levels of abstraction.

Diane+ (Tarby and Barthelet 1996) an extension of the earlier DIANE {Barthelet 1988 #890} formally models a task with three concepts: operation, sequencing and decomposition. DIANE+ like MAD employs a rich graphical notation to represent the decomposition of tasks as well as temporal and logical relationships between the tasks.

More recently, Paternò's **ConcurrentTaskTrees CTT** (Paterno 1999), a graphical notation with a dedicated support tool, CTT Environment (CTTe) have enabled the distinction of abstract, user, interaction and application tasks (see Table 1-1) as well as the possibility to model temporal aspects of activities when specifying a model. A set of operators, mainly taken from the LOTOS (ISO/IS 8807 1988) notation, is used to indicate the temporal relationships among tasks such as iteration, sequentially, concurrency, disabling, and recursion.

Table 1-1. CTT Task Types

Graphical Symbols	Description
	Abstract Tasks: Tasks that require complex activities whose performance cannot be univocally allocated.
	User Tasks: Usually they are important cognitive activities.
	Application Tasks: Can supply information to the user.
	Interaction Tasks: Between the user and the system.

Furthermore, CTT allows the specification of roles and objects of collaborative tasks, tasks involving more than one operator. Temporal operators (see Table 1-2) are used to link sibling tasks. The task models can then be simulated to study different possible paths of interaction.

Table 1-2. Operators used in the CTT notation

Notation	Description
T1 >> T2	Enabling
T1 []>> T2	Enabling with information processing
T1 [> T2	Deactivation
T1 [] T2	Choice
T1 *	Iteration
T1 [I] T2	Concurrency with information exchange
T1 > T2	Suspend resume
T1 T2	Independent concurrency
T1 (n)	Finite iteration
[T1]	Optional task

The operators provided within the CTT notation are described in Table 1-2. Figure 1-4a depicts the “choice” operator in which the interactive “Task” is performed by either interactive task 1 or interactive task 2 (the

diagrams are read from left to right). Figure 1-4b illustrates the use of an “abstract” task which is performed firstly by a user task, perhaps cognitive, which enables the interactive task.



Figure 1-4. a) Choice Operator, b) Enabling Operator with varied task type

There are also downsides to the CTT notation. The context and environmental conditions within which the activities are taking place are not considered when modelling tasks in CTT. Surrounding circumstances could have effects on the process. Also, details of artefacts being manipulated during tasks could be useful because the time spent performing a task could be dependent on the artefact in hand. The cognitive workload and users' current state is also not detailed. For example, conditions such as stress or tiredness or being under pressure may affect the way in which actions are performed with respect to their efficiency and effectiveness. CTT does not allow the designer to detail the type of low-level interaction taking place such as a mouse click or a keyboard entry. Furthermore, the “shift of focus”, the name we use to refer to the process of when users change their glance when interacting with an interface is not taken into account. This level of detail could be crucial in safety-critical interactive systems.

3.2.1 Collaborative Task Modelling

The complexity of the operation of many safety-critical interactive applications means that the dynamics of the process are distributed and goals can only be achieved if tasks are performed in cooperation among individuals, locations and automated systems

In addition to standard tasks models representing the behaviour of a single user, Paternò et al., (Paternò et al. 1998) have devised a method by extending the CTT notation and additional tool support for the modelling of cooperative applications that allows designers to describe the relationships between activities performed by various users involved in cooperative environments. Though the authors refer to their models as formal, we tend to disagree that any CTT model is formal. This is because they are largely under specified. A cooperative task is defined as “a task that requires activities from two or more users for its performance” (Paternò et al. 1998). After a user has performed a subtask belonging to a cooperative task, some reaction by one or more users is expected and necessary in order to complete the task. The method involves an additional cooperative tree which shows the temporal relationship between tasks belonging to different users. For each basic task the user who performs it is indicated.

The specification of a cooperative task must include the following:

1. Task name
2. Roles of users involved
3. Names of the sub-tasks which have to be performed by each type of user
4. The media used to communicate
5. Type of communication protocol (for example, synchronous or asynchronous, point-to-point or broadcast), this aspect is often related to the media chosen,
6. The objects which are manipulated to perform the task
7. Roles cardinality (how many users for each role are involved in the cooperative task)
8. Some additional informal comments that can be added to further describe the task or some of its main features.

Figure 1-5 provides an example of the modelling and representation of a cooperative task using the extended CTT notation and tool support. It could be argued that such a task is not cooperative since the two users do not share the same goal. One wishes to buy while the other wishes to sell. An application such as an Air Traffic Control (ATC) workstation and tasks to be performed in such an environment would be more appropriate for representing cooperative tasks.

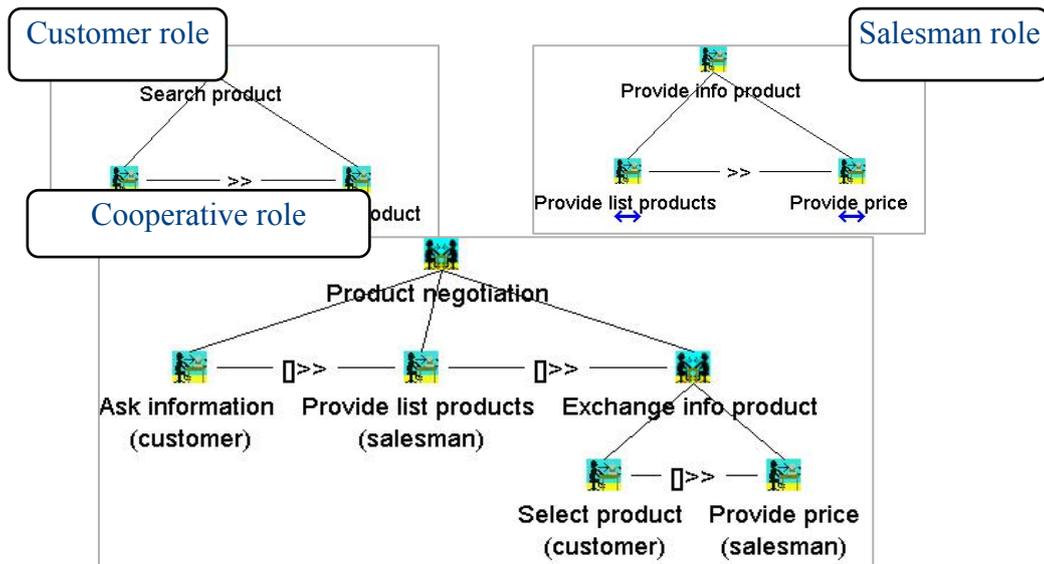


Figure 1-5. Simplified example of a cooperative task (taken from (Mori et al. 2002))

The discussed task analysis techniques have been summarised in Table 1-3 and Table 1-4. The table has been split in two, one dedicated to notations the other dedicated to tool support. Both aspects are important since the notation must be able to support the kind of information we are interested, but when dealing with large model manipulation, tool support is essential.

The notations have been evaluated according to criterion we believe our important for our research. The criterion are on two distinct levels, notation/language and tool support. Where insufficient data was available, we have used “?” to denote that the criterion has not been evaluated as positive or negative. Following our analysis of the state of the art in task analysis techniques, we agree that ConcurTaskTrees and its supporting CTTe tool are the most appropriate for our research. The negative points associated to CTT according to the table (denoted as “x’s”) are that it does not support textual annotations, it does not support quantitative user performance, though this does not interest us since we are not performing any user evaluations, the notation is not entirely human-oriented, which is a unfortunate because task analysis and the produced models are user centred. Also, it does not explicitly support the description artefacts being used, though it does allow (in the CTTe tool) specifying what type of platform the task will be carried out on (i.e. PDA, Desktop, Mobile etc.). The notation does not explicitly support the handling of human errors. However, its supporting tool, CTTe has been positively evaluated according to our requirements, providing model simulations, scenarios, and graphical model editing. Unfortunately, the tool does not currently support the notion of patterns.

Table 1-3. Summary of Task Analysis Notations State of the Art with respect to Languages

FULL NAME		Hierarchical Task Analysis	Goals, Operators, Methods and Selection rules model	GroupWare Task Analysis	Méthode Analytique de Description de tâches	Task Knowledge Structure	Task Analysis for Knowledge Descriptions	User Action Notation	Diane+	Visual Task Modelling Language	Concurrent Task Trees
AUTHORS		Annett & Duncan, 1967	Card et al., 1983	van der Veer, 1996	Scapin et al., 1989	Johnson, 1992	Diaper & Johnson, 1989	Hix & Hartson, 1993	Tarby and Barthelet, 1996	Brown & Leveson, 1998	Paternò, 1999
ACRONYM		HTA	GOMS	GTA	MAD	TKS	TAKD	UAN			CTT
NOTATION / LANGUAGE	Textual	√	√	x	x	√	√	√	x	√	x
	Graphical	√	x	√	√	√	√	x	√	√	√
	Cognitive	x	√	x	√	√	√	x	x	√	√
	Hierarchical	√	√	√	√	√	√	√	√	x	√
	Quantitative Performance User	x	√	x	x	x	x	x	x	x	x
	Order Relations	x	√	√	√	√	x	√	√	√	√
	Different Task Types	x	√	√	x	√	x	√	√	x	√
	System Oriented	x	√	x	√	x	x	x	√	√	√
	Human Oriented	√	√	√	x	√	√	√	x	√	x
	Collaboration	x	x	√	x	√	x	x	x	√	√
	Artefacts	x	x	√	√	√	x	√	√	x	x
	Formal	x	√	x	x	x	x	√	x	x	√
	Human Error	x	x	x	x	x	x	x	x	x	x
For Safety-Critical Int.Sys	x	x	x	x	x	x	x	x	x	x	

Table 1-4. Summary of Task Analysis Notations State of the Art with respect to Tool Support

FULL NAME		Hierarchical Task Analysis	Goals, Operators, Methods and Selection rules model	GroupWare Task Analysis	Méthode Analytique de Description de tâches	Task Knowledge Structure	Task Analysis for Knowledge Descriptions	User Action Notation	Diane+	Visual Task Modelling Language	Concurrent Task Trees
AUTHORS		Annett & Duncan, 1967	Card et al., 1983	van der Veer, 1996	Scapin et al., 1989	Johnson, 1992	Diaper & Johnson, 1989	Hix & Hartson, 1993	Tarby and Barthelet 1996	Brown & Leveson, 1998	Paterno, 1999
ACROYNM		HTA	GOMS	GTA	MAD	TKS	TAKD	UAN			CTT
TOOL SUPPORT	TOOL NAME	TaskArchitect (Stuart & Penn, 2004)	CAT-HCI (Williams 2000); CRITIQUE (Hudson et al., 1999); GLEAN3 (Kieras et al., 1995.); GOMSED (Wandmacher, 1997); QGOMS (Beard et al., 1996)	EUTERPE (van Welie et al., 1998)	IMAD (Gamboa et al., 1997)	ADEPT (Johnson 1993)	LUTAKD (Diaper, 2001)	Quantum (Hix & Hartson, 1994)	TAMOT (Paris et al., 2001); Isolde:U2 (T Lu et al., 1999); Isolde:IR (P aris et al., 2001); Isolde:T2T (Brasser & VLinden 2002)	SpecTRM Tool Suite (Lee et al, 2002)	EL-TM (Paterno & Mancini 1999); CTTe (Paterno, 1999); RemUSINE (Paterno & Ballardini 1999)
	Textual	√	x	√	?	?	?	?	√	√	√
	Graphical	x	√	√	?	?	?	?	√	√	√
	Re-use / Patterns Produces Scenarios	x	√	x	?	?	?	?	x	x	x
	Simulation	x	√	x	?	?	?	?	x	√	√

3.2.2 Task Modelling and Human Error

When modelling user behaviour, an error-free perspective is usually employed. It is usually during the testing phase of the system development cycle that errors are realised and taken into account. Task modelling as yet, does not allow for the description, representation and analysis of any unexpected eventualities that may occur including human error. Since the task model influences the system design it is important to understand how to manage and overcome possible errors.

3.2.3 Task patterns

Alexander, (1977) defines patterns as “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution”. Although this was described in the context of architecture the principles can be applied to various domains. The main underlying principle to the notion of patterns is that they are elements of design that have to be discovered from the analysis of current practice.

Software patterns first became popular with the object-oriented Design Patterns book (Gamma et al. 1995) and further studied in, (Buschmann et al. 1996). Since then, design patterns have been extensively promoted in the field of software engineering as they solved several known problems by providing reliable, usable and efficient solutions to them. More recently, patterns have been researched within the field of HCI. For example, interaction design patterns (Walldius 2003), usability patterns, (patterns that guide the practice of performing task analysis, writing scenarios, or articulating user requirements for particular systems) (Walldius 2003), pattern-based design for use within the UCD approach (Wesson and Cowley 2003) and finally, usability-oriented pattern language (Mahemoff and Johnston 1998) based on popular style guides such as Nielsen’s heuristic guidelines (Nielsen and Molich 1990). Figure 1-6 illustrates an extract of the interactive activity pattern, “iterate” taken from the paper passed on such design principles.

<p>Name: Iterate</p> <p>Motivation: The Select pattern suggests that one way to select objects is to iterate through them.</p> <p>Context : This pattern is applicable wherever the user can select from a finite set of objects.</p> <p>Forces:</p> <ul style="list-style-type: none"> • The user may be iterating to find a particular object, or to process a sequence of objects one at a time (i.e. process, select next item, process, etc.). • The user may want to iterate through a subset of the objects, and at the same time be free to jump to an out-of-sequence object. • The computer must be provided with a way to order the objects. This sequence should be predictable to the user. • The user should generally be able to start iterating from any object, and this means that the sequence should be circular, i.e. “wrap around”. It is usually desirable to iterate forwards and backwards, possibly to let the user Return to Earlier State. <p>Solution: Consult users to determine the logical sequence for a particular set of objects. Even if the order is not immediately obvious, it should at least be predictable. Try to make the iteration modeless, so that the user can easily switch between processing and iterating, and can also jump to another object easily.</p> <p>...</p> <p>Examples: Editing a sequence of documents prepared by someone else; stepping through months on a calendar; activating “Search next” command; iterating through items on a GUI menu; channel-surfing through television stations; reviewing command-line history.</p>

Figure 1-6. Extract of the interactive activity pattern, “iterate” (taken from (Mahemoff and Johnston 1998))

Task patterns for interactive systems design is a relatively new concept with the aim of solving design problems using existing knowledge of previously identified patterns and solutions. The majority of research to date focuses on user interaction with software and interfaces providing interface design patterns and task based

patterns to improve usability rather than task based patterns that focus on user behaviour intended for testing the compatibility with system design.

Indeed, the focus is mostly on what we could call generic user behaviours. Such work proposes then a set of user interface design solutions to such user behaviours.

Task patterns were first introduced by Breedvelt and Paternò et al., ((Breedvelt et al. 1997) & (Paterno 1999)) as reusable structures for task models. The patterns are described as hierarchical structured task fragments that can be reused to successively build the task model. In (Breedvelt et al. 1997) the authors state that reusable task structures are intended to help designers building task models for large systems to speed up the application design. The reusable structures proposed in the paper are based on the study of an industrial application, they include the following:

- Multi-values Input Task: editing values and confirming them
- Search Task
- Evaluation Task: consists of two activities, selection of the data to be evaluated and the evaluation itself, which can be repeated until the user decides to stop.
- Recursive Activation Task: allows the user to interrupt their current task at any point in order to create a new instance of their current task.

Reusable structures are then used to create task models. One would imagine that reusable structures would be “plugged in” to an existing task model, but the paper shows that the final task model is based on the structures, and then extended according to specific tasks available on the application under design. This view differs from ours, since we propose to begin with a task model and extend it with patterns. Not only do we handle task patterns in a different way, we also propose the use of task patterns to deal with erroneous user behaviour. To make explicit interactions and tasks that could potentially be erroneous, with a higher level goal of testing a system model to see how it would deal with such errors.

Work presented in (Sinnig et al. 2003) focuses on establishing and integrating patterns as building blocks for the creation of the task model in order to merge software and usability engineering. The authors propose patterns using the UML and CTT notations. They define task patterns as follows; “Task Patterns describe the activities the user has to perform while pursuing a certain goal. The goal description acts as an unambiguous identification for the pattern. In order to compose the pattern as generic and flexible as possible the goal description should entail at least one variable component. As the variable part of the goal description changes, the content solution part of the pattern will adapt and change accordingly. Task Patterns can be composed out of sub-patterns. These sub-patterns can either be task patterns or feature-patterns” [applied to the user-task model describe the activities the user has to perform using a particular feature of the system]”. (Sinnig et al. 2003).

The “feature patterns” indicate that the task model is heavily dependent on the system. Whereas we believe a task model and therefore a task pattern should be purely related to the user task and not to the system they will be using. Though the “feature patterns” shown in Figure 1-7 appear to be examples of non-system dependent patterns.

A four-stage strategy for the process of pattern application is also detailed. Steps include Identification, Selection, Adaptation and Integration. Such task patterns are context specific and problem centred as opposed to guidelines which can often be too simplistic, too abstract and difficult to interpret. The task patterns proposed only model error-free behaviour, that is, the possibility of human-error would be considered later in the design lifecycle during the testing phase of the system for example.

The proposed four-step approach for applying task patterns can be summarised as:

1. Identification: of a sub-tree for pattern application within an existing task
2. Selection: A appropriate pattern is selected for application
3. Adaptation: The pattern is adjusted to the current context of use.
4. Integration: The sample solution will be integrated into the current task model resulting in a modified task model. This can affect the task tree in 2 different ways.
 - a. Adding a new branch to the tree: This occurs if the pattern introduces a new feature that did not exist before

- b. Modifying an existing branch: The application of the pattern re-shuffles the particular branch of the task tree in order to improve the arrangement of the user activities with respect to a certain goal, the user has.

The example provided in the paper, and the approach in general applies to basic interactive applications, for example an online shopping site. The authors show how task patterns can be applied to a typical user task, such as “find”. Figure 1-7 illustrates the “find” task pattern modelled using UML. The pattern is composed of a browse, search and agent feature patterns.

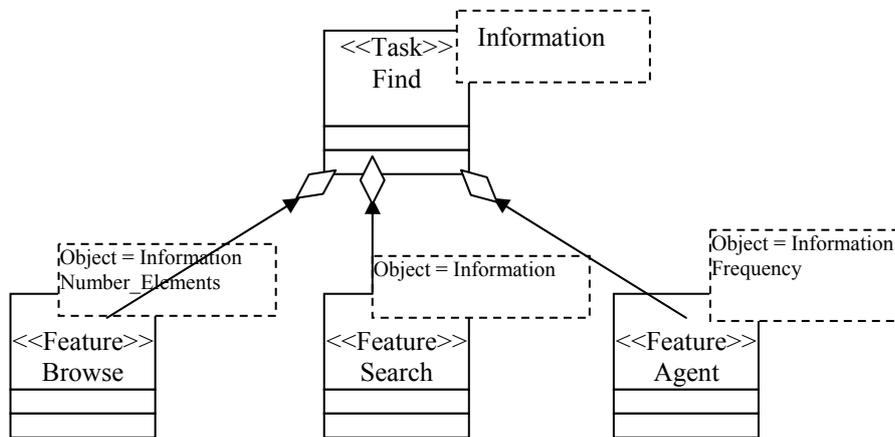


Figure 1-7. Find pattern and its sub-patterns modelled in UML (taken from (Sinnig et al. 2003))

The pattern is then applied to a task model, in this paper the authors use the CTT notation. Furthermore, the authors propose tool support, called the Task Pattern Wizard which runs through the pattern tree and questions the user each time it encounters a variable, which has not been resolved yet.

Work presented in (van Welie M et al. 2000) also addresses the issues raised by user interaction stating that patterns for user interface design should help make systems more usable for humans.

We suggest that task patterns can be used as a means of incorporating erroneous behaviour into a standard task model as a method for making explicit human “errors”. This will be explained the sections on our contributions in Chapter 4.

3.3 Usability Analysis

Heuristic Evaluation (HE) (Nielsen and Molich 1990) is an established Usability Inspection Method (UIM) in the field of HCI (Human-Computer Interaction). The analyst, who must have knowledge in HCI, follows a set of guidelines to analyse the user interface. The technique is inexpensive, easy to learn and can help predict usability problems early in the design phase if prototypes are available. Examples of HEs include (Nielsen and Molich 1990) (listed below), (Pierotti 1995) and (Gerhardt-Powals 1996).

1. Visibility of system status: The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. Match between system and the real world: The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
3. User control and freedom : Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
4. Consistency and standards: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5. Error prevention: Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
6. Recognition rather than recall: Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

7. Flexibility and efficiency of use: Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. Aesthetic and minimalist design: Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
9. Help users recognize, diagnose, and recover from errors: Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
10. Help and documentation: Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

In addition to Nielsen, who has been a prominent figure in the HCI domain, Shneiderman and Norman have also contributed. For example, Shneiderman's (Shneiderman 1998) three principles consist of the following, 1) Recognize the Diversity (usage profiles, task profiles, and interaction styles), 2) Use the Eight Golden Rules of Interface Design and 3) Prevent Errors (correct matching pairs, complete sequences, and correct commands). Furthermore, Shneiderman's eight 'golden rules' for interface are:

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialog to yield closure
5. Offer simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

As mentioned, Don Norman has contributed extensively to the field of HCI an interface design. Similar to Nielsen's heuristics and Shneiderman's golden rules, Norman suggests for example to, make it easy to determine what actions are possible at any moment (make use of constraints) to "make things visible, including the conceptual model of the system, the alternative actions and the results of actions" and "make it easy to evaluate the current state of the system"(Norman 1990).

3.4 Stakeholder Analysis

In addition to task analysis, stakeholder analysis is an essential component to User Centred Design. The previous section argued for the importance the user and the understanding of users' tasks have on the design of a usable interactive system. Though the design of a system, and in particular a safety-critical interactive system requires input from many people who share an interest in the future design. For example, business rules and design regulations cannot be gathered from users directly. Such people are known as stakeholders. A more appropriate definition is provided by Clarkson (Clarkson 1995)p.106 who describes stakeholders as, "persons or groups that have, or claim, ownership, rights, or interests in a corporation and its activities, past, present, or future. Such claimed rights or interests are the result of transactions with, or actions taken by, the corporation, and may be legal or moral, individual or collective". Stakeholder analysis originates from business and management science though is now used in many other domains such as information systems, marketing and production etc.

Involving as many stakeholders as possible will help ensure that the design will meet the requirements of all interested parties. Typical stakeholders for a safety-critical interactive application for an Air Traffic Control (ATC) workstation would include, ICAO representative, ATC users, ATC "head of operation rooms", finance and maintenance directors, system developers, certifiers, manufacturers. Such stakeholders have requirements, which can be grouped into data-centric requirements, process-centric requirements and people-centric requirements.

Table 1-5. Requirements Types and Stakeholders

Stakeholder Requirements Type	Stakeholders
Data-centric: Content, retention, reproduction, security, privacy	Certifiers
Process-centric: Activity, ordering, timing, documentation	Certifiers, Management, maintenance directors, system developers, manufacturers
People-centric: Accountability, and responsibility, knowledge and believe, and education and training	Certifiers, ATC users, ATC “Head of operation rooms”,

Thus, though we understand the importance requirements capturing and stakeholders have on the design process, we do not focus on these issues but discuss them briefly where appropriate. To be more specific, requirements engineering is part of the Integrated Modelling Framework but has not been studied. We also touch on certifiers when we consider the ARINC 661 Cockpit Display System Standard Specification in Chapter 9, which must be satisfied in order for certifiers to certify new generation interactive cockpit applications.

4 Model Based Interactive Systems Engineering

Model based approaches for interactive systems engineering promote the use of models for the design, specification and validation (sometimes through verification) of interactive systems. Figures in this section position related work in the field according to three criteria:

- The evolution (according to time) of the underlying formal description technique used for describing interactive application,
- The part of the interactive application that has been explicitly addressed by the notation. These parts correspond to elements of architectural models like Seeheim (Green 1985) and Arch (Bass et al. 1991b). Sometimes a notation would have been able to address a wider part of the interactive application but we only represent here what has been made explicit by the authors in the referenced paper,
- The type of interaction technique that has been dealt within the referenced paper.

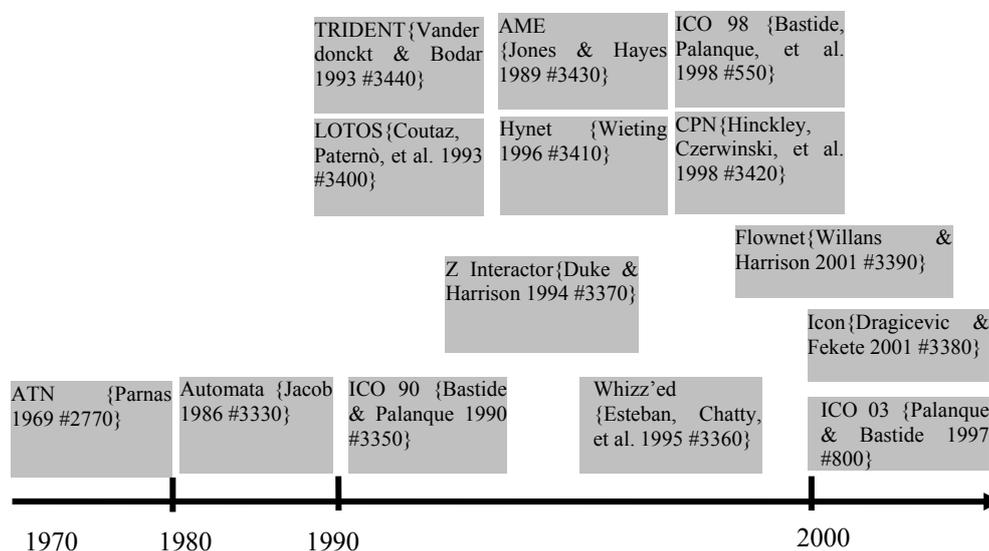


Figure 1-8. Model-based approaches and notations for Interactive Systems engineering.

Since the late sixties a lot of work has been devoted to the definition of notations and tools to support interactive systems design at a higher level of abstraction than implementation code. Figure 1-8 shows some approaches that have been proposed for the modelling of interactive systems over the years. As the complexity of interactive application was growing, the expressive power of the underlying formalisms has also been increased. Indeed early work from Parnas (Parnas 1969b)] based on finite state automaton was only able to model application featuring a small number of states. In order to deal with large scale applications this restriction has been removed by the use of more expressive notations like Petri nets (Petri 1962). Over the years, other notations based on finite state automata have been proposed such as statecharts for describing the behaviour of interactive

components but as stated by van Biljoen (van Biljon 1988) "... some investigation into extensions of finite state machines to enable them to provide this power led to the realisation that Petri nets already do this".

4.1 System Modelling & Notations

This section discusses the use of formal notations and modelling techniques for the specification of interactive systems. We will provide a brief overview of the concept of notations and modelling before describing established interactive systems modelling techniques from the literature.

A *formal language* is a set of symbols with formation rules that guide the structure of sentences and inference rules that define the manipulation of the symbols. Importantly, a formal language has a well-defined semantics (Burns 2000). A *model* is a set of concepts with relationships between those concepts. An Entity Relationship Model is an example of a model. Its concepts include entity types, relationship types, attributes etc. A relationship between concepts is "a subset of the Cartesian product of two entity types". An entity e1 cannot have more than one relationship with another entity e2 through the same relationship type.

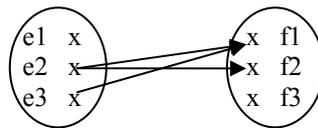


Figure 1-9. Example of an Entity Relationship Model

A model provides completeness (the ability to be able to express all information desired), consistency (contradictory elements cannot be expressed) and most of the time, generality (independence from a given application domain).

In addition to the given description of a model, a formalism dedicated to a Model can be defined as a set of conventions for representing the concepts of a Model. Conventions include lexical elements (graphical or textual), concrete syntax (separators, terminators etc), and a formal definition of these conventions, otherwise the formalism is simply called a notation. A formalism can provide expressiveness, conciseness and closeness of the representation to the application domain and completeness with respect to the Model.

Formal methods is a discipline for studying how mathematical models of systems can be used to develop efficient, reliable, and safe designs (Heymann and Degani 2007). The methods can be used to express specifications, requirements and to perform systematic analysis and verification of models.

We aim to model systems to provide an abstract description of the system independent from the implementation. Such models describe the outputs of the system according to the inputs it receives and provide a means for facilitating discussion between various stakeholders.

Parnas (Parnas 1969b) was one of the first to use formal methods, and in particular finite state machines (FSMs) (see 4.1.1 below) for the specification human computer interaction issues. His work focused on the dialogue within an interactive system.

The potential benefits of formal methods are accepted (Hinchey and Bowen 1995) and a number of standards bodies are recommending their use in security and safety-critical systems (Bowen 1993). Furthermore, Heymann and Degani (Heymann and Degani 2007) argue that an important facet of formal methods is to prove that a given model of the system fulfils certain design criteria, or properties. In this context, a *property* can be a simple statement about something that the system model does (or does not do).

Interestingly, just over ten years ago, Bowen (Bowen and Hinchey 1995) provided a list of ten rules, known as the ten commandments of formal methods which are listed below.

1. Thou shalt choose the appropriate notation
2. Thou shalt formalize, but not overformalize
3. Thou shalt estimate costs
4. Thou shalt have a formal method guru on call
5. Thou shalt not abandon thy traditional development methods
6. Thou shalt document sufficiently
7. Thou shalt not compromise thy quality standards
8. Thou shalt not be dogmatic

9. Thou shalt test, test, and test again
10. Thou shalt reuse.

Leveson (Leveson 2003b) agrees with some of the above commandments when stating models must be readable without graduate level training in discrete math, and mathematical proofs must be understandable by application experts. These are valid points, though one must bear in mind that the models will in most cases be designed by experts with training in the field of formal specification. We agree however, that the models should be readable by those stakeholders concerned. This is a reason that tools that provide simulations are useful for explaining models to less experienced stakeholders.

The list of commandments is very much focused on selecting the most appropriate language for the system to be specified. The characteristics of the language should match the characteristics of what must be modelled. For example, when modelling tasks, it may be appropriate at times to use a simple language such as Hierarchical Task Analysis (HTA) (Annett and Duncan 1967), though for a cognitively demanding task, a language such as Task Knowledge Structures (TKS) (Johnson 1992) would be more appropriate to capture specific details. We have shown in section 3.2 for the task modelling part of the proposed approach in this thesis, ConcurTaskTrees (CTT) (Paterno 1999) is the most appropriate notation for us. While for the system modelling, we have also selected the most appropriate language for our requirements, which is Interactive Cooperative Objects (ICOs) (Navarre 2001). From an industrial perspective, selecting a specification language may rely more heavily on what would impress potential clients, methods that are mainstream and that they are used to.

Burns (Burns 2000) highlights that at present, the use of formal methods is largely restricted to initial system design, and their use later in the system development lifecycle is much less common. Though we show in this thesis that formal modelling of accident reports, human error analysis, barrier analysis are examples of formal methods can be used later in the lifecycle even after a system has been designed. Though of course it is preferable to analyse and model such data as early as possible to improve the design of the system.

As well as demonstrating the benefits offered by formal specification of safety-critical interactive systems, this thesis addresses practical concerns relating to the proposed approach, for example scalability and cost-benefit issues. Bowen et al (Bowen et al. 1996) state that many formal methods lack scalability, the ability to represent large, complex systems without losing structural clarity. We account for this issue and highlight the benefits of certain newly developed tools which have been specifically designed to support scalability when modelling large interactive applications.

4.1.1 Formal specification techniques for systems modelling

Safety critical systems have been for a long time the application domain of choice of formal description techniques (FDT). In such systems, where human life may be at stake, FDTs are a means for achieving the required level of reliability, avoiding redundancy or inconsistency in models and to support testing activities. The perceived cost of the application of FDTs is justified by the increase in confidence in the correct functioning of the system that they can provide. The use of formal languages and methods is a common means to gain high confidence in the accuracy of information in the field of safety-critical system engineering (Burns 2000).

A noticeable trend in the safety-critical systems domain, particularly in the field of aviation is the increase of highly interactive user interfaces, such as airplane cockpits, air traffic control workstations, nuclear plant control systems, etc. Interactive cockpits have been embedded in the new Airbus A380 and will be implemented in the new Boeing 787. In turn, the appearance of such advanced user interfaces emphasises the need to properly account for human factors in these new systems, and to ensure their usability as well as their reliability.

In addition to the standard advantages of using FDTs which provide complete, concise and non-ambiguous specifications, the formalisms are more beneficial if they allow both verification and validation of system specifications.

Hix and Hartson (1993) provide simple definitions of verification and validation. Verification answers the question "Are we building the system right?". Does the system verify crucial properties? With the help of FDTs, such questions can be answered through mathematical analysis of the specifications. Validation answers the question "Are we building the right system?". Does the system meet the requirements? Is it usable and useful? When the emphasis is on validation, iterative design processes

Formal verification of models can only be achieved if the formalism used is based on mathematical concepts. This allows proofs to be made on the system model in addition to empirical testing once the system has been implemented. Verification of models has economical advantages for resolving problems in terms of consumption

of resources. Further advantages include the possibility to locate an error (though perhaps not the exact causes for the error).

The aim of the formal analysis is to prove that there is no flaw in the models. Using the ICO formalism for describing the models, the analysis is done by using the mathematical tools provided by the Petri net theory. Using those tools, one can prove general properties about the model (such as absence of deadlock) or semantic domain related properties (i.e. the model cannot describe impossible behaviour such as the light is on and off at the same time).

Modelling systems in a formal way helps to deal with issues such as complexity, helps to avoid the need for a human observer to check the models and to write code. Allows us to reason about the models via verification and validation and also to meet three basic requirements notably: reliability (generic and specific properties), efficiency (performance of the system, the user and the two systems together (user and system) and finally to address usability issues.

In the following paragraphs, we present several FDTs classified as either state-based formalisms and event-based formalism.

4.1.1.1 State based formalisms

State based formalisms provide a description of all of the states a system can be in and the transitions between these available states. By doing so, these formalisms allow us to take into account the interaction context and are therefore more adequate for modelling interactive applications within which the user actions play an important role. The following provides a brief overview of state based formalism.

Finite state machines

FSMs or finite automaton are a mathematical formalism which offer a graphical representation of models. The models represent the behaviour and the modelled system composed of states, transitions and actions. FSMs can be represented using a state diagrams or state transition diagrams as shown in Figure 1-10. Such graphical representations are graph-oriented within which states are represented as circles and the operators which change the states are the labels on the arcs between the states. A state stores information about the past by reflecting the input changes from the system start to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment.

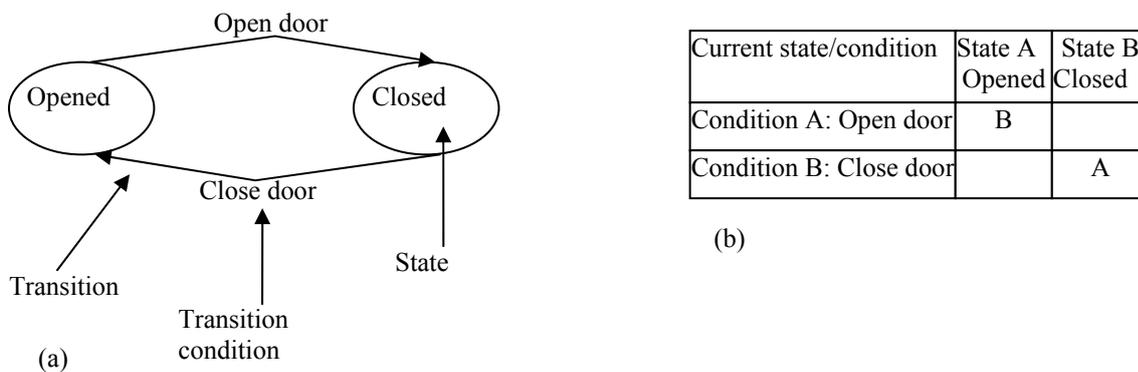


Figure 1-10. Example of a (a) finite state machine and a (b) state transition table

Extensions to this formalism have been made to deal with its constraints. For example, Recursive Transition Networks (RTN) (Woods 1970) providing hierarchical relationships between state machines, Augmented Transition Networks (ATN) also defined by Woods (Woods 1970) authorizing the use of preconditions via variables on arcs. Generalized Transition Networks (GTN) proposed by Kieras (Kieras and Polson 1985) where preconditions, actions and states can be GTNs.

The above mentioned extensions to state machines have been used for many years and are still used within many object oriented design methods.

In the domain of Human-Computer Interaction, their use is also important, either as a formalism for specifying and designing the interface, or simply as a formalism for specifying the dialogue part.

The advantages of finite state can be summarised as follows:

- They are based on a representation of system states which favours the modelling of a system in terms of states and state-changing operator
- They provide a relative static validation of models
- They offer a graphical representation allowing a clear and precise vision of models
- They allow a simulation of the behaviour of the system by executing the models. In the domain of HCI, numerous algorithms are available for automatic generation of an interface prototype from its finite state machine specification (REF Wasserman 86, Green 86)
- The extensions made to finite state machines, RTN, ATN and GTN allow the representation of an aspect of the structure of the system.

The disadvantages of finite state machines and their extensions can be summarised as:

- The representation of systems with a large number of states is often difficult to read
- They do not allow the representation of the real structure of the system
- The extensions, GTN and RTN are not a mathematical formalism but have been added « bit by bit » to be able to model systems that are difficult to represent using finite state machines. This lack of formalism makes it impossible to statically validate the models and at times their execution
- State machines and their extensions do not allow parallelism and synchronisation of transitions.
- State machine and their extensions require the description of each accessible state of the system and the operators available from each of these states. In the domain of HCI this constraint is critical, in particular in the case of user interfaces. It may be desirable to represent the non-available states from a given state.

Statecharts

Statecharts is a formalism developed in 1987 by Harel (Harel 1987) for the graphical modelling of complex systems in which concurrence is important. Statecharts extend traditional FSMs with useful characteristics such as the modeling of superstates (providing hierarchy), and concurrent behaviour. They have become used widespread with a variant of them being part of the UML. Their mathematical foundation allows us to envisage (at least partly) model validation.

Classic state diagrams are so called "or" diagrams, because the machine can only be in one state or the other. With Harel statecharts it is possible to model "and" machines, where a machine is in two or more states at the same time (see Figure 1-11). This is due to the possibility of having superstates. In Figure 1-11, superstate S1 is comprised of states S2 and S3 simultaneously. States S2 and S3 are OR operations, whereby S2 is either in state A or B and state S3 is in either state C, D or E. Default states are identified by an incoming arrow. In this example, states B (a substate of state S2) and state E (a substate of state S3) are those which will be activated on a transition to that state unless the transition has an explicit specified entry state.

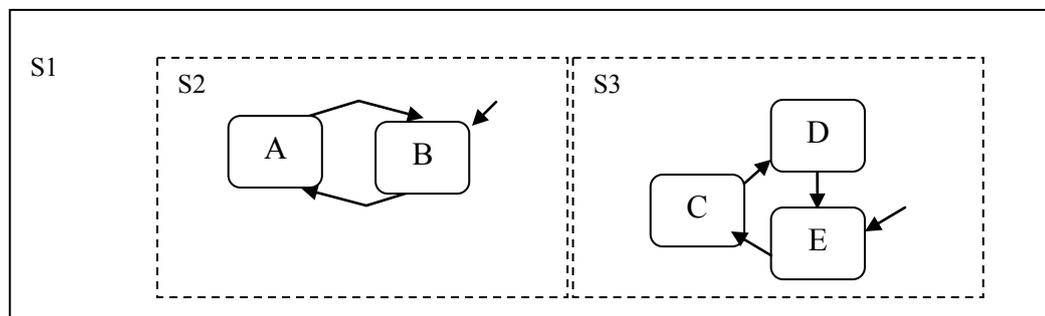


Figure 1-11. Example of a statechart

Statecharts propose two types of objects, states and events. Graphically the states are represented by rounded rectangles while events are arcs labelled with the name of the event and the condition that authorises this event. There are several types of arcs:

- Standard arcs from the perimeter of one state to another state on the same level

- Hierarchical arcs from the interior of a state to the interior of another state on a different level (superior or inferior).

To take into account this notation of hierarchy, further graphical representations have been introduced. It is possible to trace the “history” of a state. Once a state is over, it is possible to save its value by writing an “H” in that state. The H signifies that the state in which the model has been terminated is stored and that returning to this state will not be done using the default arcs, but by putting the current state in the stored state. A H* ends the restoration of the group of sub-states. It is also possible to delete the history to signify that the history of the state E1 is lost and that the return to this state can only be achieved using default arcs.

Statecharts have been used to model graphical interfaces by (van Zijl and Mitton 1991) but without demonstrating how to verify the possibilities of mathematical validation of the models.

However, certain StateCharts constructions, such as those related to the history, are difficult to analyse. Also, these graphical constructions reduce the readability of models because the dynamics of the models are no longer represented by arcs but also with textual notations inside the states. Model validation is never practically envisaged. This inconvenience highlights the advantages of Petri nets in comparison to StateCharts, which uses linear algebraic to look into model properties.

Statecharts as originally proposed have not been adequate for certain tasks such as describing navigation over web applications (Winckler 2004) and needs specific to safety critical systems design (Tsai 2004), such as representation of risks posed by system states, equipment failure and provision of additional safeguards. This has led to the proposal of many different Statecharts variants to overcome such inadequacies (von der Beeck 1996). The two variants made to satisfy the above inadequacies are StateWebCharts (SWC) (Winckler 2004) and SafeCharts (Dammag and Nissanke 1999b). Safecharts will be discussed in more detail due to its relevance to the thesis, though briefly, SWC is a formal notation that extends Harel’s StateCharts (Harel 1987) by adding necessary concepts for describing navigation in the context of Web navigation. SWC adds appropriate semantics for states and transitions, includes notions such as dialogue initiative control and client and transient activities. Since SWC is a formal notation based on StateCharts, it eases the description of non-ambiguous requirements. In addition, it is possible to apply formal verification and check system properties (e.g., is it always possible to reach pages in one click, is it always possible to come back to the home with less than three clicks, etc.) as it is done with StateCharts models (Winckler 2004).

SafeCharts

Safechart (Dammag and Nissanke 1999b) is a variant of StateCharts intended exclusively for safety-critical systems (Dammag and Nissanke 2003). Additional features include explicit representation of risks posed by different hazardous states, a separation of functional and safety concerns, a representation of component failures and characterization of transitions based on the nature of their risk. Safecharts divides the representation of any system into functional and safety layers to highlight safety issues; both using StateCharts.

The risk ordering relation can construct the risk graph, exclude those illegal transitions between risk non-comparable states, and solve the conflicting transitions by assigning different priorities according to the risk distance (Tsai 2004).

States in Safecharts are ordered according to their relative risk level. In diagrams, higher risk states are placed higher in the SafeCharts than lower risk states.

In SafeCharts, a transition is fireable if the following conditions are satisfied:

- The source and target state of the transition must be risk comparable
- The transition has only one source state and one target state
- The transition does not cross the internal boundaries of an AND state which is a common ancestor state of its source and target states
- The source state and the target state of the transition must be not ancestrally related

Originally, SafeCharts did not cover verification issues of models. Though (Tsai 2004) has proposed a method for doing this. The author demonstrates how to check for deadlocking properties of a SafeCharts model and also propose a verification method by means of model checking. The method uses model translation of SafeCharts into Extended Timed Automaton (ETA) for the model checking and state graph detection mechanism. The ETA is fed into the verification CASE tool, State Graph Manipulator (SGM).

4.1.1.2 Event-based formalisms

Event-based formalisms, in contrary to the previously presented state based formalisms often have a relatively informal definition and are not based on mathematics. They are grounded on three concepts: events, event management and event processing procedure.

Here we will briefly provide definitions of event-based formalisms and the associated philosophies without discussing recent improvements, which justifies the fact that some references here are dated.

An event is the initiation of an operation that is defined by a type and a set of parameters. This operation can change the system state, create new events or trigger any execution. The management of events is done via a queue of events. A dispatching mechanism forwards them towards procedures that are capable of taking them into account and treating them. Each procedure handling an event (call event-handler) processes the event according to the, the current state of the system and the parameters of the event. Event-based models are widely used and have been subjected to numerous extensions (Carlsen et al. 1989), they also form the foundations of numerous information systems design methods. Certain attempts have been made to formalise event-based formalisms such as (Alexander 1987), (Green 1986) but have not achieved the main goal: model validation. A detailed and structured case study of various event based models is presented in (Petoud 1990).

4.1.1.3 Petri nets

Petri nets (Petri 1962) are a widely used formal description technique in systems engineering. A Petri net models a discrete event-based system using state variables, the four elements that the formalism are comprised of are, the places (depicted as ellipses) and a number of state changing operators. These are transitions (depicted as rectangles). The state of the system is modelled by the distribution of tokens in the places of the network (these tokens represent the value of state variables). The places and transitions are related by arcs which represent the necessary pre and post conditions for state changes and the state changes themselves (see Figure 1-12).

Petri Nets were developed to support the engineering of concurrent systems. It is possible to model and visualize behaviours comprising concurrency, synchronization and resource sharing.

Figure 1-12 provides an example of a Petri net, which can be read as follows:

- Place p1 contains a token and place p2 does not. The current state of the system can be described as: state = {M(p1)=1; M(p2)=0} where M(p) represents the quantity of tokens held in place p;
- The transitions t1 and t2 need at least one token in place p1 and p2 respectively in order to be executable (this is known as firing a transition);
- Since place p1 contains a token, the transition t1 can be fired by removing the token from place p1 and placing a token in place p2 (see Figure 1-12b).
- After the firing of transition t1, the place p2 contains a token making transition t2 fireable.
- The firing of transition t2 is translated by the deposit of a token in place p1, making transition t1 fireable etc.

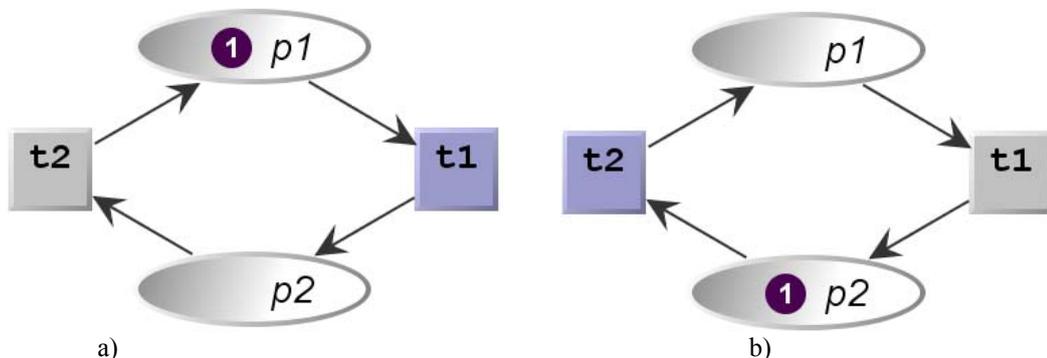


Figure 1-12. Example of a Petri net with two states

In addition, with respect to, basic Petri nets, a place can contain more than one token, enabling the specification of concurrent behaviours. Furthermore, different types of arcs exist, including inhibitor arcs and test arcs which provide more expressive power. An inhibitor arc is presented in Figure 1-13.

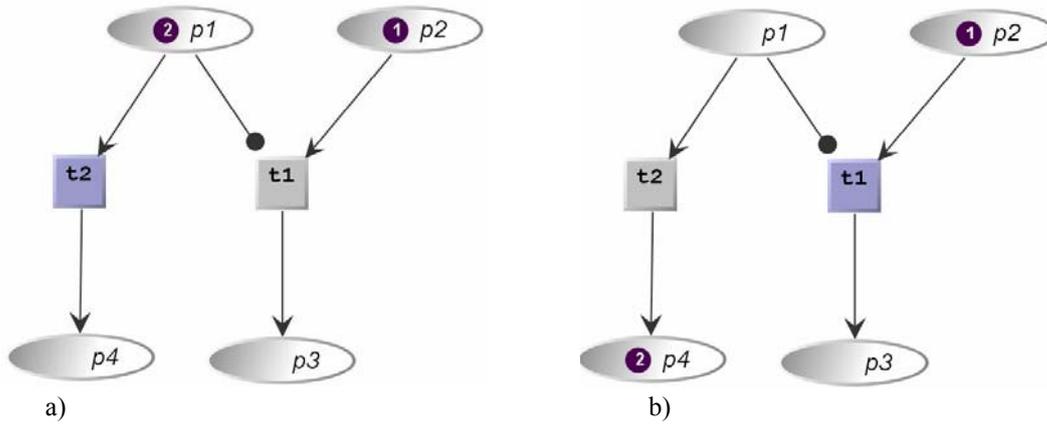


Figure 1-13. Basic Petri net with an inhibitor arc, a) before firing of transition t1, b) after firing of transition t1

In the example in Figure 1-13, the inhibitor arc behaves by preventing t1 from being fireable unless place p1 contains no tokens, and place p2 contains at least 1 token.

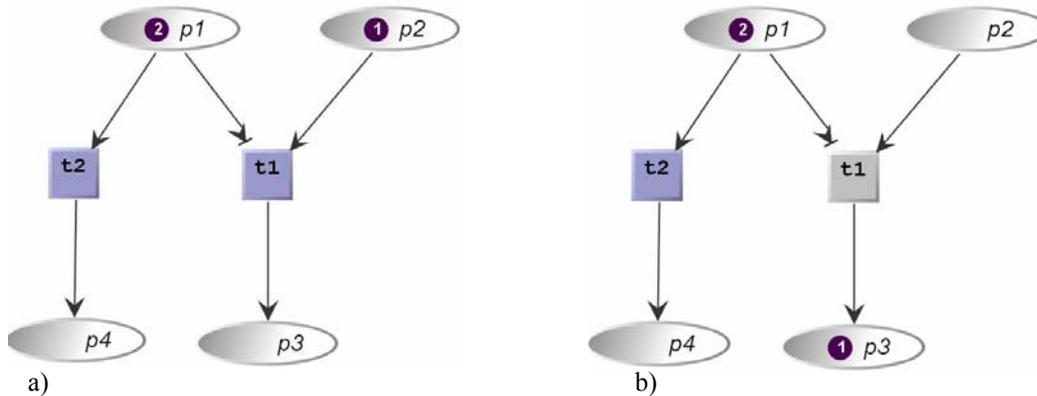


Figure 1-14. Basic Petri net with a test arc, a) before firing of transition t1, b) after firing of transition t1

Figure 1-14 presents the same Petri net as that in Figure 1-13 except this time, the inhibitor arc has been replaced by a test arc. The test arc changes the behaviour of the network. Transition t1 is fireable if there is at least one token in place p1 and at least one token in place p2. However, the token taken from place p1 will be consumed as usual, and a new token placed in place p3, though the test arc also returns a token in place p1.

4.2 Formal Analysis Techniques

The use of formal specification techniques can be greatly enhanced if they are supported by formal analysis techniques. In the area of Petri nets, such analysis techniques can be used for the detection of deadlocks (when no actions are available/fireable in the Petri net model) ((Dix 1991), (Harrison and Thimbleby H.W 1990), (Paterno and Santoro 2002)), the presence or absence of a terminating state, the boundedness of the model, the liveness of the model, the reversibility and home state or to investigate temporal logic, i.e. to verify properties of the system model such that a certain state can never be reached for example. Furthermore, certain analysis techniques can be used to extract scenarios of events leading to a particular state, this is useful for tracing history.

Petri nets provide a number of benefits for designers who wish to reason about the effects of proposed changes to a system. Numerous formal analysis techniques exist for exploitation on Petri net models which can be used to analyse and verify the correctness of the specifications.

Methods of analysis, include checking for service availability, which can be used to identify deadlocks and analyse the model liveness. Also, limiting the number of occurrences, meaning a model is bounded by a set number of tokens, which is useful for representing physical limits of system. When modelling a system, it may

be important to ensure that the initial state is always reachable. This is particularly relevant for safety-critical interactive system that may be required to reset/restart/shutdown quickly in an emergency situation. Such design properties allow evaluation of the quality of the model before system implementation.

Since our interest is in human-computer interaction, tracing scenarios of activities is particular useful for us. For this reason we will describe one formal analysis technique called marking graphs (or reachability analysis) which can be used to identify the set of reachable states provided that some information is given about the initial status or 'marking' of the system. We have explored the use of this technique for this research.

The analysis starts from a known initial marking and identifies all the markings that a system can possibly reach as well as the necessary conditions and even sequences that lead to each marking. Though from a safety perspective, forward reachability is not as effective as backward reachability for two main reasons, it is often impractical to generate and analyse the entire reachable state spaces and also because it is a claim that unsafe states are not reachable is more effective than stating all reachable states are safe. This technique helps to produce what can be thought of as a form of state transition diagram. Several tools are currently available to support this analysis. However, this style of analysis can impose considerable overheads when considering complex systems such as the case studies used in this thesis. Figure 1-15 and the descriptions that follow provide an explanation of marking trees and marking graphs.

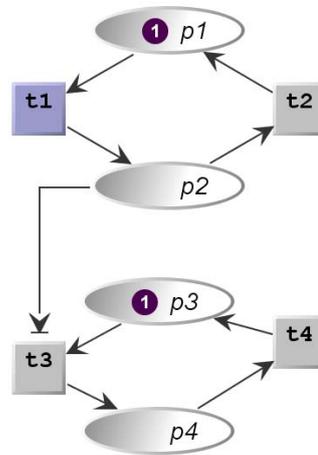


Figure 1-15. Simple Petri net to explain marking graph analysis technique

The Petri net presented in Figure 1-15 can be in any of 4 possible states. The initial state can be defined as $M_0: \{p_1, p_3\}$ signifying that places p_1 and p_3 contain a token. This is the current marking in Figure 1-15. The four states are therefore:

$M_0: \{p_1, p_3\}$; $M_1: \{p_2, p_3\}$; $M_2: \{p_2, p_4\}$; $M_3: \{p_1, p_4\}$; this is the marking tree of the Petri net in Figure 1-15.

The marking graph of the Petri net is somewhat similar to a state diagram in that it shows, depending on the current state, which transitions are fireable and therefore what the next possible states can be. From state M_0 , only one transition is fireable, that is t_1 , thus the transition from state M_0 to state M_1 can be defined as:

$M_0: (t_1: M_1)$; this means from state M_0 , transition t_1 must be fired in order to reach state M_1 . The complete marking graph, in textual format is thus:

$M_0: (t_1: M_1)$
 $M_1: (t_2: M_0) (t_3: M_2)$
 $M_2: (t_2: M_3) (t_4: M_1)$
 $M_3: (t_1: M_2) (t_4: M_0)$

The marking tree (represented in textual format above) for the Petri net in Figure 1-15 indicates that there are 4 reachable states (M_0 , M_1 , M_2 and M_3 respectively), or 4 nodes, and that the net can always return to the initial state, M_0 . The graphical marking tree is presented in Figure 1-16. Each state is represented in an eclipse while the transitions are the arrows between the states. The state M_0 is represented as $M_0: \{1,0,1,0\}$ which means that there is one token in place p_1 , no tokens in place p_2 , one token in place p_3 and no tokens in place p_4 .

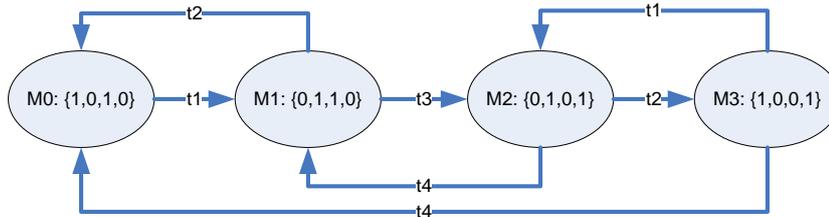


Figure 1-16. Graphical Marking Graph for Petri net in Figure 1-15

This approach can be seen as useful for example after an incident or accident has occurred and a model of the system is created in order to simulate events leading to the unsafe state. Such analysis techniques are less useful if the unsafe states are unknown, typically during early design phases.

4.2.1 Marking Graph Tool Support

Currently, Petshop is unable to support automatic generation of marking trees and marking graphs. For simple Petri nets such as the one shown in the previous section, it is possible to perform the analysis by hand, though when the net is more complex, the procedure becomes extremely time consuming and difficult to manage. Our research found two tools that are able to support the automatic generation of marking trees and marking graphs if used in combination, of considerably large models (too large to perform by hand), though they were still unable to support the complete models that we use in our case studies.

The first tool is called JARP Petri Net Analyzer version 1.1 (Figure 1-17 presents a screenshot) developed by Sangoi Padilha (<http://jarp.sourceforge.net/us/index.html>) which has been developed as an auxiliary tool for the Petri net analyzing tool ARP developed by Alberto Maziero (<http://www.ppgia.pucpr.br/~maziero/diversos/petri/arp.html>).

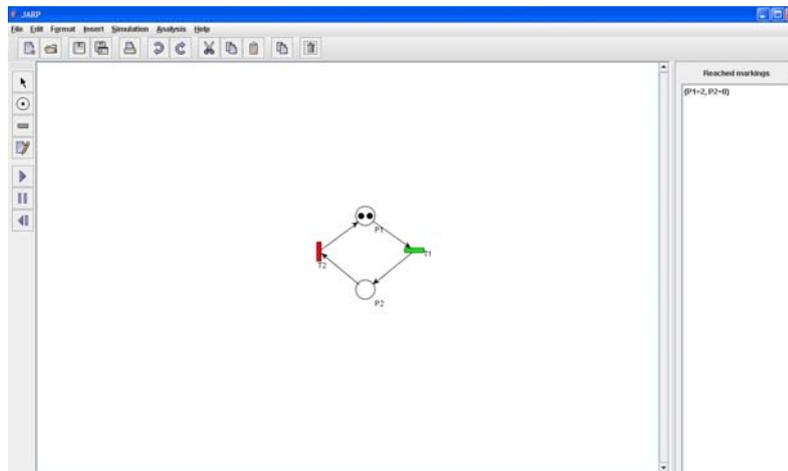


Figure 1-17. Screenshot of the JARP Petri net editor tool

Both of which have been developed by the Department of Automation and Systems Engineering of the Federal University of Santa Catarina in Brazil. The main features of JARP are graphical Petri net editing, interactive net simulation, possibility to export nets to ARP file format for analysis purposes, export nets as XML based PNML (Petri net mark-up language) file format as well as GIF, JPEG, PNG and PPM graphical file formats.

The ARP tool, which stands for "Analisador de Redes de Petri", the Portuguese translation for Petri Net Analyser is a tool (Figure 1-18 presents a screenshot) written in Turbo Pascal 6 for MS-DOS for Petri net analysis and simulation. It accepts the following Petri net models:

- Place/transition nets (numeric markings, no time information);
- Timed nets (firing intervals [tmin, tmax] associated to transitions);
- Extended timed nets (probability function associated to firing intervals).

It offers the following analysis modules:

- Reachability graph generation and analysis
- Place/transition invariant analysis

- Verification of the equivalence between the state graph and a user given automata
- Performance evaluation
- Step-by-step simulation

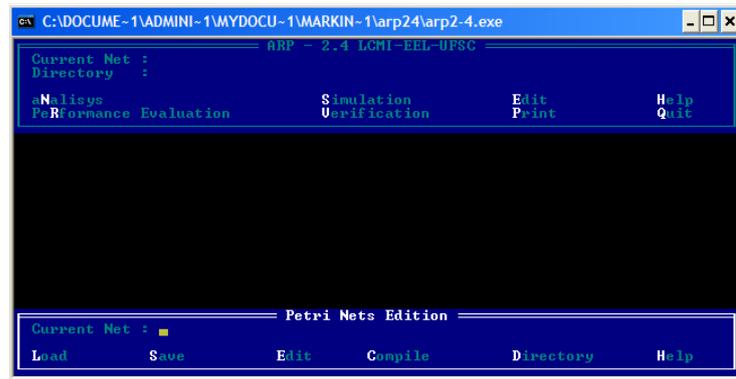


Figure 1-18. Screenshot of the ARP Petri net analysis tool

The procedure used in this thesis for producing marking trees and marking graphs can be summarised as follows:

- Create Petri net using Petshop
- Open PNML file of Petshop Petri net in JARP
- Save the PNML in JARP as an ARP 2.4 PN file
- Open the ARP 2.4 PN file in the ARP tool (see Figure 1-19)
- Use the ARP analysis tools to produce textual marking trees and marking graphs

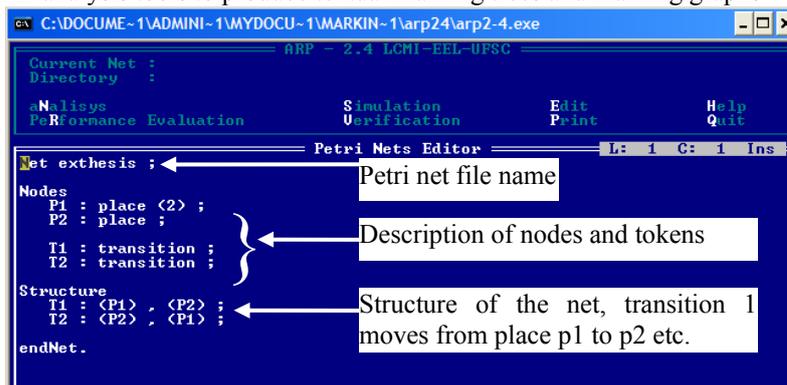


Figure 1-19. Petri net of Figure 1-17 opened in the ARP tool

Figure 1-19 illustrates the available information when the Petri net of Figure 1-17 is opened in the ARP tool. The information available includes the file name, a description of the nodes and token distribution as well as the structure describing which places the transitions connect.

The reachable states result using the ARP tool is presented in Figure 1-21. The results show there are three reachable states, M0: {2*p1}, M1: {p1, p2} and M2: {2*p2}.

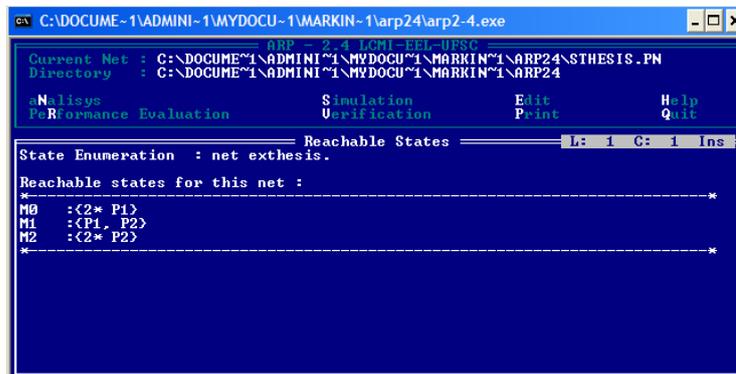


Figure 1-20. Results of reachable states for Petri net in Figure 1-17

In addition to reachable states, the ARP tool provides further verification properties. These are presented in Figure 1-21 and briefly explained below. Finally Figure 1-22 illustrates the results of the marking graph analysis provided by the tool.

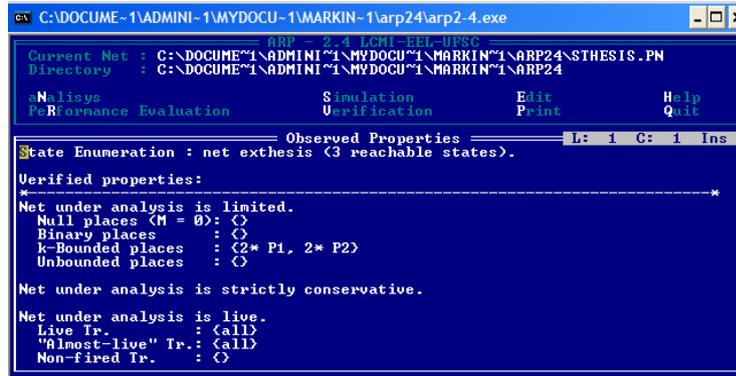


Figure 1-21. Marking Tree for Petri net in Figure 1-17

- Net under analysis is limited: this means there is not an infinite number of tokens in the network
- Null places $\langle \rangle$: There is never a state with no tokens in the network
- K-Bounded places: places p1 and p2 are bounded to 2 tokens each
- Unbounded places $\langle \rangle$: There are no places with an infinite number of tokens
- Net under analysis is strictly conservative: whatever transition is fired the Petri net model is not gaining or losing tokens
- Net under analysis is live: whatever state the network is in there will always be a transition fireable
- Live Tr $\langle all \rangle$: there is always at least one transition fireable
- “Almost-live” Tr: whatever state the model is in, there is always a sequence of transition that will make fireable any transition in the model
- Non-fired Tr $\langle \rangle$: There are no transitions that are non-fireable

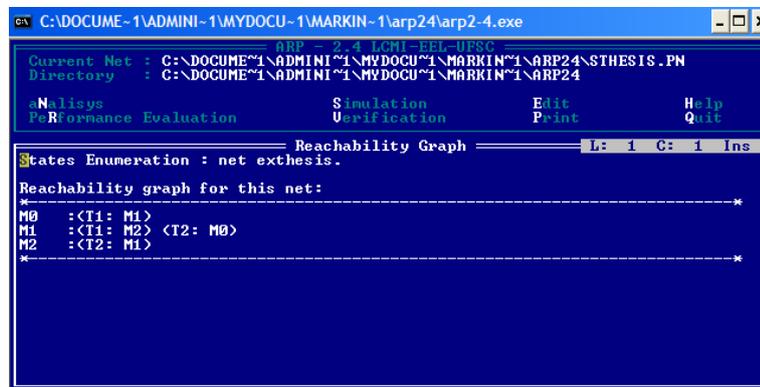


Figure 1-22. Marking graph

Though the tools have been very useful for us, they both have considerable limitations. Firstly, the procedure is lengthy, with the need to convert files into many formats. JARP still has many bugs which is why it was not used to create the Petri nets. It also does not high-level Petri net modelling as Petshop does. The ARP tool is DOS based making it hard to use for the inexperienced. Obtaining the marking graphs were the results of many attempts at trial and error.

5 Interactive Cooperative Objects (ICOs) & Petshop

We have seen in the previous section that Petri nets are highly capable of specifying event and state-based features of systems. Further more, they allow mathematical based verification and validation techniques to be applied for property reasoning. We have also argued that formal specification techniques are beneficial due to their powerful expressiveness, ability apply non-ambiguous, complete and concise notations.

The extensions and dialects of Petri nets described in the previous section relate to basic Petri nets. Nets with additional features such as tokens holding values, transitions containing preconditions, arcs having variables for which the main point is the flow of tokens rather than available actions are called high-level Petri nets.

A notation, which comprises the above extensions resulting in high-level Petri nets exists called Interactive Cooperative Objects (ICOs) (Navarre 2001). Since we are interested in specifying the behaviour of interactive systems, and more particularly safety-critical interactive systems which require complete, concise and non ambiguous descriptions, and thus we will focus our attention on ICOs which is the most appropriate formalism allowing us to specify the most relevant information. Free choice, coloured networks, stochastic networks and object networks do not allow the specification of interactive behaviours where both event-based and state-based descriptions are required.

This section presents the Interactive Cooperative Objects (ICO) formalism that we will use for safety-critical systems modelling in this thesis. ICO allows for describing, in a formal way, all the components of highly interactive applications, including post-WIMP (Windows Icons Menus and Pointers) applications. The ICOs formalism is a formal description technique dedicated to the specification of interactive systems (Navarre et al. 2003). It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets (Genrich 1991) to describe their dynamic or behavioural aspects.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user). An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented (Bastide et al. 2004). The specification can also be validated using analysis and proof tools developed within the Petri nets community.

ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behaviour of objects and communication protocol between objects are described by Petri nets. The formalism made up of both the description technique for the communicating objects and the communication protocol is called the Cooperative Objects formalism (CO).

In the ICO formalism, an object is an entity featuring four components:

1. **Cooperative Object (CO):** a cooperative object models the behaviour of an ICO. It states how the object reacts to external stimuli according to its inner state. This behaviour, called the Object Control Structure (ObCS) is described by means of high-level Petri net. A CO offers two kinds of services to its environment. The first one, described with Java software interfaces, concerns the services (in the programming language terminology) offered to other objects in the environment. The second one, called user services, provides a description of the elementary actions offered to a user, but for which availability depends on the internal state of the cooperative object (this state is represented by the distribution and the value of the tokens (called marking) in the places of the ObCS).
2. **Presentation part:** the Presentation of an object states its external appearance. This Presentation is a structured set of widgets organized in a set of windows. Each widget may be a way to interact with the interactive system (user \rightarrow system interaction) and/or a way to display information from this interactive system (system \rightarrow user interaction).
3. **Activation function:** the user \rightarrow system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the ICO's user services. The relation between user services and widgets is fully stated by the activation function that associates to each couple (widget, user action) the user service to be triggered.
4. **Rendering function:** the system \rightarrow user interaction (outputs) aims at presenting to the user the state changes that occurs in the system. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes.

ICO are used to provide a formal description of the dynamic behaviour of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e., how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e., when and how the application displays information relevant to the user). Time-out transitions are special transitions that do not belong to the categories above. They are associated with a timer that automatically triggers the transition when a

dedicated amount of time has elapsed. When included in a system model such transition is considered as a system transition. They can also be included in a user model representing spontaneous user's activity.

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented (Navarre et al. 2004). The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique.

ICOs are used to provide a formal description of the dynamic behaviour of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e., how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e., when and how the application displays information relevant to the user). Time-out transitions are special transitions that do not belong to the categories above. They are associated with a timer that automatically triggers the transition when a dedicated amount of time has elapsed. When included in a system model such transition is considered as a system transition. They can also be included in a user model representing spontaneous user's activity.

5.1 Petshop Environment

In this section we present the PetShop environment which fully supports the ICO formalism. The tool has been developed over the past 8 years by members of our research team LIIHS. Figure 1-23 presents the general architecture of PetShop. It is composed of a set of rectangles and documents-like shapes. The rectangles represent the functional modules of PetShop. The documents-like shapes represent the models produced and used by the modules.

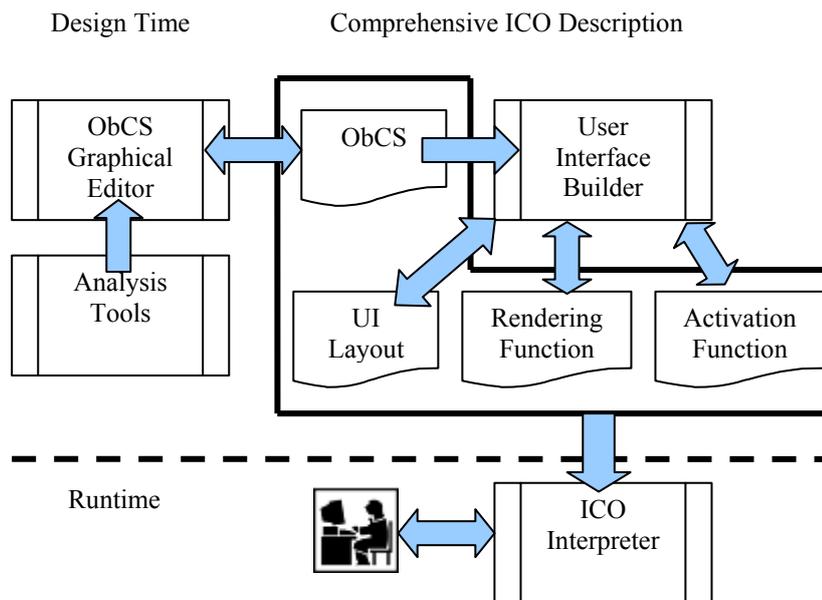


Figure 1-23. Architecture of the PetShop Environment

PetShop features an Object Petri net editor (Figure 1-24) that allows for editing and executing the ObCSs of the classes.

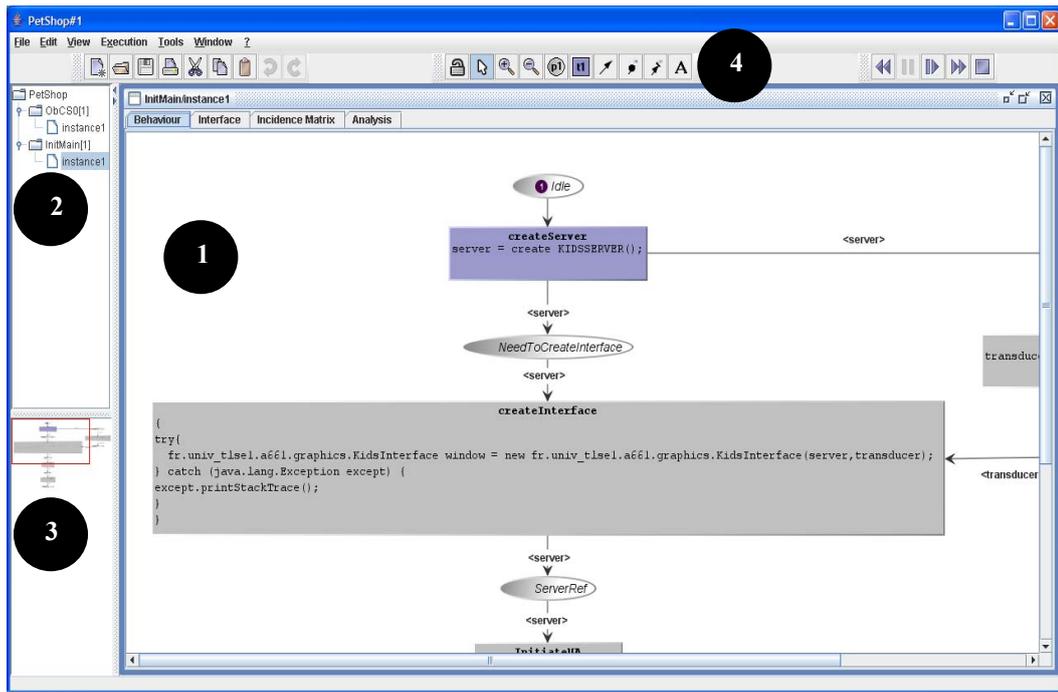


Figure 1-24. The ObCS editor in Petshop

Figure 1-24 has been annotated for explanatory purposes. 1) is the area in which networks are edited and modelled, 2) provides a list of network objects, 3) provides a miniview of the entire network and 4) is the toolbar.

All the components of the ObCS can be interactively built using PetShop. The editing of the Object Petri net is done graphically using the Palette in the centre of the toolbar (see number 4 of Figure 1-24). The left part of the toolbar is used for generic functions such as load, save, cut-copy and paste. The right hand side of the toolbar drives the execution of the specification. A well-known advantage of Petri nets is their executability. This is highly beneficial since as soon as a behavioural specification is provided in term of ObCS, this specification can be executed to provide additional insights on the possible evolutions of the system.

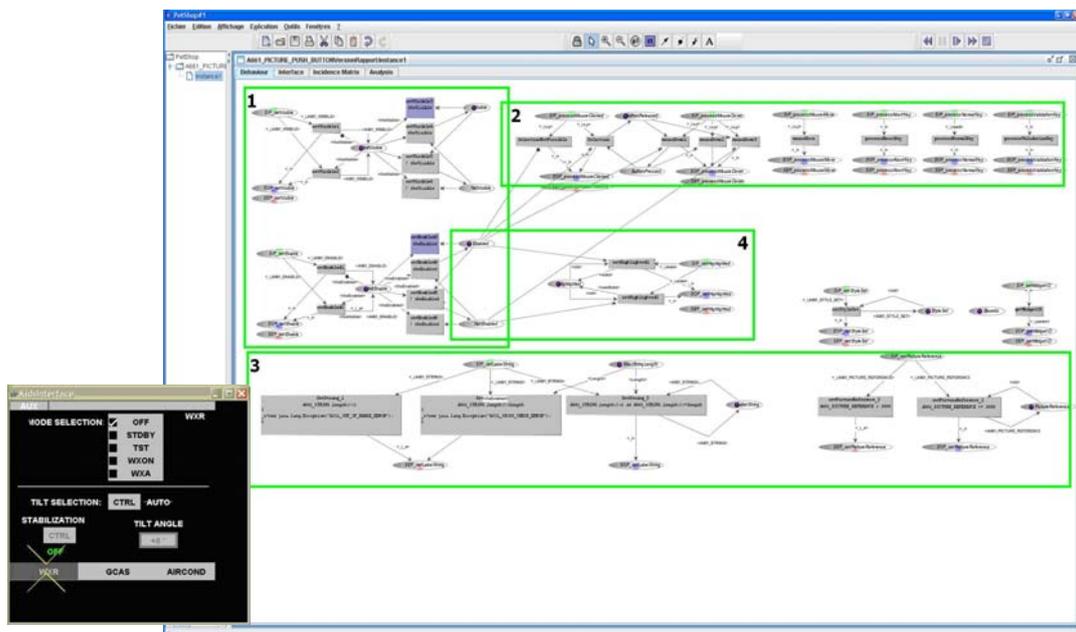


Figure 1-25. Interactive Prototyping in Petshop

Figure 1-25 shows the execution of the specification of a cockpit software application in Petshop. The ICO specification is embedded at run time according to the interpreted execution of the ICO. At run time, the user

(system developer/designer) can both look at the specification and the actual application. These are presented in two different windows overlapping in Figure 1-25. Each time the user acts on the cockpit application interface, the event is passed onto the interpreter. If the corresponding transition is enabled then the interpreter fires it, performs its action (if any), changes the marking of the input and output places and perform the rendering associated (if any).

5.2 Scalability of ICOs and Petshop

Usually, one of the points put forward while designing formal description techniques such as those presented in this chapter is to provide a notation powerful enough for describing/specifying the systems under consideration. We have shown in this chapter that the *expressive power* of the Interactive Cooperative Objects (ICO) formalism is sufficient for describing interactive applications such as (for instance) the ones of interactive cockpits or ground segments for satellite control rooms. In this section, we address issues relating to the *applicability power* of the formalism and describe extensions that have recently been defined and implemented to make it usable for real size interactive applications and even in an industrial context. The extensions comprise of three innovative techniques, to specifically deal with large and real-life safety-critical interactive systems. The solutions include model restructuring and visualization techniques (labels hiding, virtual places and mini-view) (Barboni et al. 2006b). We present these solutions in the following paragraphs.

Although we discuss modifications with respect to the ICO formalism any other graphical notation such as Statecharts (Harel 1987) would require similar extensions to deal with the kinds of applications we are interested in, real life safety-critical interactive systems.

The discussion in this section does not provide a formal description of the ICO extensions, rather it discusses the solutions to problems encountered when studying a real life military cockpit application and how these proposed solutions have been implemented allowing ICOs to scale up in order to handle large interactive software systems.

Restructuring models for graphical appearance

The goal of model restructuring is twofold; firstly it simplifies the management of graphical aspects, reduces the size of the models and reduces development time. Secondly, it provides an elegant solution to a recurring problem related to the selection of widgets, called “picking”. Picking is a task that has to be solved by the user interface server in order to determine which widget has been the target of a user action on the input device.

Label hiding

A simple solution to the problem of illegible arc labels in large scale models is to give the designer the option to mask arc labels. This modification is purely graphical and does not change any modelling aspect. Once the arc labels are masked, it is possible to edit the arcs in the usual way (double clicking on the desired arc).

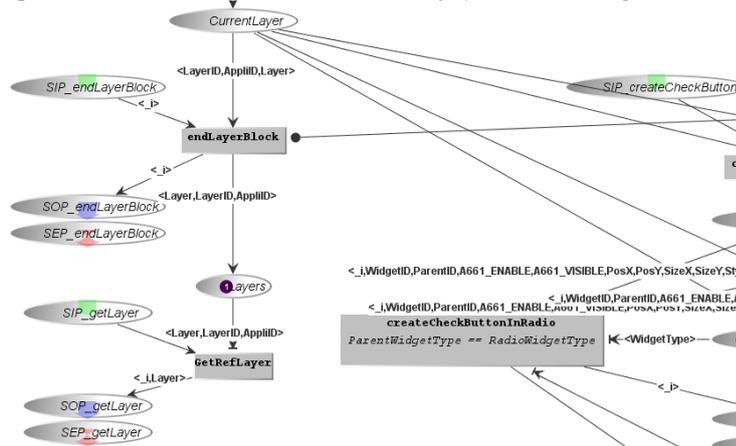


Figure 1-26 Partial view of the MPIA Server with displayed arc labels (taken from (Barboni et al. 2006b))

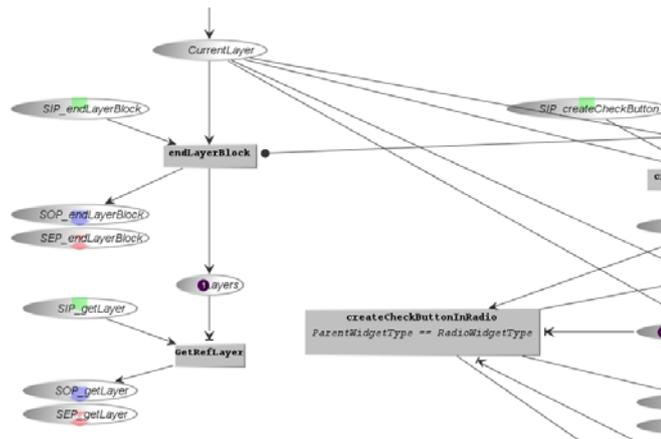


Figure 1-27 Partial view of the MPIA Server with arc labels masked (taken from (Barboni et al. 2006b))

Virtual places

This is a third solution implemented in order to improve legibility of the model by reducing the number of places in the net. A virtual place is a partial copy of a normal place. It adopts the display properties (such as markings) but not the arc connections. The arcs can be connected to normal places or to virtual places. The display is therefore modified allowing easier reorganisation of models. On a semantic level, the group of virtual places and the “source” place are considered as a normal place for which the connections ensemble of the virtual place arcs and of the “source” place. The left hand side of Figure 1-28. shows three places P1 (the “source” place with a darker border and the two virtual places) each with an arc. The right hand side of Figure 1-28. shows the result of the reconstitution of the place.



Figure 1-28. Real view (left) versus Semantic view (right)

This functional structuring of models and the support of virtual places reduces significantly the time and the effort for construction and modification of models. It is however important to note that this kind of construct has to be used carefully as, breaking down the graphical dynamics of the models can also result in making more difficult the understanding of their behaviour. To solve this problem the authors (Barboni et al. 2006b) have defined usage rules limiting their use to places storing parameters and not being part of an information flow.

Mini map

In order to simplify the navigation in large models, Barboni et al., (Barboni et al. 2006b) have added a “mini map” feature to Petshop (see Figure 1-24). The bird’s eye view is a representation of the complete model but on a much small scale. The bird’s eye view contains a small box which highlights what is currently visible in the main viewing area. The user can change which part of the model is visible in the main viewing area by moving a small box on the bird eyes view. On the contrary, if the user zooms or move within the main viewing area, the box on the bird’s eye view will move accordingly. This feature can be seen near label 3 of Figure 1-24.

These recent features will be used in thesis to deal with large models.

5.3 Integrating Human Factors with ICOs

Before the work presented in this thesis, there has been limited focus on using the ICO notation to make explicit human factors issues even though the notation is capable of representing such information. The developers of ICOs and Petshop have shown an interest in usability issues and have briefly explored the use of ICOs to represent both a task model and a system model. In addition to this, they have applied Fitts Law (Fitts 1954) to assist designers in making more suitable design decisions when designing UIs by making use of performance evaluations. These approaches are briefly described below.

Palanque and Bastide, (Palanque and Bastide 1997) demonstrate the possibility and the benefits of using a common formalism to describe both the system and the tasks of users interacting with it. The user behaviour is included in a CTT task model which is then made consistent with the system model and performance evaluation

data using ICOs. Using the same formal framework for both the task and system model provides an easy way to check compliancy between the two models, and to perform quantitative analysis to make sure the models comply with pre-planned objectives. ICOs can be used to describe interactive tasks using the “user services” class (described previously in Section 5 Interactive Cooperative Objects (ICOs) & Petshop), services that can be triggered interactively by the user through the use of an input device, or some conventional user interface element. Typically, performance evaluation for formal specification involves modelling temporal aspects.

The authors ground the performance evaluation on the model of the human processor (Card et al. 1983) in which the user is modelled as a set of three processors interacting together : the perceptual system, the motor system and the cognitive system. The Petri net models can include values of the human process model about human performance while interacting with a system. This is particularly important as it shows actual behaviour of a user.

In addition to this performance evaluation, the use of quantitative evaluation is also advocated in order to capture information of the sequences of actions users must perform to achieve their goals for example. This can be achieved using stochastic or timed Petri nets which typically provide data on the number of actions a user must perform in order to reach a goal, the number and length of cycles in a model and data relating to temporal constraints.

Similarly, based on the relation between the task model and system model, Navarre et al., (Navarre et al. 2001a) make use of CTT and ICOs to support the development of interactive systems. The work does not use performance evaluation to inform the design, it uses scenarios as a cornerstone for achieving consistency between models. The task modelling scenarios illustrate user activities required in order to achieve a goal, while on the system modelling side, the scenarios illustrate a sequence of user actions. Since the two notations have their own tool support, the contribution in this research is tool support, called the correspondence editor, via scenarios for integrating the two techniques.

More recently, (Lacaze et al. 2002) have also extensions to the ICO notation and Petshop to support temporal aspects of user interaction. This time, the goal was to use formal descriptions of temporal elements for evaluating in a quantitative way the relative efficiency of different designs. Representation of time can be constant, stochastic or related to the value of a token. The extensions to ICOs related to three ways in which timing issues can be modelled. These are: 1) time associated to transitions (timed and immediate), 2) time associated to places, and 3) “temporal” Petri nets where a timer is associated to each token. This work is extremely low level and is used to inform the designer of the average time it will take a user to respond to an interaction. The authors carry out a performance evaluation of two similar designs based on a scenario extracted from a task model. Again, using the three components of the human model processor, motor processor, perceptual processor and cognitive processor are evaluated. Regarding the motor processor component, Fitt’s law (Fitts 1954) is applied, for the perceptual processor, fixed temporal information is applied, for example, 33ms for comparing two numbers.

Table 1-6 has been taken from (Lacaze et al. 2002) and simplified as the precise details of the scenario are not necessary here, however it shows the classification of different tasks from the scenario and the effective mean time taken to perform each.

Table 1-6. Table describing effective time for each action or transition (adapted and taken from (Lacaze et al. 2002))

Scenario 1	Type of transition	Time
Verify	Cognitive	100ms
Press on	Physical	251ms
Read	Perceptual	100ms
...
Total		1599ms

In this thesis, we do not extend the ICO notation, instead we exploit the existing expressive power of ICOs and the Petshop tool to make explicit human factors issues such as human “errors” as well as software and hardware barriers which help prevent erroneous activities.

6 Model Based Safety Critical Interactive Systems Design

The previous sections have presented established model based design techniques and dedicated notations for the specification of safety-critical interactive systems. Here we introduce research in the field of model-based design for safety-critical interactive systems.

This section also presents a discussion on formal approaches for describing interactive systems and human computer interaction followed by a discussion on existing model-based design approaches for safety critical interactive systems.

With respect to task modelling, Paternò and Santoro (Paterno and Santoro 2002) have worked extensively on using semi-formal task modelling techniques and support tools for modelling human deviations from intended plans. The goal is to identify the possible deviations based on a set of predefined classes of deviations identified by guidewords, which are words or phrases referring to a specific type of abnormal system behaviour. The approach is described in more detail in section 3.1.

The method consists of three steps:

- Development of the task model of the application considered; in order to identify how the system design requires that tasks are performed.
- Analysis of deviations related to the basic tasks; the basic tasks are the leaves in the hierarchical task model, tasks that the designer deems should be considered as units.
- Analysis of deviations in high-level tasks; these tasks allow the designer to identify groups of tasks and consequently to analyse deviations that involve more than one basic task.

Hsiung et al., (Hsiung and Lin 2005) propose an approach to deal with the lack of support of software code testing in model-based approaches. The authors propose a model-based formal verification technique for safety-critical systems by applying a model checking paradigm to a safecharts model. Safecharts (Dammag and Nissanke 1999a) is a variant of Statecharts (see section 0) intended exclusively for safety-critical systems design. With two separate representations for functional and safety requirements, Safecharts brings the distinctions and dependencies between them into sharper focus, helping both designers and auditors alike in modeling and reviewing safety features. The approach takes a hierarchical safetchart model of the system and transforms it into an Extended Timed Automaton (ETA) model which is used as the input to a State Graph Manipulator (SGM) model checker to verify the safety-critical system satisfies functional and safety properties.

Heiner and Heisel (Heiner and Heisel 1999), combine the Z specification notation with Petri nets for modeling safety-critical systems. They claim that this combination preserves the strengths of the two formalisms while ameliorating their drawbacks. The drawbacks mentioned are the fact that Petri nets become large when specifying realistic systems and with respect to Z, the fact that a set of operations cannot be further defined to express the desired order of operations. The authors argue that this approach is advantageous because of available animation tools can provide an executable model of the system that allows stakeholders to obtain an impression of how the system will behave. Furthermore, a variety of analysis tools (which are available free of charge) provide richer validation facilities than they are available for other formalisms. Checking consistency of the two parts of the specification further enhances reliability in the model.

Recognising the need for model-based reasoning about sociotechnical safety-critical system design, Cebulla (Cebulla 2004) focuses on modelling concepts of such systems. The author provides extensions to formal methods, requirements engineering and software architecture based on visual notations to make them more suitable to the challenges of sociotechnical systems such as structural dynamism, uncertainty and relevance of cognitive features. Interestingly, the author also provides provide concepts to deal with adaptive system behaviour and human error. Cognitive mental models are used to takes into account the agents subjective motivations and their influence on the global system's behaviour. Fault trees are used to represent human errors.

Cebulla (Cebulla 2004) claims that a visual modelling notation significantly increases system transparency, support interdisciplinary system analysis and is well-suited to support measurements of further education. In our experience however, when demonstrating ICOs and CTT to THALES engineering teams, we found that a significant amount of training is required before other members of the design team can understand and value what the models tell us. It is unclear from the paper whether it would be possible to simulate the operation of the

modelled sociotechnical system as we are able to with Interactive Cooperative Object (ICO) models in the Petshop tool (see section 5)

Ortmeier et al. (Ortmeier et al. 2003) have developed an integrated approach called ForMoSa (FORmal MOdels and Safety Analysis) for safety analysis of critical, embedded systems by covering three different aspects of safety: fault tolerance, functional correctness and quantitative analysis. ForMoSa is split into three different sections. The first deals with building a formal specification, (usually with statecharts) which also models possible failures, this is called the functional error model. The second part deals with qualitative analysis. The goal in this part is to find the causal dependencies between component failures and system failure. The last step deals with quantitative approaches, namely the approximation of hazard probabilities and the optimization of the system. Following the research, the authors conclude that it is important to combine different techniques for safety analysis. This coincides with our ideas as will be shown when we present the core of this thesis, the Integrated Modelling Framework. This is because different methods not only examine different aspects of the system, but also give contrary views (Rouvroye and Bliet 2002).

The complex and currently mostly informal process of gathering, refining and representing multiple source data for the design of an interactive safety-critical system can be supported using models that allow handling data at a high level of abstraction. However, not all relevant information can be embedded in a single model. Thus, the various models need to be consistent and coherent with one another. These issues are addressed in our proposed Integrated Modelling Framework which describes various models, approaches, tools and techniques for the data to be embedded in each model and for accurate interconnections of models.

6.1 Formal approaches for describing interactive systems and human-computer interaction

More generally than the specific use of ICOs for describing interactive systems and human-computer interaction, others have applied formal approaches for achieving similar goals.

Specifying non-interactive systems using formal methods has been highly publicised. Models describing the behaviour of simplistic systems such as light switches to more complex models of car transmission systems have been provided using methods such as state transition diagrams (STDs) and Petri nets for example (see (Degani 2004) p.14 and p.241). Though these examples do involve a user, Degani's models typically focus on system behaviour and system modes in order to address usability design issues and human-computer interaction.

This section will provide an overview of formal specification techniques (FSTs) for describing system behaviour though we will focus on the use of FSTs for interactive systems. The formal specification of an interactive system differs greatly from a non-interactive system due to the incredibly high number of available states available in a given system.

The system model of an interactive system represents the system data and its available actions. The model describes the behaviour of the system including what actions are offered by the system, when an action is available according to the current state of the system and what the effect of an action will be on the state of the system. The system model of an interactive system represents both how the system is presented to the user and how the user interacts with it.

A considerable amount of research has been carried out in the field of formal approaches for interactive systems ((Harrison and Thimbleby H.W 1990), (Johnson C.W 1992)) but little has been targeted at the use of formal methods for the design of an interactive system. When dealing with interactive systems it is now widely agreed upon that user information has to be taken into account and that this must be done through task analysis and task modelling.

Rushby (Rushby 1999),(Rushby 2001) uses model checking techniques to detect inconsistencies between machine and user models. The goal of this work is to discover error-prone designs. Rushby checks to see whether the actual system behaviour violates plausible models and natural expectations in accordance with training materials for example. The checking is achieved by simultaneously tracking the operation of both models and then using an iterative search. This enables modification to the machine and user model in order to achieve consistency.

6.1.1 Examples of FSTs for Interactive Applications Design

In their Cognitive Complexity Theory (CCT), Kieras & Polson (Kieras and Polson 1985) used formal methods to quantify the complexity of human-computer interaction. This was achieved by modelling the device and the user tasks as FSMs. CCT assumes that speed of learning and performance for a computer based device is a function of the content, structure, and amount of the knowledge required to carry out routine tasks (Lee et al. 1989). Since both the device and the task were represented in the same formalism, they were able to identify cases in which the user's task structure did not correspond with the device's structure (Heymann and Degani 2007).

6.1.1.1 OFAN

In his PhD thesis, Degani (Degani 1996) developed a framework that describes the environment, user tasks, device functionality and interface information as four concurrent processes using Statecharts (Harel 1987). In his OFAN modelling technique, (named using the Hebrew word for a set of perpetuating wheels, OFAN) the system representation as analogous to a set of cog wheels where an event in one wheel affects the adjacent wheel etc. Each module contains a process and broadcasts its active states and events to other modules in the network, often triggering other modules to change states. Through these broadcast communications, the five modules in the framework are all interconnected to form a whole—the human-environment-machine system. An OFAN model describes a system from the perspective of a single device within the system by means of pre-defined categories which have been annotated in Figure 1-29: 1) control elements (description of input controls), 2) control mechanism (model of the device functionality), 3) displays (description of the output elements), 4) environment (model of relevant environmental properties) 5) user tasks (sequence of user actions required to complete a task).

This work addresses mode errors, in particular automation-induced mode errors and to identify a variety of general interface ambiguity problems. The focus is on the system. Our approach differs in that we propose to inform a system model by taking into account human factors issues.

More recently, Heymann and Degani (Heymann and Degani 2007) investigate the correspondence between the machine's behaviour and the (abstracted) information that is provided to the user. These two aspects are formally described and analysed by considering the following four elements: (1) the machine, (2) the user's tasks, (3) the user interface, and (4) the user's model of the machine. Furthermore, the authors present a systematic procedure based on a reduction algorithm for reducing the machine model according to the user's task. The goal of this work is to assist in the generation and analysis of correct human machine interfaces. An application has been designed for the automatic generation of the UI. According to Heymann and Degani (Heymann and Degani 2007), an interface is correct if there are no error states, no restricting states, and no augmenting states.

- An error state occurs when the user interface indicates that the machine is in one mode when, in fact, the machine is in another. Interfaces with error states lead to faulty interaction. Frequently (but not always), error states are caused by the presence of non-deterministic responses to user interaction.
- A restricting state occurs when the user can trigger certain mode changes in the machine that are not present in the user model and interface. Interfaces with restricting states tend to surprise and confuse users.
- An augmenting state occurs when the user is told that certain transitions are available when, in fact, they cannot be executed by the machine (or are disabled). Interfaces with augmenting states puzzle users and have contributed to operational errors.

In contrast to Heymann and Degani ((Heymann and Degani 2007) (described above), (Shiffman et al. 2005)) and also Paterno's ((Pangoli and Paterno 1995) ,(Paterno et al. 1999) and (Mori et al. 2004), approaches (the most recent based on a transformation-based environment called TERESA <http://giove.cnuce.cnr.it/teresa.html>), we do not believe it is a good idea to automatically generate user interfaces from models. Clearly, a UI relies on a combination of information from a task model and a system model. Even though a task model specifies the activities a user must perform in order to reach a goal, it does not provide useful information on how or what to present on an interface. Similarly, a system model describes what actions and events will take place on the system side, but it does not help in understanding what to represent or how to represent this information in order to make a useful and usable interface. Automatic generation of UIs disregards the artistic side of design. Designing is also a complex process, many factors must be taken into account. These complex factors influence design decisions which must be justified. This is discussed in Lacaze's PhD thesis (Lacaze 2005) in which he extends the QOC (MacLean et al. 1996) notation, called TEAM (Traceability, Exploration and Analysis Model) and provides tool support called DREAM (Design Rationale Environment for Argumentation and Modelling) for the proposed notation.

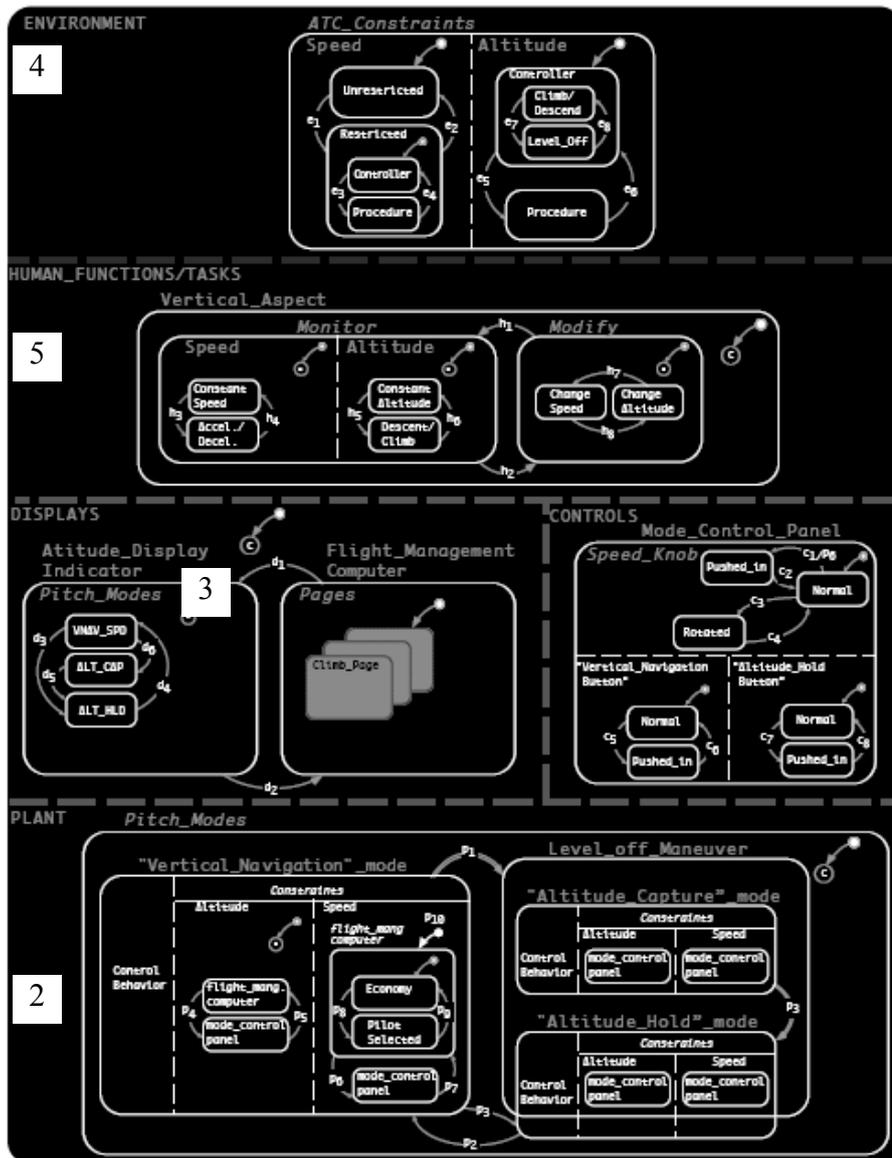


Figure 1-29. An example of an OFAN model (taken and adapted from (Degani and Kirlik 1995))

6.1.2 MUSE*/JSD

JSD (Jackson System Design) is a design method for software applications which includes software engineering processes, while MUSE (Method for Usability Engineering) is a method developed specifically to extend the scope of human factors contributions beyond late evaluation and thus increase their effectiveness and uptake. This is accomplished by making inputs more explicit, early and continuous throughout the system development process. Since MUSE's scope spans user requirements to user interface design, it supports active human factors involvement in both design specification and evaluation. MUSE defines how, what and when particular human factor concerns should be addressed. It also specifies the procedures, notations, and documentation involved (Lim K.Y and Long J 1995).

The contribution of such methods is large. Indeed, they provide a dedicated place for human factors issues, which enriches the methods on which they are grafted. Indeed, the MERISE and JSD methods have already proved their value in the field of the traditional software engineering.

This way of proceeding remains incomplete, because even if a role is attributed human factors actors, it remains marginal. Indeed, for these two methods, the introduction of these actors is done only by influencing the phases of the method used, without modifying them. In other words, far from being modelled for the support of the design of interactive systems, these methods are extensions to existing methods.

In the following section, we present the MEFISO method, a model-based design method for safety-critical interactive systems.

6.1.3 MEFISTO

The process of experts gathering data from various domains for input into the system design has been studied as part of the 'Mefisto' method "Modeling, Evaluating and Formalizing Interactive Systems using Tasks and interaction Objects" (Palanque et al. 2000) which is a development process for interactive systems design. The approach integrates model-based design techniques developed by members of the Mefisto project. That is to say, all information gathered via analyses or produced by design, is stored in relevant models. The method also highlights the kinds of information that should and should not be stored in any particular model. Furthermore, the development process provides tool support.

Mefisto proposes three main cycles:

- Process cycle: The first cycle of the method, describing how the various phases fit together
- Design cycle: shows how the process cycle can be conducted in an interactive way and how the process integrated various notations and techniques
- Abstraction cycle: focuses on notations and models that have to be built throughout the design and development of interactive safety critical systems

The process cycle (Palanque et al. 2000) describes a path that has to be followed to build both usable and reliable interactive systems (see Figure 1-30). In the first phase of the process cycle, the observation phase, information such as work practice, existing artefacts, business and organisational constraints are gathered. However, the process cycle does not detail how the information is gathered, who will gather it, or how the information will be recorded and reused.

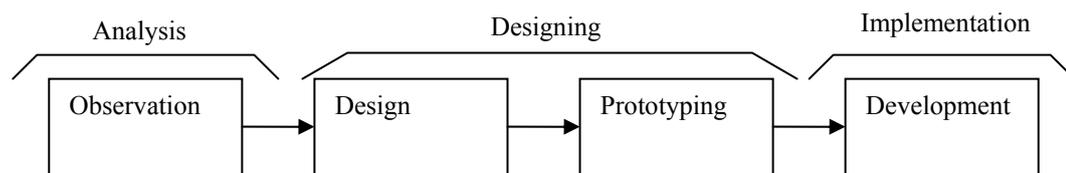


Figure 1-30. Schematic view of the Process cycle

Other approaches such as MUSE (Polet et al. 2003), a task-based approach, argue in the same way although the proposed process is different. Lim and Long's MUSE method provides a way of integrating human factors into information systems development approaches based on a structure chart notation. The authors propose to construct task models from which a composite task model is developed for the new system under design. The composite model describes both user and system activities. From this, separate system and user task models are developed.

Like most other design methods MUSE consists of an analysis phase, a synthesis phase and specification phase, as follows (de Haan 1998):

1. Information Elicitation and Analysis Phase
 - a. Extant Systems Analysis
 - b. Generalized Task Model
2. Design Synthesis Phase
 - a. Statement of User Needs
 - b. Composite Task Model--> JSD functions list
 - c. System and User Task Model<-- JSD information flow
3. Design Specification Phase--> JSD implementation model
 - a. Interaction Task Model
 - b. Interface Model and - Display Design

The second phase of the cycle of the Mefisto method is the design cycle. The aim of this cycle is to show how the various notations and activity performed and defined in the project Mefisto fit together. Figure 1-31 illustrates the design process which is naturally iterative due to the nature of interactive systems design. The diagram contains four quadrants representing the four main proposed activities, task modelling, system modelling, prototyping and user testing. It is clear that the design process of the Mefisto method is very much based on a User Centred Design (UCD) approach which is necessary for interactive systems as discussed early in this chapter. The central spirals represent the beginning of the process which can be either the task analysis phase or the human error analysis phase (within the prototyping phase). The authors argue that this important choice is to deal with the current state of the initial system which may influence the starting point of the design cycle. The

outer circle labelled ‘scenarios’ is to highlight that scenarios are used in all phases of the cycle. For example, to drive user testing, to start task modelling activities, to check a system model against a task model and also for driving a human error analysis.

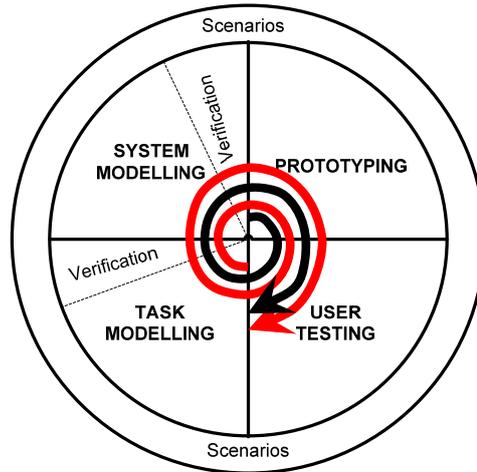


Figure 1-31. Detailed iterative design process (adaptation from (Palanque et al. 2000))

The third cycle of the Mefisto method is the abstraction cycle which describes the various models that are built following the Mefisto method. Figure 1-32 illustrates the abstraction process. The top level “concepts” is abstract and represents independent information and artefacts in its models. The second level, “information” contains models representing independent information from the implementation phase however more concrete than that of the conceptual level. The lower level “details”, represents concrete and precise information that is likely to change frequently. In addition to these three levels, Figure 1-32 contains three columns providing different viewpoints on the above described levels of abstraction. Column one is concerned with the system/software viewpoint, column two, the contextual information and column three, the users’ activities.

	Objects	Context	Activity	
CONCEPTS Conceptual level	1 Domain level	2 High Level Scenarios Stages	3 Goals	Functional goal What the organisation does. Business and Safety constraints and regulations
INFORMATION: Organisational Level	4 Mefisto domain model	5 Roles and Agents Environments	6 Task Models	Architecture What does what, when, and Where
DETAILS: Physical Level	7 Oriented Objects Programs	8 Stages Scenarios Instances	9 Practice	Physical How to perform actions

Figure 1-32. Content and structure of the Abstraction cycle (from (Palanque et al. 2000))

The Mefisto design cycle is the closest work to that of this thesis. Though the Mefisto method describes what should be done in order to develop an interactive system, it does not describe in detail how it can be done. This is particularly important for a safety-critical interactive systems design. It is far easier to state what needs to be done than describing how it can be done. This thesis will attempt to provide a framework providing a solution to the design of safer safety-critical interactive systems.

7 Specificities of safety-critical systems: The certification phase

This section briefly describes the certification phase of the development process of a safety-critical system. Though we do not directly contribute to this area, we make use of standards which have a strong implication on such certification phases. The case study section of this thesis makes use of the ARINC 661 standard.

Certification is a phase of the development process specific to safety critical systems. This activity involves independent organisations responsible for providing clearances prior to the actual deployment of the systems. This activity has a significant impact over the development process as its successful accounting is perceived by designers and developers as one of the main targets to achieve. Indeed, in case of certification failure, the whole development can be stopped and most of the time restarted with many negative economical and technological consequences. For this reason, certification authorities have developed specific development processes that 'guarantee' the quality of the product by means of structured and reliable processes. For instance DO 178 B (RTCA 1992) is a document describing such a design process widely used in the aeronautical domain.

The certification phase of the development of a safety-critical system is what makes the development process unique from that of a 'normal' (non safety-critical) system. In many highly critical projects the final state is to convince some external regulating body that the system is safe, and thereby to achieve certification (Storey 1996). Certification is the process of issuing a certificate to indicate conformance with a standard, a set of guidelines or some similar document (Storey 1996). The aims of certification are to improve safety of critical systems, to increase the awareness of the implications of system performance on safety, to enforce minimum standards of design and manufacture within the relevant industry and to encourage a structure of professional responsibility (Storey 1996).

In part 2 of the ARINC 661, a standard which aims to normalize the definition of a Cockpit Display System (CDS), and the communication between the CDS and User Applications (UA), it is stated that it is widely recognised that software qualification and system certification costs dwarf all other aspects of developing, installing and updating a Cockpit Display System (CDS) (ARINC 661-2 2005).

For a safety-critical system, safety aspects must be considered and the development processes must be able to address them in a balanced and consistent way. While these concerns are typical of the development process of any type of software systems, safety critical ones put a special emphasis on the certification phase. Indeed, developers of such systems are responsible for demonstrating that the system is 'acceptably safe' before certification authorities grant regulatory approval.

Storey (Storey 1996) indicates that there are three typical aspects to the certification of safety-critical systems:

- A demonstration that all important hazards have been identified and dealt with, and that the integrity of the system is appropriate for the application.
- Evidence of compliance with some particular standard.
- A rigorous argument to support the claim that the system is sufficiently safe and will remain so throughout its life

8 Conclusion

In this chapter we have provided an overview of the current state of the art in model based design, user centred design, human factors issues for user centred design, and formal specification techniques, models and methods dedicated to safety-critical interactive systems.

It is clear, that techniques such as ICOs with its recent extensions to support scalability are sufficient for describing the behaviour of large-scale systems. This can be justified following the work carried out for THALES in the field of interactive cockpit application both for military and civil aircrafts and for CNES (National Centre of Spatial Studies) on ground segment for satellite control systems etc.

However, apart from the brief use of Fitts law with a formal specification technique, none of the model-based design methods described in this chapter explicitly take into account human factors.

The highlighted limitations in current model based design techniques are a justified need for improved methods for which many other considerations must be taken into account to ensure efficient and successful human machine interaction particularly for safety-critical interactive systems. The following chapter will further exemplify this issue, by presenting issues relating more closely to safety and reliability.

Chapter 2

Designing for Resilience

“If we knew what it was we were doing, it would not be called research, would it?” (Albert Einstein, 1879 - 1955)

Chapter Summary

In continuation of the previous literature review chapter focussing on model-based design, this chapter highlights another aspect of the thesis which is safety, reliability, error tolerance and human factors issues. This chapter presents an overview of safety analysis methods for the design of safety-critical interactive systems based on the concept of resilience engineering. We introduce incident and accident investigation techniques as well as established theories on human error that will be applied within in our contribution to the thesis.

In this chapter, when we refer to resilience, we make explicit reference to Hollnagel's theory (Hollnagel 2006b)p16. published in his recent book “Resilience Engineering Concepts and Precepts” (Hollnagel et al. 2006). We are not trying to define resilience in this chapter or thesis. We simply use it as a framework for describing where the work of this PhD fits into current develop ideas in the field of safety-critical systems engineering and design.

Even though this concept and term is under definition, we peruse the same goals as the experts of this resilience field.

1 Introduction

While the previous chapter introduced the state of the art in formal specification techniques and model based design approaches, which are necessary for achieving reliability, this chapter introduces another perspective of the thesis, which is safety. This is of course necessary when considering the design of safety-critical interactive systems.

This chapter presents a relatively new area of research for the design of safety-critical interactive systems known as “resilience”. Though the term, resilience and its implementation have been applied to organisations and enterprises (which is discussed in this chapter), resilience engineering is now having an impact during the design phases of safety-critical interactive systems.

In this chapter, we identify known techniques for safety analysis and human error analysis for use in the development process. We also identify relatively newer techniques for considering such safety aspects in the model-based design process, which is our particular interest. Where possible, we have focused on the use of formal methods for safety analysis and error mitigation. We also tie together methods presented in the previous chapter, on task modelling with aspects of human error presented in this chapter.

The chapter is structured with a literature review of the term “resilience” and its origins, followed by a presentation of the concept of system safety. Approaches for the design of resilient socio-technical systems are then given which includes methods for learning from experience, techniques for anticipating failures and vulnerabilities, ways of achieving hardware and software safety as well as incorporating human factors in the design process.

These concepts and theories are interesting to present as they relate closely to the work in the PhD. In this chapter we move from hardware failures, to software failures and then to human ‘failures’ and errors. We provide examples, such as modes and mode confusion as a means of linking humans to software and software drivers as a means of linking hardware and software failure issues.

2 Background of Resilience

The concept of “resilience” is borrowed from the field of ecology. Holling (Holling 1973) characterised natural systems that are resilient, i.e., that tend to maintain their integrity when subject to disturbance. This is related to

the idea of stability. The informal concept of stability refers to the tendency of a system to return to a position of equilibrium when disturbed {Holling 1997}.

Within the domain of organisation and safety management, resilience is defined as the intrinsic ability of an organisation to keep or recover a stable state, thereby allowing it to continue operations after a major mishap or in presence of continuous stress (Hollnagel 2006a). Further definitions of resilience include, the capacity of a system to survive, adapt, and grow in the face of unforeseen changes, even catastrophic incidents (Center for Resilience), or the ability to resist disorder (Gunderson and Protchard 2002).

Traditional systems engineering practices try to anticipate and resist disruptions but may be vulnerable to unforeseen factors. An alternative is to design systems with inherent “resilience” by taking advantage of fundamental properties such as diversity, efficiency, adaptability, and cohesion (Fiksel 2003).

As highlighted above, resilience is a common feature of complex systems, such as companies, cities, or ecosystems (Center for Resilience). However, since our focus is on the design of interactive safety-critical systems, we will focus our attention on the human and computer-based systems aspects.

In the most recent book (at the time of writing this thesis) on resilience engineering, Hollnagel states that resilience is the ability of a system or an organisation to react to and recover from disturbances at an early stage, with minimal effect on the dynamic stability (Hollnagel 2006b)p16.

Hollnagel & Woods argue, (Hollnagel and Woods 2006a) that it is not sufficient for a system to be reliable and that the probability of failure is below a certain value, it must also be resilient and have the ability to recover from irregular variations, disruptions and degradation of expected working conditions. The authors also express two key points in resilience engineering, firstly that resilience itself cannot be measured, only the potential for resilience can be. One such measure of resilience is the ability to create foresight – to anticipate the changing shape of risk, before failure and harm occurs (Woods 2005).

Secondly, that resilience cannot be engineered simply by introducing more procedures, safeguards, and barriers. Resilience engineering instead requires a continuous monitoring of system performance, of how things are done. The second point is further exemplified in Section 2.2 describing resilient organisations.

Figure 2-1 illustrates the components of resilience. There are three qualities a system must have in order to remain in control and therefore be resilient. These are knowledge, competence and resources. Here, Hollnagel and Woods are referring to the “system” as a whole, including humans.

Knowledge is necessary for recognising what to expect (anticipation) and knowing what to look for and where to focus next (attention, perception). Knowledge requires constant updating in order to develop knowing what to look for. Competence and resources are important for the system’s ability to respond rationally. In this case, competence refers to knowing what to do how to do it, whereas resources refer to the ability to do it (Hollnagel and Woods 2006a). The dashed line across the centre of Figure 2-1 represents the boundary between the system and the environment.

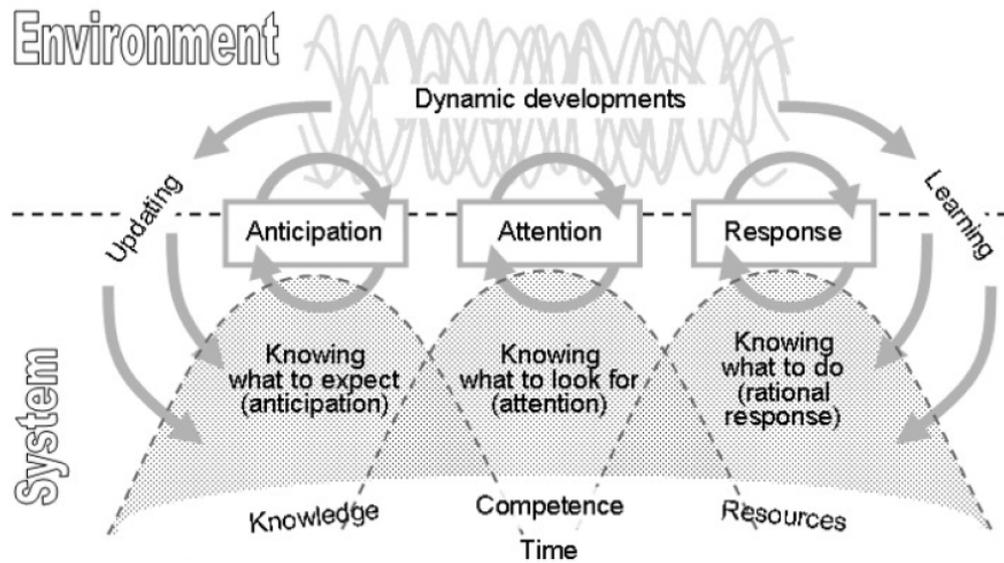


Figure 2-1. Required Qualities of a Resilient System(Hollnagel and Woods 2006a)

2.1 System Safety

System safety includes both hardware and software (Storey 1996), though if we widen our perspective of what a system is, system safety must also include humans. To clarify this concept, a definition of a system is provided.

2.1.1 System Definition

A system is a set of components that act together as a whole to achieve some common goal, objective or end. The components are all interrelated and either directly or indirectly connected to each other (Leveson 1995)p.136. A similar definition is provided by (Avizienis et al. 2004a), a system is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena. These other systems are the environment of the given system. The system boundary is the common frontier between the system and its environment. It is important to note, that these boundaries and interdependencies are often blurred and thus difficult to define.

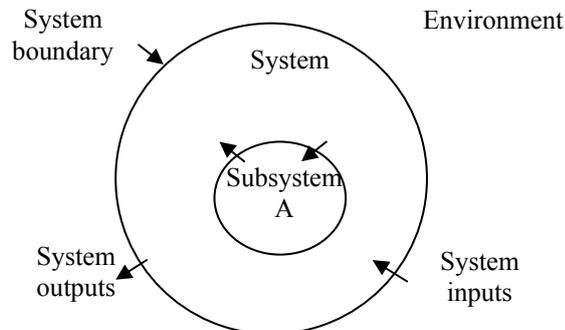


Figure 2-2. Definition of a system (taken from (Leveson 1995))

A system can at any time, be in one of many states. Taking a light switch as an example, the switch could be either on, off. The system state at any point in time is the set of relevant properties describing the system at that time (system specification is defined further in Section 4.1). The system environment is a set of components (and their properties) that are not part of the system but whose behaviour can affect the system state. The existence of a boundary between the system and its environment implicitly defines as inputs or outputs anything that crosses that boundary. Any set of components that is considered to be a system is, in general, at least potentially part of a hierarchy of systems: A system may contain subsystems and may also be part of a larger system (Leveson 1995).

2.1.2 Safety Definitions

According to (Storey 1996),(Hollnagel et al. 2004) (Weill-Fassina et al. 2006), safety is a property of a system; encompassing components, subsystems, software, organizations, human behaviour, and their interactions(Weill-Fassina et al. 2006), that it will not endanger human life or the environment (Storey 1996). Terms such as

“endanger”, as Johnson remarks, (Johnson 1999a) can clearly be interpreted subjectively. Furthermore, Johnson and Holloway exemplify the fact that safety is a dynamic property of the system, because the constraints that are applied and the degree to which a system satisfies those constraints will continually evolve over time (Johnson and Holloway 2003). Leveson (Leveson 2004) affirms the issue of emergence stating, safety is an emergent property of systems that arises from the interaction of system components (Johnson 2006).

However, in more recent literature, Hollnagel & Woods’ views on safety as a property appear to have changed according to (Hollnagel and Woods 2006a) which states system safety is not a property, safety is something a system *does*, rather than something a system *has*.

Leveson defines safety as freedom from accidents or losses (Leveson 1995)p.181 while the Mine Safety and Health Administration (MSHA) defines it as freedom from risks (Sammarco et al. 2001). The problem with such definitions stems from the subjectivity of the terms loss and risks. What is considered a loss or a risk? There may be various levels of acceptability of loss and types of loss will vary between domains and applications. Additionally, in the industrial context, a safety manager who must assess system safety could simply assume that since there has not been any loss or risk, the system is currently safe.

Further views of safety extracted from International Standards Organizations (ISOs) include freedom from unacceptable risk of physical injury or of damage to the health of people, either directly or indirectly as a result of damage to property or to the environment (International Electrotechnical Commission IEC 61508 2003) section 3.1. Furthermore, Laprie (Laprie 1998), whose research area is dependable computing defines safety as the non-occurrence of catastrophic consequences on the environment (Laprie 1998).

It is clear that there are many definitions and views on safety. In the real world, it is acceptable to assume that complete freedom from adverse events is not possible, at the same time, proving a system is absolutely safe is also not possible. Although a system could be operating as expected for a considerable amount of time without problems, variables such as the hardware failures, environmental changes, modifications to system operators etc., could at a certain point in time, combine to change the level of safety in a system and consequently result in the system moving into an unsafe state. Safety managers must not become complacent, in that continued safe operation does not necessarily guarantee a safe system. Therefore, the goal in system safety design and system safety management is to attain an acceptable level of safety, or “relative safety” as Johnson (Johnson 1999a) prefers to refer to safety as. The term relative safety relies on an individual’s perception of risk. Though risk has been widely cited in the literature as “risk = frequency x cost”, frequency and cost are relative and subjective to a project or system. It should be noted however, that cost is not just financial. Cost can be loss of trust (in a system by people if a machine crashes), as well as loss of life, environmental damage etc. Both parameters should be taken into account however, because failures can be of the type that occur daily, costing little to resolve, or they can occur very rarely incurring high costs, most likely resulting in a disaster.

Even if risk could be measured, there is still the problem of selecting the level of risk to use in the decision making process. Thus the most common criterion is that of “acceptable risk”. A threshold level is selected below which risk will be tolerated (Leveson 1995).

2.1.3 Safe Systems

Now that the terms system and safety have been defined independently, the notion of a ‘safe system’ can be analysed. Hollnagel (Hollnagel 2006a) describes a safe system as one in which nothing unwanted happens. This concept is illustrated in the following diagram adapted from Hollnagel’s keynote presentation (Hollnagel 2006a).

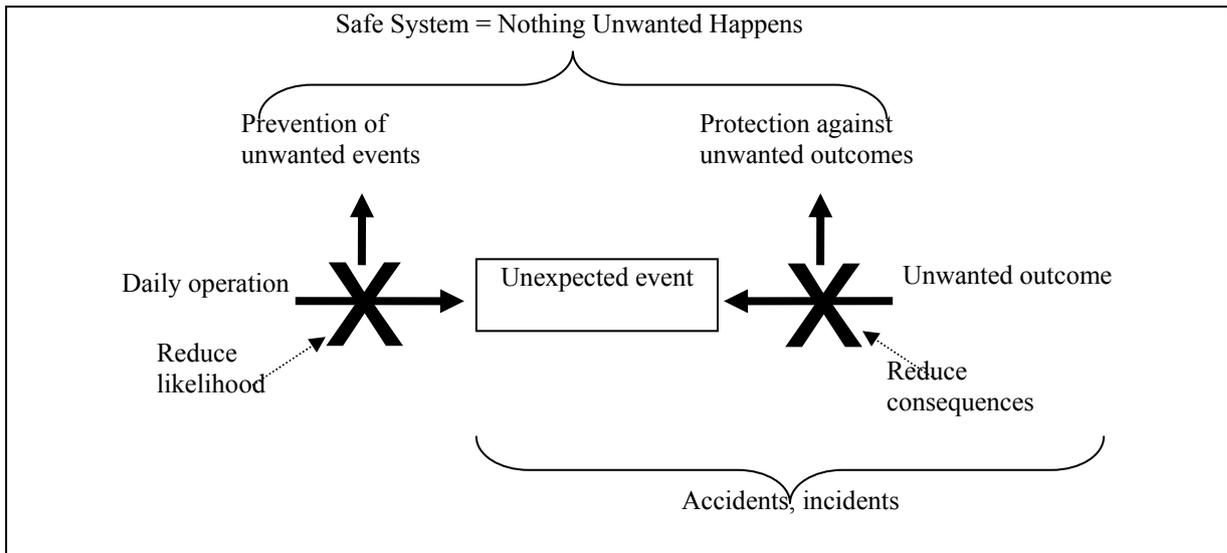


Figure 2-3. Safety as a non-event (Hollnagel 2006a)

System safety is a planned, disciplined, and systematic approach to identifying, analyzing and controlling hazards throughout the life cycle (of a project, programme or activity (Adduci et al. 2000)) of a system in order to prevent or reduce accidents. (Leveson 1995). Leveson (Leveson 1995) distinguishes several principles of system safety. These are as follows, system safety;

- emphasizes building safety, not adding it on to a completed design.
- deals with systems as a whole rather than with subsystems or components
- takes a larger view of hazards than just failures
- emphasizes analysis rather than past experience and standards
- emphasises qualitative rather than quantitative approaches
- recognises the importance of tradeoffs and conflicts in system design
- is more than just system engineering.

2.2 Resilience in organisations

Recent major mishaps and case studies have identified the critical need for organizations to re-tool their engineering processes and capabilities to address human and organizational risk factors (Weill-Fassina et al. 2006). Resilience engineering is a paradigm for safety management that focuses on how to help people cope with complexity under pressure to achieve success (Hollnagel and Woods 2006b). Resilience engineering stands for the view that failure is the flip side of the adaptations necessary to cope with the complexity of the real world, rather than a breakdown or malfunctioning as such (Hollnagel et al. 2004).

According to Hollnagel (Hollnagel 2006a), there are three types of organisation. Reactive, interactive (attentive) and proactive (resilient). A reactive organisation (Figure 2-4) will perform safety planning including safety analysis, identification of potential threats etc., and then react to an incident realising that the operation of the system was not as planned. An example of reactive organisation taken from Hollnagel is Mont Black Tunnel Fire (March 26 1999).

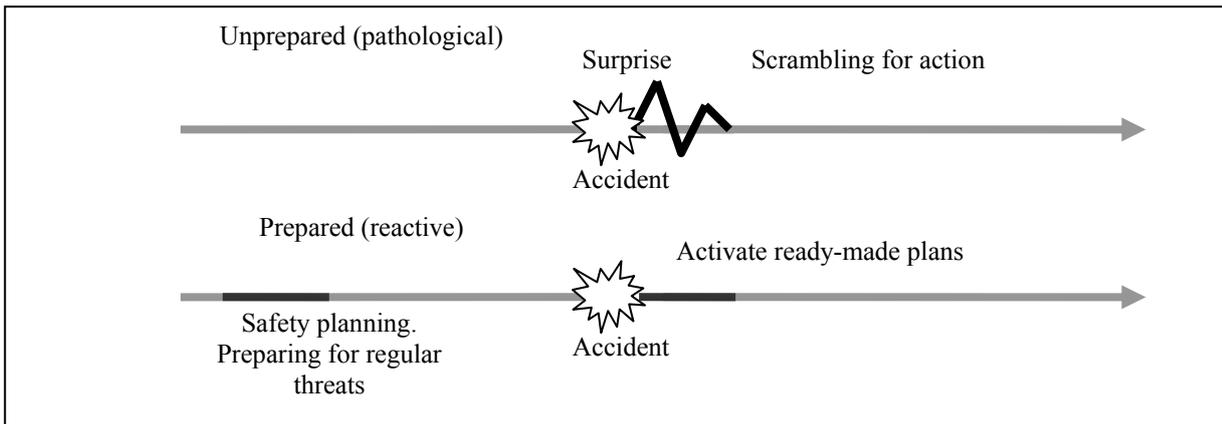


Figure 2-4. Reactive Organisation (Hollnagel 2006a)

An interactive (attentive) organisation (see Figure 2-5) is one that will try to prepare for irregular threats and attempt to learn from them. It looks for expected situations and maintains regular or irregular health checks to determine if the organisation is running how it should be. An example of an interactive organisation taken from Hollnagel is the aviation industry, nuclear power plants.

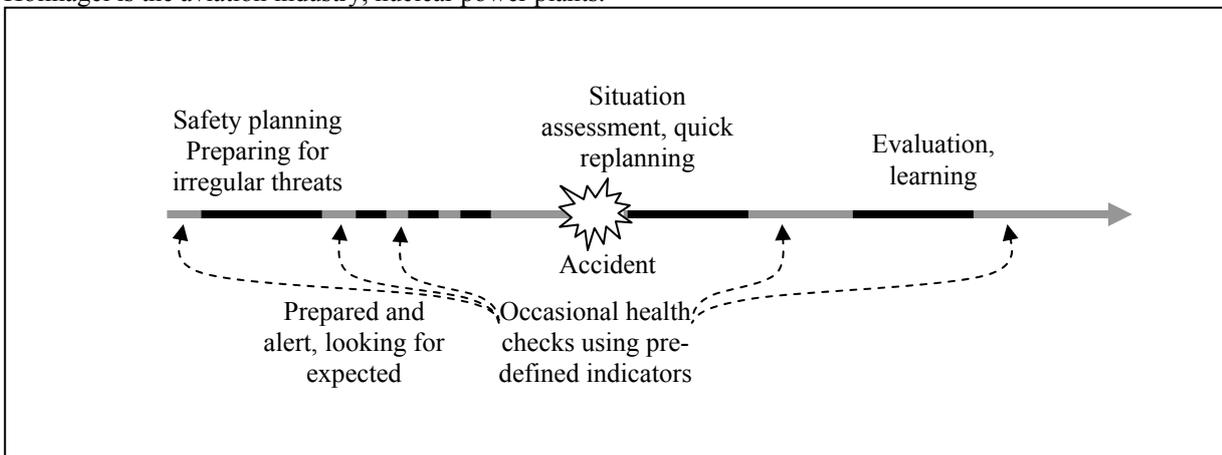


Figure 2-5. Interactive (attentive) Organisation (Hollnagel 2006a)

A proactive (resilient) organisation (Figure 2-6) performs the same activities as reactive and interactive organisations but also has a sense of constant alertness about it. The organisation will constantly be asking itself “are we ready”. An example of proactive organisation taken from Hollnagel is Israeli hospitals, who are adapted and able to take care of victims of bus bombings for example as well as continue to perform normal activity.

It is important to note however, that designing for resilience may result in more complex and hierarchical, procedural and organisational issues that might in turn result in less resilient systems. For instance, to prevent people with contagious diseases entering a hospital, a procedure may be implemented requiring patients to fill in complex forms for a decision to be made on their current health situation. One could argue, that imposing such procedures could consume resources (time, money, staff) resulting in suffering to other patients.

On a more abstract level of detail, the Ohio Center for Resilience (<http://www.resilience.osu.edu/>) state that the most powerful lever for enhancing enterprise resilience is product and process design. Design protocols have evolved from an emphasis on cost and customer satisfaction to a broader consideration of other life cycle success factors such as manufacturability and maintainability. However, the external boundary conditions are generally assumed to be constant. Consideration of how external conditions might change will enhance system resilience.

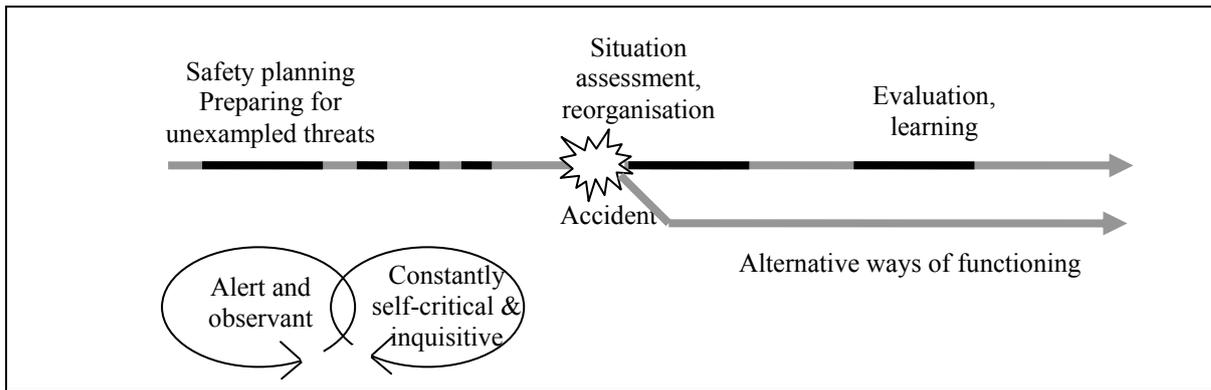


Figure 2-6. Proactive (resilient) Organisation (Hollnagel 2006a)

3 Achieving Interactive Systems Safety by Resilience Engineering

Resilience engineering can be incorporated into the many phases of the development process of a safety-critical interactive system. Since the focus of this PhD is on human-computer interaction issues, this section centres on techniques relating to resilience design for humans and technical systems. Hollnagel (Hollnagel 2006a) provides three critical components that the practice of resilience engineering must comprise of:

- Ways to analyse, measure and monitor the resilience of organisations in their operating environment
- Tools and methods to improve an organisation's resilience vis-à-vis the environment
- Techniques to model and predict the short and long-term effects of change and decisions on risk

This section will present techniques from the literature including learning from experience, anticipating failures and vulnerabilities, hardware and software safety including barrier analysis and implementation as well as human error management.

3.1 Learning from Experience

A common approach to enhance a system's safety is to learn from experience by looking and taking into account information from previous real life cases. Typically, negative cases or failures are investigated. A failure is normally explained as a breakdown or malfunctioning of a system and/or its components. (Hollnagel 2006a). One such usually available and particularly pertinent source is the outcome of an incident or accident investigation. However, accidents can be considered rare with respect to the number of system implementations and Johnston et al, (Johnston et al. 2006), state that although there are well publicised incidents involving mass transport systems, such as railways or commercial aircraft, records show that these remain the safest way of travelling. Johnston et al. (Johnston et al. 2006), argue that designers could also learn from successful cases. Success is due to the ability of organisations, groups and individuals correctly to make these adjustments (to information, resources and time) in particular to anticipate failures before they occur. Safety must encompass strengthening this ability, rather than just avoiding or eliminating it (Hollnagel 2006a). In Hollnagel's most recent book, *Resilience Engineering, Concepts and Precepts*, he appears to agree with Johnston et al, stating "Safety is the sum of the accidents that do not occur. While accident research has focused on accidents that occurred and tried to understand why, safety research should focus on the accidents that did not occur and try to understand why" (Hollnagel 2006b).

Thus though there are few accidents, investigators and designers are able to extract a significant amount of information from incident (such as near misses) and accident investigations, which infers there must also be considerable useful information that we are missing from the successful implementations that could influence future design. Kontogiannis et al., (Kontogiannis et al. 2000) argue that the analysis of near misses, for instance, can reveal useful information about recovery actions that stopped the sequence of events from giving rise to an accident.

An example of a successful system can be inferred from Cook & Nemeth's (Cook and Nemeth 2006) explanation of the Haddassah Ein Karem Israeli hospital and emergency services following a suicide bomb. The chapter describes the procedures in response to a bus bombing. Cook relates the definition of success with the definition of a resilient performance. "To perform resiliently, practitioners necessarily defer or abandon some

goals in order to achieve others". Furthermore, the authors defines resilience in resilient performance as "the degree to which the operators are able to handle the disruption to the usual goal" (Cook and Nemeth 2006).

Alternatively, Holloway & Johnson (Holloway 2006) highlight that system safety professionals and researchers could reap important benefits, including a recognition of the limits of mathematical models and increased understanding of the consequences of unlikely events, from reading accident reports. There is currently no proposed method for extracting, collating, interpreting and making use of this positive information. It would be difficult to identify, within a system, which components made the system a success.

Based on the lack of literature available with respect to the use of successful information, the remainder of this thesis will focus on the use of incident or accident investigation data to feed into the development process.

3.1.1 Incident and Accident Analysis

Incident and accident analysis can extend the scope of our design and development "imagination" by providing important insights into previous failures. Before discussing incident and accident investigation techniques useful for the design of a safety-critical interactive system, definitions of an incident and accident are provided. An incident is an action that involves no loss (or only minor loss) but with potential for loss under different circumstances' (Leveson 1995). An accident is an undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss' (Leveson 1995).

An accident report is the published results of an investigation into an accident. The investigation is performed either internally, by the organisation involved, or externally by a governing body (Burns 2000). Accident reports are intended to ensure that the faults of previous systems are not propagated into future applications. They contain the analysis of many different experts: human factors specialists; control engineers; meteorologists etc. (Johnson 1999b). Few accidents are now caused by a single error or failure (Burns 2000). Rather they generally stem from complex sets of technical, organisational, sociological, and environmental factors, some of which may have existed in the system for many years, which combine in an unforeseen way to penetrate the defences of the system. The term 'organisational accident' is used to describe such accidents (1997).

The results of an accident investigation provide suggestions for improving the system or future designs to avoid a similar accident. Interestingly, Holloway & Johnson, highlight that there is a misconception that accident investigators terminate their investigation once an operator has been identified to blame. Many researchers believe that accident investigators are inclined to blame human operators at the expense of conducting thorough examinations into organizational and other systemic failures. Such a notion is perhaps not surprising...when statements such as '75% [or some other high percentage] of accidents are blamed on human error' (Holloway 2006).

Johnson and Holloway argue that it is inaccurate to assert that: (1) the operator is always blamed, (2) most investigations stop as soon as they find someone to blame, or (3) organizational causes are usually ignored (Johnson and Holloway 2004).

In a recent study (Johnson 2006 (in review)) (Johnson and Holloway 2004), Johnson and Holloway performed an independent analysis of the probable and contributory causes of aviation accidents in both the United States and Canada between 1996 and 2003. 27 accident reports were taken from the Canadian Transportation Safety Board (TSB) and 26 from the US National Transportation Safety Board (NTSB). The authors identified that apart from human error, wider systemic issues, including the managerial and regulatory context of aviation operations contributed to a high percentage of the accidents under analysis.

Based on the Canadian reports, the results suggest that the majority of these high consequence accidents were attributed to human error (Johnson and Holloway 2004). However, the results for the US indicate that organizational issues appeared as a more prominent cause of aviation accidents than individual human error (Johnson 2006 (in review)). This is further investigated by the authors who indicate that the TSB focus on the role of the human error as a potential cause in most of the adverse events that they investigate (Johnson and Holloway 2004). To summarise, it was found that managerial issues account for around 1% of all causes and 27% of contributory factors across both analysts. Regulatory issues account for 3% of all causes and 10% of all contributory factors.

Quite rightly, human error is not and should not be the terminating point of an accident investigation. Identifying humans as casual factors of an accident can provide ways of unveiling much more detailed information about larger more complex causal factors. These may include management and training failure, environmental effects,

operator state-of-mind etc. This indicates that management, organisation and regulation issues must also be explicitly addressed by resilience engineering.

Section 3.1.2 presents established safety analysis methods from the literature including the method used in the proposed integrated framework.

3.1.2 Overview of Safety Analysis Methods

Continuing this section on resilience engineering, we now present a number of established methods of analysing safety.

One of earliest methods described in this section is STEP developed by Hendrick and Benner (Hendrick and Benner 1987). The STEP technique provides a systematic method for the reconstruction of the accident based on multi-linear events sequences. The main concepts in STEP are the initiation of the accident through an event or change that disrupted the technical system, the agents which intervene to control the system (e.g. human beings, automatic controllers, equipment, and monitoring systems), and the elementary event building blocks (Sklet 2002). The process is plotted on a worksheet which is a simple matrix of rows and columns. Each row represents an actor and the columns represent a time frame. The method is considered systematic because it also includes testing procedures. The row test determines if more information is required for any individual actor listed and the column test determines whether the sequence of events is correct by checking new events with actions of actors. The 'necessary-and-sufficient' test is used when suspicion of actions by one actor could have triggered actions by another actor.

Preliminary Hazard Analysis (PHA) ((Leveson 1995) Appendix), is an early form of hazard analysis. As the name implies it is applied during initial phases, in the concept planning and preliminary design. PHA accomplishes the following tasks; Identify system hazards, Translate system hazards into high-level system safety design constraints, Assess hazards if required to do so, Establish the hazard log (Leveson 2003a).

Failure Modes and Effects Analysis (FMEA) can be applied to preliminary design, final design and implementation phases. In FMEA, each mode of the components of the system is considered in order to assess the effect on the system. Hereby it is possible to identify the failure modes that are critical with regard to some property, e.g. safety (Isaksen et al. 1996). Failures Modes, Effects and Criticality Analysis (FMECA) is an extended version of FMEA based on a block diagram where each failure mode is attributed some occurrence probability and the seriousness of its consequences are rated. Several tools exist to support FMECA, these include Relex ® FMEA/FMECA (<http://www.relex.com/products/fmeafmecca.asp>) and ITEM software (<http://www.itemuk.com/>).

Fault Tree Analysis (FTA) can be applied to preliminary design and final design stages. The analysis starts with a single undesirable event from which a process of identification is carried out to identify combinations of events that could lead to the single undesirable event. The method is represented graphically with the undesired event at the root. Events are connected using logic gates i.e AND, OR, NOT.

Fault trees are, are typically used pre hoc to analyse potential errors in a design. They have not been widely used to support post hoc accident analysis. They do, however, offer considerable benefits for this purpose (Johnson 1999c).

A Hazard and Operability Studies (HAZOP) (MOD 1996) analysis is a systematic technique that attempts to consider events or process in a system exhaustively. Within a particular system domain (or scenario), items (or events) are identified and a list of guidewords is applied to the items. The approach is applied to a graphical representation of the system. In the chemical industry, from which the approach originated, Piping and Instrumentation Diagrams are typically used. The guidewords prompt qualitative consideration of deviations to item behaviour (guideword examples include less, more, none, more than) to elicit the potential for arriving at possible hazardous states (Smith and Harrison 2002). These guidewords provide the structure of the analysis and can help to ensure complete coverage of the possible failure modes (Pumfrey 1999). HAZOP is similar to FMECA in that it often neglects the human and contextual factors (Jeffcott and Johnson 2002b) even though, there are a number of human orientated versions of HAZOP that apply the analysis to task representations (Leathley 1997) and have been applied to a variety of safety critical human operator interfaces.

Event Tree Analysis (ETA) is also known as incident sequence analysis (Leveson 1995) p.158. The purpose of the method is to identify event sequences and their potential consequences. An event tree analysis starts with one so-called 'initiating event' and builds a tree of potential consequences depending on the following events. Each

path from the root of the tree (the initiating event) to a leaf is an event sequence. Typically a safety-critical system will have several layers of defence in order to control or limit any damage due to faults within the system. Hence a sequence (or combination) of safety-related failures will typically constitute a critical event sequence in an event tree analysis (Isaksen et al. 1996).

Probabilistic Risk Assessment (PRA) is a quantitative methodology used in many fields such as nuclear, transportation, or rail for assessing the probability of failure or success of a system. (Kumamoto and Henley 1996) define risk as "a combination of five primitives -- outcome, likelihood, significance, causal scenario, and population affected". The authors define the purpose of risk assessment as to propose alternatives, evaluate risk profiles, make safety decisions, choose satisfactory alternatives to control the risk and exercise corrective actions. The United States Nuclear Regulatory Commission's (NRC) Regulatory Guide 1.177 - An Approach for Plant-Specific, Risk-Informed Decision making: Technical Specifications, (NRC 1998) recommends that the use of PRA technology should be increased in all regulatory matters to the extent supported by the state of the art in PRA methods and data and in a manner that complements the NRC's deterministic approach and supports the NRC's traditional defence-in-depth philosophy. Though PRA is known to ignore software contributions to risk, Bin Li's PhD thesis (Li 2004) introduces a methodology to account for the impact of software on system failure that can be used in the classical PRA analysis process.

A recently developed method for analyzing system safety is STAMP, Systems-Theoretic Accident Model and Processes (Leveson 2002), which is specifically targeted for use after a mishap has occurred. STAMP is motivated by the observation that elements of systems theory might be applied to support the analysis of incidents and accidents (Johnson and Holloway 2003). Leveson argues that the chain of events can easily miss subtle and complex couplings and interactions among failure events and omit entirely accidents involving no component failure. The idea behind STAMP is to focus on a control based view of an accident and the basic concepts are constraints, control loops and process models, and levels of control (Leveson 2004).

A control-model is constructed once investigators have identified the sequence of events leading to an incident (Johnson and Holloway 2003). This control model can be analysed by checking where constraints were inadequate or missing which consequently led to the factors causing the failure. In their study, Johnson and Holloway determined that the approach is relatively easy to use and is arguably cost effective in terms of the insights that are obtained for the effort involved in performing the analysis.

3.1.2.1 Barrier Analysis

Within the section of safety analysis methods, this section is dedicated to barrier analysis, a technique we focus on and contribute to in this thesis.

Barrier analysis is typically applied after an incident or problem has been identified. The analysis addresses which barriers were in place and how they performed, which barriers were in place but not used, barriers that were in place but not required, and further barriers (if present or existing barriers that could be strengthened) could have prevented the same or a similar accident from occurring.

There are many definitions of a barrier. For example, Hollnagel states that a barrier is an obstacle, an obstruction, or a hindrance that may either (i) prevent an action from being carried out or an event from taking place, or (ii) prevent or lessen the impact of the consequences, limiting the reach of the consequences or weakening them in some way (Hollnagel 1999b). Johnson states that barriers represent the diverse physical and organisational measures that are taken to prevent a target from being affected by a potential hazard (Johnson 2003) while Schupp et al., describe barriers as the combination of technical, human and organizational measures that prevent or protect against an adverse effect (Schupp et al. 2004). Though with these many definitions, what is clear is that a barrier is either a protective or preventative measure involving technical and human systems.

Risk reduction is a key factor in the design of safety critical systems and in assessment of their operational safety. It is achieved either by preventing hazards or by protecting against hazards. Prevention typically involves design modifications of the total system, including for example operating procedures. Protection involves the design of additional systems, which embody barriers that fend against adverse events, damage or harm (Schupp et al. 2004).

3.1.2.2 Barrier Classifications

In addition to the definitions of barriers, Leveson and Hollnagel have provided classifications of barriers. According to Leveson (Leveson 1991), a distinction can be made between three types of barriers called lockout, lockin, and interlock, respectively. A lockout device "prevents a hazardous, event from occurring, while a lockin

device maintains safe conditions. An interlock is used to ensure that a sequence of operations occurs in the correct order.”

Hollnagel (Hollnagel 2004)p.87 proposes four categories of barrier systems which are described below. A summary of these classifications and examples of each are provided in Table 2-1.

1. Physical or material barriers which physically prevent an action from being carried out or the consequences of a hazard from spreading. For example a fence or wall.
2. Functional (active or dynamic) barriers which impede an action from being carried out, for instance the use of an interlock
3. Symbolic barriers which require an act of interpretation in order to achieve their purpose. For example a give way sign indicates a driver should give way but does not actively enforce/stop non-compliance
4. Incorporeal barriers which lack material form or substance in the situation but depend on the knowledge of the user to achieve their purpose. For example the use of standards and guidelines.

Table 2-1. Barrier systems and barrier functions (taken from (Hollnagel 1999a))

Barrier system	Barrier function	Example
Material, physical	Containing or protecting. Physical obstacle, either to prevent transporting something from the present location (e.g., release) or into present location (penetration).	Walls, doors, buildings, restricted physical access, railings, fences, filters, containers, tanks, valves, rectifiers, etc.
	Restraining or preventing movement or transportation.	Safety belts, harnesses, fences, cages, restricted physical movements, spatial distance (gulfs, gaps), etc.
	Keeping together. Cohesion, resilience, indestructibility	Components that do not break or fracture easily, e.g. safety glass.
Functional	Dissipating energy, protecting, quenching, extinguishing	Air bags, crumble zones, sprinklers, scrubbers, filters, etc.
	Preventing movement or action (mechanical, hard)	Locks, equipment alignment, physical interlocking, equipment match, brakes, etc.
	Preventing movement or action (logical, soft)	Passwords, entry codes, action sequences, pre-conditions, physiological matching (iris, fingerprint, alcohol level), etc.
Symbolic	Hindering or impeding actions (spatio-temporal)	Distance (too far for a single person to reach), persistence (dead-man button), delays, synchronisation, etc.
	Countering, preventing or thwarting actions (visual, tactile interface design)	Coding of functions (colour, shape, spatial layout), demarcations, labels & warnings (static), etc. Facilitating correct actions may be as effective as countering incorrect actions.
	Regulating actions	Instructions, procedures, precautions / conditions, dialogues, etc.
	Indicating system status or condition (signs, signals and symbols)	Signs (e.g. traffic signs), signals (visual, auditory), warnings, alarms, etc.
	Permission or authorisation (or the lack thereof)	Work permit, work order.
Immaterial	Communication, interpersonal dependency	Clearance, approval, (on-line or off-line), in the sense that the lack of clearance etc., is a barrier.
	Monitoring, supervision	Check (by oneself or another a.k.a. visual inspection), checklists, alarms (dynamic), etc.
	Prescribing: rules, laws, guidelines, prohibitions	Rules, restrictions, laws (all either conditional or unconditional), ethics, etc.

Kjellén, s proposes to separate barriers into two classes, active or passive (Kjellén 2000). For physical barriers it is convenient to talk about passive and active barriers (Kjellén 2000). A passive barrier is embedded in the design of a system and is independent of the operational control systems though still requires maintenance and follow-ups. An active barrier is dependent on human actions and depends on the technical control systems in order to function.

Smith et al., (Smith et al. 2004) highlight that there is commonly not a one-to-one relation between hazards and barriers. One hazard may be protected against by several barriers and one barrier may feature in the mitigation arguments of several hazards.

3.1.2.3 Barrier Violations

Building on the work in the field of barrier definitions, Polet et al., (Polet et al. 2003) and Vanderhaegen (Vanderhaegen 2003) taken the theory of barriers further by considering intentional violations of barriers and the resulting new states of stabilization resulting from these migrations (deviating from a barrier). A safety-related violation is defined by the authors as an intentional misuse or disobeying of a barrier provided that adequate conditions are present (Polet et al. 2002). Violations can be migrations or complete barrier removals. These are referred to as barrier crossings which can take the form of “border-line tolerated conditions of use” (BTCU). BTCU refer to undesirable behaviours, violations and undesirable behaviour resulting from socio-technical migrations.

This work focuses on a theory of the safety-related violations that occur in practice during normal operational conditions, but that are not taken into account during risk analysis. The safety-related violations are so-called barrier crossings. A barrier crossing is associated to an operational risk which constitutes a combination of costs: the cost of crossing the barrier, the benefit (negative cost) immediately after crossing the barrier, and a possible deficit (extreme cost) due to the exposure to hazardous conditions that are created after the barrier has been crossed. A utility function is discussed which describes the consequence-driven behaviour and uses an assessment of these costs functions (Polet et al. 2002).

Characteristics of BTCUs include:

- Non-respect of procedure
- Inhabitation of physical barriers
- Problems with manpower and/or training
- Intervention on running equipment
- Maintenance

Motivations for barrier crossings result from a combination of immediate benefit, immediate cost and possible deficit. To combat the problem of barrier crossings, the authors suggest that one way to improve the effectiveness of barriers is to increase the perception of the utility of barrier crossing; i.e. decrease the perception of the benefit and increase the perception of the costs and possible deficit

Vanderhaegen’s perspective has the same aims as our own, in that he advocate early consideration of potential migrations in order to improve the robustness of safety analysis techniques which in turn could provide improved design of safety-critical systems. However this work focuses on intentional human deviation behaviour, whereas our approach focuses on unintentional human and technical-related deviations. This perspective however, also considered in Laprie et al’s classification of combined faults (Laprie et al. 1995) is very interesting and should be considered in future work.

3.1.3 Formal Methods for Safety Analysis

This section ties together our interest in the use of formal methods, particularly Petri nets and approaches for incident and accident investigation. Primarily, we show how research into formal methods for safety analysis has focused on improving the usability of accident reports, reducing inconsistencies and omissions in reports, identifying scenarios leading to hazards and generally improving the quality of the accident reports. It appears that the most recent research in this area is during 2003, as the following paragraphs highlight, however the dated work is still relevant and will be detailed.

It has been recognised that systematic analysis should be undertaken for incidents and accidents and even near misses because of the greater amount, reliability and accessibility of information that can be made available in these situations (Kontogiannis et al. 2000).

3.1.3.1 Petri net-based Approaches for Safety Analysis

Leveson and Stolzy (Leveson and Stolzy 1987) used timed Petri nets (see Section 4.1.1.3) for safety analysis with timing constraints. The approach analyses the effects of failures and deviations of software safety requirements based on untimed reachability graphs. The aim is to identify and eliminate critical states. A critical state is one in which the system has multiple paths and where subsequent states include not only a hazardous state but also a safe one. Following the identification of critical states, the proposed method modifies Petri nets

with interlocks (see Section 3.1.2.1 on barriers) or timing constraints in order to eliminate the unsafe paths from the critical state to the hazard. The approach employed in this thesis has the same goal of identifying and eliminating hazardous states in a Petri net system model, however we do not only focus on timing issues. We also consider human interaction, operator procedures and human error to expose safety issues.

Continuing the work of Leveson and Stolzy, Cho et al., (Cho et al. 1996) propose a safety analysis method using coloured Petri nets (CPN see Section 4.1.1.3), with which different data types are assigned colours. The method employs a backward analysis approach where a hazard is assumed to have occurred and backward simulations from the hazard are performed in order to determine if and how the hazard might occur. In order to achieve their goal, the authors have extended the semantics of CPN by proposing symbolic markings to reduce complexity. They also define backwards reachability graphs of CPN which is a process for building marking graphs starting from a final state (instead of an initial state). The proposed approach is illustrated on the shutdown system of a Korean nuclear power plant. The goal of this work is to determine whether a system is free from a hazard that is to say that all states of a system cannot reach a hazard. In the approach, a hazard is defined as a symbol marking with a ‘don’t care’ condition, and backward reachability graph whose root node is the hazard generated. By ‘don’t care’, the authors are referring to the fact that the presence of tokens in certain places of the model are not related to the fact that the system is in a hazardous state. The condition represented by these additional tokens is considered unnecessary from the hazard point of view. One of the problems highlighted in this approach is the issue of too many, or infinite number of marking which represent a hazard in CPN. The values of tokens can have very high integer or enumerated values which means in practice, it is very hard to analyse all possible markings representing a hazard and furthermore, impossible when tokens have an infinite range such as real numbers. A lot of work, in the last 10 years, has been devoted to handle an infinite numbers of states and some tools are already dealing with it by constructing equivalence classes of markings i.e. grouping all together values of markings that have the same behavioural implications. For instance, if there is a precondition testing whether a value of a token is greater than 5, there will be two equivalence classes of markings (the ones that are less, and the one that is greater than 5). This allows reducing the potential infinite number of values for the integer into only two subsets. The ICO notation (See Chapter 1) can benefit from such approaches, however, this problem is more complex as ICOs allows tokens to hold not only basic types (integer, string etc) but also object types. .

In contrast to Leveson and Cho’s approaches of analyzing a known hazard, Cukic et al (Cukic et al. 1998) propose a formal method approach for predicting risk factors or system components of software, based on severity and complexity measures, again using CPNs. The CPNs are used to formally model software system specifications which are developed from system requirements and from risk analysis guidance. The models are analysed via sets of scenarios that are identified by experienced domain experts. The technique is based on input domain partitioning which means slicing of the CPN models, a concept that provides program simplification by identifying the statements which (may affect the value of a particular variable at a given statement (Cukic et al. 1998)). The authors argue that by disassembling the CPN model into slices allows the experts to notice and select scenarios that lead to system states with critical outputs. Furthermore, the approach ‘injects’ possible failure modes and faults represented in a FMEA into the CPN. This allows the authors to precisely study the failure propagation throughout the system by simulations.

3.1.3.2 Comparing Petri net Based Approaches with Other Formal Approaches for Safety Analysis

In (Botting and Johnson 1998) the authors use Object Z, an object oriented formal specification language for representing accident report data as well as for modelling operator tasks in order to incorporate human factors into a formal analysis of an accident. Using Object Z as a means for representing an accident report may be useful for representing information in a complete, concise and non ambiguous way however, analysts would require special training to be able to use such a technique. As illustrated in the paper, the notation is textual meaning it would be difficult to use to communicate with other parties of the investigation. Lack of tool support also means that simulations of the modelled accident are not possible.

In (Kontogiannis et al. 2000), the authors compare the use of Fault Tree Analysis (FTA), Sequentially Timed Events Plotting (STEP) and Petri nets with the aim of evaluating techniques for accident investigation that specifically aim to provide more complete accounts of critical events and human actions in terms of their underlying workplace and management causes. The authors produce the Petri net by first identifying the physical system, human agents and the messages communicated during the course of the accident. They state, the key concept is that the physical system, during its operation, sends messages to human operators, through the monitoring and alarm systems. These messages initiate human actors, some of which may change the state of the physical system (Kontogiannis et al. 2000) and is quite similar to Leveson’s control models in STAMP. In the model, the states of the physical system are represented by individual Petri net “modules”, the messages and

human actions are represented as places. Transitions are used to represent pre-conditions (signals or external events) and relationships between places. The models presented include a combination of the system behaviour, human behaviour and messages. Places labelled “pump-A-off”, “boss prepares A” and “high vibration alarm” illustrate this combination of behaviour.

The authors conclude that STEP and Petri nets appear to be superior for analysing temporal aspects of an accident sequence. Table 2-2 provides the results of the study. The only disadvantages discussed in relation to the use of Petri nets for accident analysis were that the graphs can become ‘unwieldy’ for complex scenarios and that improvement to tools should focus on providing capabilities for multiple levels of representation.

Table 2-2. A comparative assessment of the three accident analysis techniques (from (Kontogiannis et al. 2000))

A comparative assessment of the three accident analysis techniques

Criteria	Fault tree	Step	Petri Nets
Event sequence	***	***	***
Event agents	*	***	***
Event dependencies – cascade effects	*	***	***
Modelling the timing and duration	*	**	***
Multiple levels of representation	***	***	**
Modelling assumptions	***	***	***
Modelling inconsistencies	*	*	**
Cooperation facilitation	***	***	**
Event criticality	***	***	***
Modelling error recovery paths	*	**	***
Modelling the context of work	***	**	**
Preventive measures	***	***	***

In (Hill 2001), the author investigates the value of using graphical representations and hypertext to help resolve complexity in accident texts, and thus, aid their comprehension and eventually improve safety in safety-critical systems by learning from accidents. The authors introduce three graphical notations for analyzing the benefits of graphical - Petri Nets, Accident Fault Tress and Why-Because Graphs. As our system modelling approach is based on Petri nets, we will focus on the description of the Petri net based notations.

With respect to the use of Petri nets, the author reads an incident scenario and identifies the most important events. The identified events are mapped onto the Petri net syntax as states and events in accordance to previous work by (Johnson et al. 1995) where text that appeared to be describing preconditions to events are mapped onto Petri net states. Furthermore, text appearing to describe events initiated by operators, the system or the environment are mapped to events. This approach posed several problems, firstly the author is forced to fit a mapping between the text and the Petri net. Secondly, the author found a need to go beyond the information available in the incident report. The need to aggregate information from the report proved difficult and finally there was a problem of resolving temporal ambiguity.

In a three year project Johnson (Johnson 1999b) researched the use of formal methods to develop constructive techniques that support the production of accident reports. The main aims of this research were to develop a set of coherent principles that can guide the application of formal methods during accident investigations, perform a critical evaluation of temporal logic to improve an accident report and conduct a critical evaluation of executable temporal logics as a means of simulating the events leading to accidents. Some of the outcomes of this research are presented in (Johnson 2003). As well as Graphical Fault Trees, Fault Trees and Logic, Johnson proposes the use of Petri nets to reconstruct general systems failures that characterise safety-critical incidents. Similarly to Hill’s work (Hill 2001), Petri net places are used to represent conditions, causes, human behaviour, environmental attributes and behaviour of the individual systems during a mishap. The transitions refer to events that trigger a mishap. The method proposed provides a concise means for capturing events that lead to a mishap and it is argued that such a model can be used as a communication tool to show other enquiry participants. The method does not provide ways for identifying ways in which the mishap could have been avoided, though formal analysis techniques may be a possible solution to this. It also does not provide a solution to the problems of incident and accident analysis, problems such as missing information in the accident report, nor does it replace judgemental skills that Human Factors and Systems Engineers develop over time (Johnson 2003) p.287. The proposed use of Petri nets for representing incident and accident investigation analysis does not provide a means to move from the Petri net to remedies that can help prevent an incident recurring.

3.1.3.3 Combining formal methods and accident reports

Burns, (Burns 2000) proposes to use formal methods to model the behaviour of a system, as described in an accident report, for post hoc analysis. The research exploits ‘deontic action logic’ to construct models of accident reports in order to identify the ways in which mishaps often stem from the violation of ‘normal’ working practices. The notation highlights both the expected and actual behaviour in the report. It is argued that it facilitates examination of the conflict between the two. Burns uses Structured Common Sense (SCS) (Potts et al. 1986), a structured requirements engineering method, to guide the elicitation of detail from the report and for the construction of the formal model. However, more interestingly for us in particular is the framework for the formalisation of accident reports that was developed by Burns before adopting SCS. The application-oriented methodological framework incorporates hierarchical task analysis and Hollnagel’s classification of errors (see Section 3.5). Figure 2-7 provides an illustration of the approach. Starting with an accident report, a HTA performed to build a prescriptive model of the system behaviour. The second step takes Hollnagel’s classification of erroneous actions (Hollnagel 1993b) to categorise the errors documented in the report and to identify other potential errors – though the foundations of these ‘other’ errors not specifically detailed by the author. The outputs of these two steps are fed into the final step, which is the Extended Deontic Action Logic (EDAL) EDAL model.

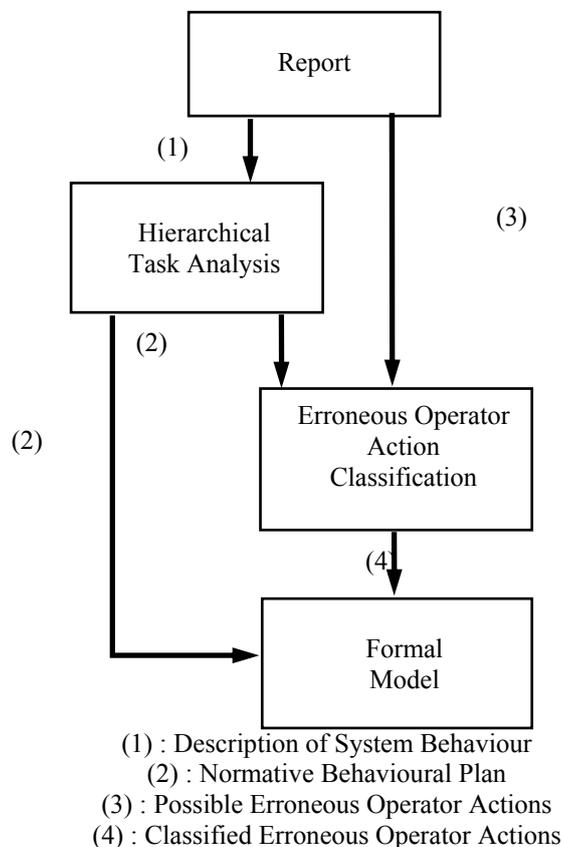


Figure 2-7. Framework for the Formalisation for Accident Reports (from (Burns 2000))

3.2 Anticipating Failure & Vulnerabilities

The previous sections in this chapter have discussed system safety, and how organisations can be more resilient as well as ways of representing and modelling incident and accident analysis data. This section will discuss methods for anticipating failures and vulnerabilities in a system, again to improve safety in design resulting in a more resilient interactive system. We distinguish between techniques for anticipating hardware failures, software failures and human failures. Human failures/error will be covered in more detail in the subsequent section, Section 3.5.

3.2.1 Hardware Failures

The safety and reliability of hardware can be compromised by different types of hardware failures. Of course, hardware safety can also be compromised by human intervention, poor organisational issues, poor maintenance

etc. The focus of the research is on Human Computer Interaction (HCI) and in particular human interaction with software applications and interfaces. Thus, this section will give a brief insight into hardware failures and faults, followed by several known fault tolerance and failure mitigation techniques.

The 6th annual Ernst and Young Global Information Security Survey of 2003, in which 1,400 organisations participated from 66 countries answered 40 questions, found hardware failures to be the number one cause of their “organization’s unexpected or unscheduled outage of critical business systems within the last 12 months”. Figure 2-8 illustrates the findings of question 36 of the 2003 survey (results are therefore for the year of 2002). The presented graph (taken from (Avizienis et al. 2004b)) is based on the original graph in the report though here the figures have been normalised to provide a 100% figure. Also, the findings have been annotated according to those which are malicious (19%) and those which are non-malicious (81%) (Avizienis et al. 2004b).

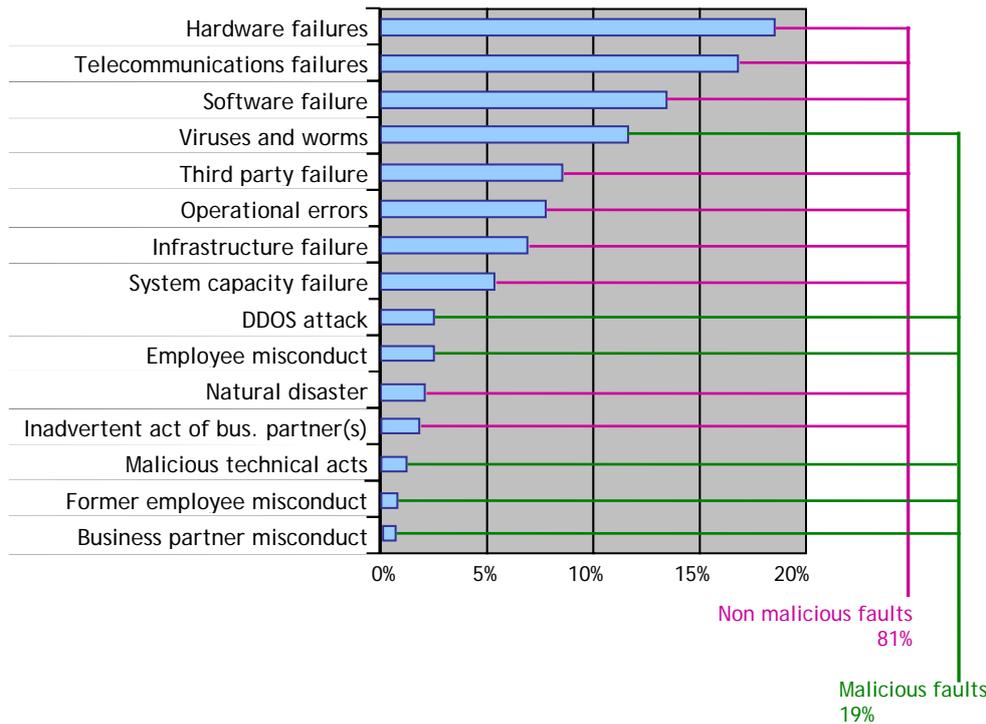


Figure 2-8. Results of Question 36 of the Ernst and Young Global Information Security Survey 2003 (from (Avizienis et al. 2004b)) based on original graph P.13 (Ernst & Young 2003a).

The report also states that 2003 showed no change with respect to the same question in 2002 (for the year of 2001), in which hardware failures were again the top cause of organization’s unexpected or unscheduled outage of critical business systems within the last 12 months. The 2004 report (for the year of 2003) again shows the same result, however the data available has been extended to include the origination of the loss of availability incidents. Table 2-3 details the top ten loss of availability incidents of the 2004 survey.

Table 2-3. Top ten loss of availability incidents. Results of the Ernst and Young Global Information Security Survey 2004 (from (Ernst & Young 2003b) p.14).

Incident	Occurrence	Origination		
		Internal	External	Unknown
Hardware failure	72%	87%	9%	4%
Major virus, Trojan horse, or Internet worms	68%	21%	76%	3%
Telecommunications failure	64%	26%	72%	2%
Software failure	57%	78%	16%	6%
Third party failure, e.g., service provider	47%	9%	87%	4%
System capacity issues	46%	91%	6%	3%
Operational errors, e.g., wrong software loaded	42%	91%	6%	3%
Infrastructure failure, e.g., fire, blackout	42%	49%	49%	2%
Former or current employee misconduct	24%	84%	12%	4%
Distributed Denial of Service (DDoS) attacks	23%	10%	85%	5%

3.2.1.1 “Natural” Hardware Failures

Here we focus on potential hardware failures that occur ‘naturally’, i.e faults that are caused by natural phenomena. For this description, we do not integrate human triggered failures as they have been dealt with explicitly in Chapter 1. Johnson (Johnson 1997) provides a good description of three types of possible failures. These include:

- **Transient failures:** Are failures that may occur periodically and may even rectify themselves. Such problems are difficult to identify because they may leave few traces and no lasting damage to the system itself. They are, however, typical of a latent error. If they go uncorrected then they may occur in a context which could lead to disaster.
- **Intermittent failures:** Intermittent failures appear, disappear and then reappear at a later time. They can be just as difficult as transient faults to detect because any testing and diagnosis must coincide with a moment of failure.
- **Persistent failures:** These are typically the easiest to diagnose because they continue for a prolonged period of time. This does not mean, however, that they are easy to diagnose.

Hardware failures, as described above, are considered different from hardware faults. Hardware faults fall under the category of physical faults (see Figure 2-10) (Avizienis et al. 2004a). Fault prevention is a complete domain of research and beyond the scope of this thesis. Figure 2-9 however and the following paragraphs present several established techniques for dealing with such hardware faults and failures

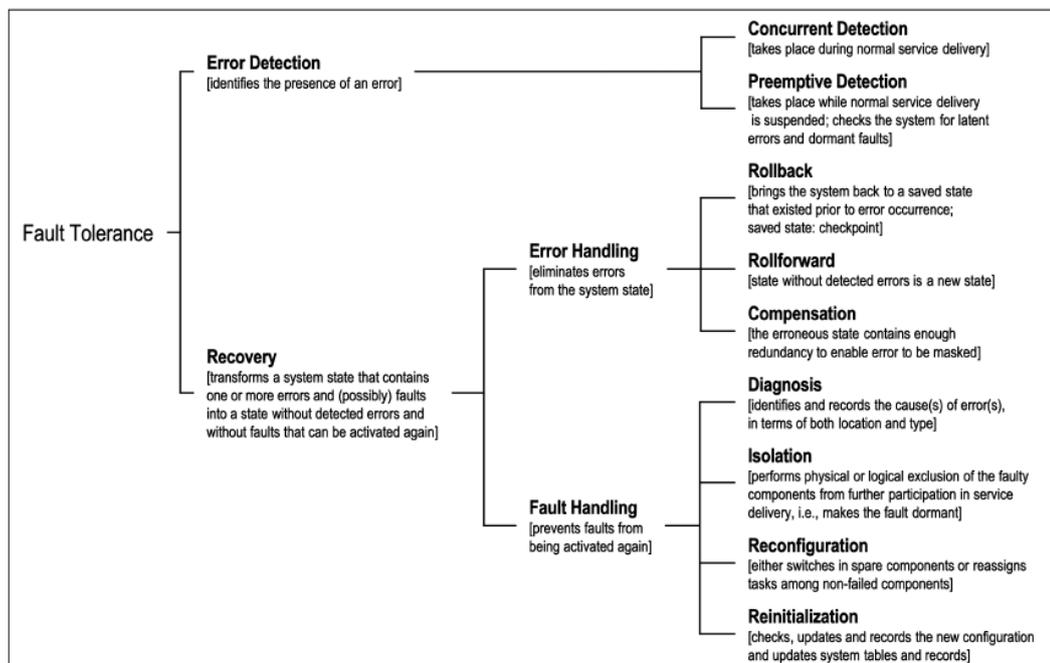


Figure 2-9. Fault Tolerance Techniques (from (Avizienis et al. 2004a))

3.2.1.2 Hardware Failure Mitigation Techniques

Also Johnson (Johnson 1997), presents a number of techniques that can be used to mitigate the effects of hardware failures which focus upon the twin problems of first detecting the failures and then resolving it.

- **Triple Multiple Redundancy:** Involves taking a 'vote' between several different pieces of hardware. If all of the components agree then the decision of the vote is unanimous. However, if the vote is not unanimous then a fault has occurred. The system will then, typically, follow the consensus (Johnson 1997).
- **Signal comparison:** In order for techniques, such as triple multiple redundancy to work, it must be possible to compare the output of different components during the voting process. This is difficult if a heterogeneous architecture is used; i.e the redundant components are not all the same. For example, if different manufacturers or designers supplied the components then the differences in the devices may mean that the results arrive in for the voting process at different times (Johnson 1997).
- **Information redundancy:** It is possible to check the integrity of a signal by encoding additional pieces of information into a signal. For example, imagine we wanted to send the following three numbers over

a network: 10, 22, 9. The recipient would have no means of checking whether they were correct or whether they had been changed through a hardware/software fault. We could reduce this uncertainty by adding add the numbers together and sending the sum as a fourth number: 10, 22, 9, 41. If the previous three numbers did not add up to 41 then either the numbers were corrupted, or the checksum was corrupted or there was a fault in the program used to check the calculation (Johnson 1997).

- **Watchdog timers:** Watchdogs simply decrement a counter after a preset interval of time. This continues until the counter reaches zero. If this occurs then the watchdog issue instructions to reset the monitored application. The monitored application, in contrast, will continue to increment the counter so that it is above zero. Only if the application crashes, will the counter start to fall towards zero (Johnson 1997).
- **Bus Monitoring:** This is a more sophisticated means of monitoring for hardware/software failure. Buses are used to transfer data between key components of the computer system. For example, a bus typically runs between the CPU (Central processing Unit) of the computer and its RAM (Random Access Memory). The CPU actually performs the calculations and the RAM acts as a temporary storage area for the results. Bus monitoring checks that any information being accessed in the computer's RAM is in an allowable area (permissible address). If not then the CPU is informed of an error (Johnson 1997).

3.2.2 Software Failures

The previous section argued that hardware failures are the most common cause of an organisation's unexpected or unscheduled outage of critical business systems according to organisation executives. However, this ignores the managerial role in procuring and maintaining the hardware. Also, distinguishing between hardware and software failures can be complicated when dealing with low level software directly responsible for handling hardware (for instance, failure in a device driver can be attributed to a failure of the input device). We also argued that hardware failures are not the main concern of the thesis. This section presents software safety issues. Software reliability is defined as "the probability of failure-free software operation for a specified period of time in a specified environment" (ANSI/IEEE 1991). It is important to note, that our interest in software safety is focused at a high level of abstraction, that of the user interacting with the finished application. Safety issues relating to programming and coding 'bugs' are beyond the scope and interest of this thesis. Clearly however, we recognise that similar interaction issues can occur during the development of the application. We present here an overview of software safety issues.

The growing complexity of software applications is increasing the level of risk in software. Ericson (Ericson 1999) describes many interrelated and complex factors that are contributing to this level of risk. These are:

- Increased usage of computers / software
- Increased dependency on computers / software
- Increased application to safety critical uses
- Significant difficulty developing software
- Significant difficulty preventing software bugs and errors

Furthermore, Ericson presents a substantial list of unique characteristics that make software difficult to work with including for example: "Software has many more failure paths than hardware, making it difficult to test all paths", and "by itself software can do nothing and is not hazardous. Software must be combined with hardware in order to do anything". We can place our interest in software safety issues within this second characteristic. Software alone without human intervention can do little harm.

Identifying hazards is key to software safety. Methods for identifying hazards include hazard analysis, specification and code correctness, software models, software testing, system testing and design tools (Ericson 1999). As described later in this thesis, a central element of the research is to determine methods for informing model-based design of a system with human safety elements. Although there are many ways for identifying hazards, we focus particularly on Ericson's first listed method, hazard analysis to identify human interaction hazards. Section 3.2.2.2 presents established software safety analysis methods.

3.2.2.1 Software Hazards

The safety of software can be comprised due to several factors. For example, bugs in programming, requirements flaws (This is the stage in software development where engineers must identify the constraints that a program must satisfy), design flaws (describing how those constraints are satisfied), implementation flaws and testing flaws (Johnson 1997). More specifically, Ericson (Ericson 1999) defines four types of software hazards.

- **Inherent Hazards** - A software controlled function which is inherently hazardous due to the hazardous nature of the equipment or process being controlled, such as hazardous materials or energy sources.
- **Timing Hazards** - Software controlled functions where the timing sequences are safety critical. This is often an overlooked area because many times sequences are taken for granted to be safe, until an accident/incident occurs.
- **Induced Hazards** - A software hazard caused by a computer hardware failure which causes a bit error, resulting in an erroneous instruction. For example, an intended word "1101" meaning "add to register A" may be changed by an induced bit error to "1011" which means "subtract from register A."
- **Latent Hazards** - A hidden condition in the software design which is not hazardous until a particular unplanned or untested set of circumstances occur (Ericson 1999).

In (Leveson 1986), Leveson, has argued that most accidents involving software are due to errors in the software requirements specification though Johnson and Holloway (Johnson and Holloway 2006) disagree. The authors use the term 'hindsight bias', which they say occurs when software engineers automatically assume that there has been a failure in requirements engineering simply because an accident has occurred. Furthermore, the term is exemplified stating "bias can be interpreted as influences that prevent objective consideration of an issue or situation". These influences can lead to or be reinforced by the use of logical fallacies to support the findings of accident investigations. In particular, they often seem to be used to justify the identification of requirements failure in the aftermath of software related incidents and accidents (Johnson and Holloway 2006).

The following section will provide a brief overview of established software safety analysis methods to deal with the above types of problems. We will then go on to discuss our interest in human-computer interaction issues relating to software.

3.2.2.2 Overview of Software Safety Analysis Methods

This section deals with hazard analysis methods targeted specifically at analysing software systems. The purpose of software hazard analysis is to make sure that the risk of the software failing in a hazardous manner is so small that it is acceptable, also in the case of e.g erroneous input, adverse environmental influence, failures of subsystems outside the software and software errors (Isaksen et al. 1996). Though these are all very useful for identifying software hazards, they are out of the scope of this thesis since we focus our attention on human factors issues. The human factors issues are presented in more detail in the subsequent sections. In large, Leveson (Leveson 1990) suggests four steps as a means of achieving software safety. These are:

1. Analyse procedures to identify hazards
2. Elimination and control of these hazards through hardware and software interlocks and other protective devices
3. Application of safety analysis techniques
4. Evaluation

Subsystem Hazard Analysis (SSHA) determines how subsystem design and behaviour can contribute to system hazards. SSHA also provides an opportunity to evaluate the design for compliance with safety constraints introduced earlier in the process (Leveson 2003a). The purpose of an SSHA is to identify hazards associated with design of subsystems including component failure modes, critical human error inputs, and hazards resulting from functional relationships between components and equipment comprising each subsystem. The purpose of an SHA is to determine the safety problem areas of the total system design, including interfaces, and potential safety-critical human error (Dryden Handbook 1999).

State Machine Hazard Analysis (SMHA) is a formal approach to safety analysis. It can be used to analyse safety and fault tolerance, to determine software safety requirements directly from the system design, to identify safety-critical software etc.

Software Fault Tree Analysis (SFTA) (Leveson 1995) can be applied throughout the development process. It is a traditional safety analysis technique that has proven to be an essential tool for software engineers during the design phase of a safety-critical software product (Leveson 1995). SFTA is a method for identifying and documenting the combinations of lower level software events that allow a top level event (or root node) to occur. When the root node is a hazard, the SFTA assists in the requirements process by describing the ways in which the system can reach that unsafe state (Leveson 1995). The method is a deductive, top-down method used to analyze system functionality.

System hazard analysis (SHA) builds on preliminary hazard analysis (PHA) as a foundation and expands upon it. SHA considers the system as a whole and identifies how system operation, interfaces and interactions between subsystems, interface and interactions between the system and operators component failures and normal (correct) behaviour, could contribute to system hazards. Through SHA, safety design constraints are traced to individual components based on the functional decomposition and allocation. Hazard causal analysis is used to refine the high-level safety constraints into more detailed constraints. This process requires a model of the system, even if that model is just in the head of the analyst. Causal analysis almost always involves some type of search through the system design (model) for states or conditions that could lead to system hazards. Search can be top-down or bottom-up and either forward or backward (Leveson 2003b). See Appendices for a diagram illustrating the difference between forward and backward search.

Before discussing software hazards relating to humans and human factors, we present a comprehensive diagram (see Figure 2-10) which illustrates in a systematic way, the types of faults that can occur in a system on three different levels, development, physical and interaction. This aids in understanding where our interests lie (i.e the interaction faults).

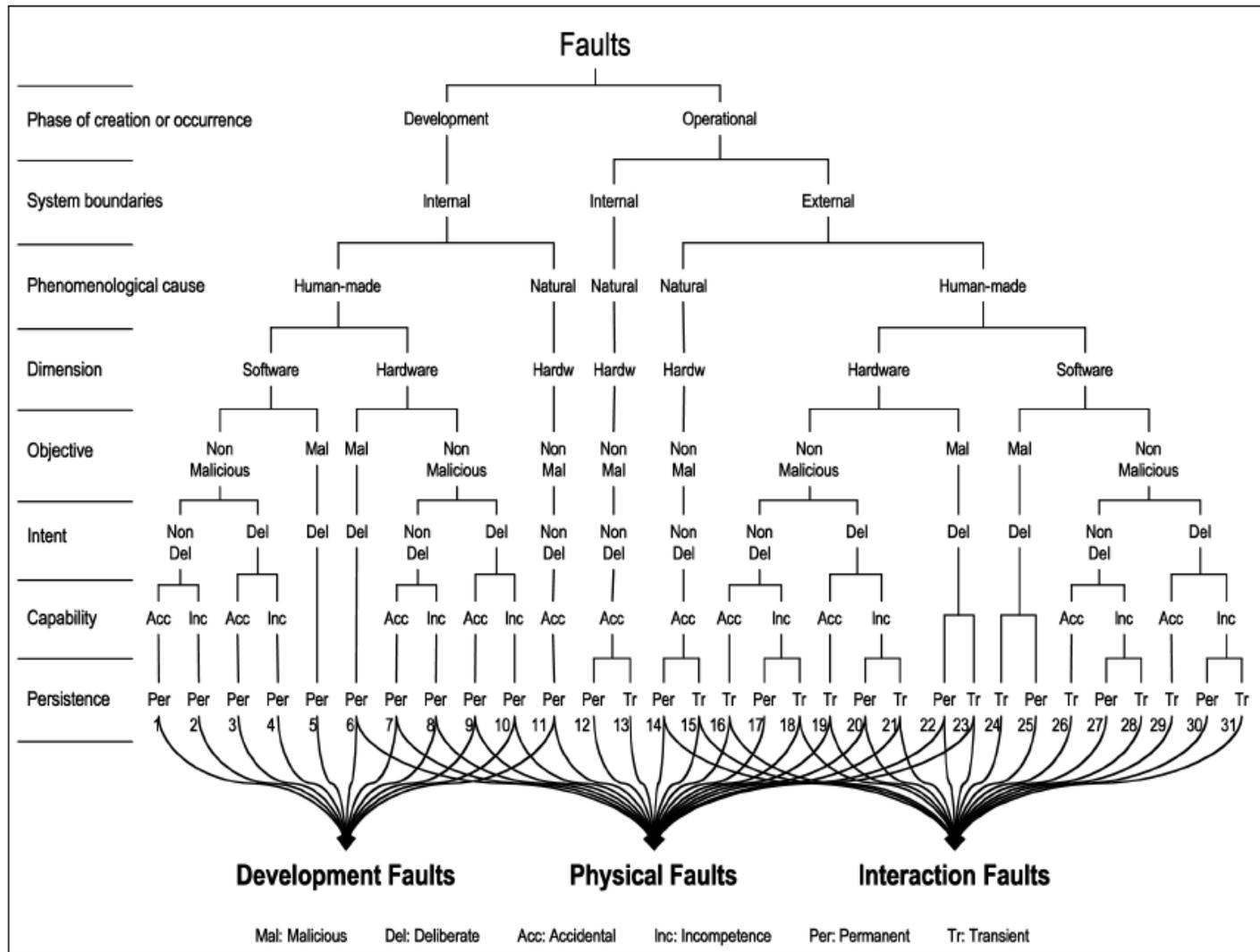


Figure 2-10. The classes of combined faults, tree representation (from (Avizienis et al. 2004a))

3.2.2.3 Application Interaction “Errors”

The design of an interface for a safety-critical interactive system raises unique challenges for designers and the HCI research community. A computer program is not unsafe when considered in isolation, it is only when it is integrated into a system (in which it can indirectly contribute to accidents) that software safety issues must be addressed (Garrett and Apostolakis 1998). We consider the human to be a part of the system and will address human factors issues in this section.

There exists extensive literature on types of human computer interaction failures that can occur while a user is operating an interactive system. Examples include mode confusions which are a kind of automation surprise (Rushby et al. 1999). Evidently, the consequence of such “errors” in a safety-critical context can potentially be devastating. The roots of such “errors” are often the result of poor design. Typically, there is a mismatch between the designer’s conception of the system and the user’s interpretation of the system. From an organizational perspective, such “errors” could also be a consequence of poor training, management etc.

Modes & Mode Confusion

This section presents a simple and small example of problems relating to software failures. Though the example is basic the root of some software failures can stem from such mode problems. However they clearly present a pedagogically bridge between system analysis and user errors.

A mode refers to the setting of a system in which inputs to the system result in outputs specific to that mode but not to other modes. Mode errors can be relatively benign when the number of modes is small and transitions between modes do not occur without operator intervention (Miller et al. 2005). Mode confusion or “automation surprise” (Palmer 1995) are pertinent sources of accidents, particularly in automated aircraft systems. An “automation surprise” occurs when the automation behaves in a manner that is different from what the operator is expecting.

Definitions of mode errors include, when the same [operator] action ... results in different system behaviour depending on what mode the machine is in (Palmer 1995) or when the operator executes a function that is appropriate for one mode of the automated system but not the mode that the system is currently in (Sarter 1997).

Mode confusion errors result from divergent controller models (Leveson et al. 1997). The root of mode confusion, according to Leveson et al., is from divergent controller models. Similarly, Brederke & Lankenau believe that if the relationship between the mental model and the machine must is not satisfied from specification to implementation, mode confusions can occur (Brederke and Lankenau 2002).

Degani et al., (Degani et al. 1999) have identified features that can lead to mode confusion. These include:

- The relationship between mode ambiguity and the user task. If distinguishing between modes is not part of the user’s task, no meaningful errors will occur.
- If the user has an inaccurate and/or incomplete model of the machine’s response map mode confusion could occur
- The interface must provide the user with all the necessary input events. If it does not, no accurate and complete model will help—the user may know what to look for but will never find it. As a result, confusion and mode error will occur.

Leveson et al. have also identified six categories of system design features that can lead to mode confusion errors: ambiguous interfaces, inconsistent system behaviour, indirect mode transitions, lack of appropriate feedback, operator authority limits, and unintended side effects (Leveson 1995).

Degani and Leveson et al., have classified modes into three categories. Degani classifies them as

- Interface modes, which specify the behavior of the interface. They are used to increase the size of the input or output space
- Functional modes, which specify the behavior of the various functions of the machine
- Supervisory modes which specify the level of interaction or supervision (manual, semi-automatic, or automatic).

Leveson et al., (Leveson et al. 1997) have used the following classification:

- Supervisory modes, which determine who or what is controlling the component at any time. Each component may have multiple controllers (supervisors). Mode awareness errors related to confusion in coordination between the multiple supervisors of a control component can be defined in terms of these supervisory modes.
- Component operating modes control the behaviour of the control component itself. They may be used to control the interpretation of the interface or to describe its required process-control behaviour.
- Controlled-system operating modes specify sets of related behaviours of the controlled system and are used to indicate its operational state.

The second classification of modes is defined with respect to the control component that is being specified. Component operating modes are the equivalent of Degani's interface modes.

In (Hughes and Dornheim 1995), 184 aircraft incidents and accidents involving mode awareness are listed. In the complexity of modern, computerized systems, the current system state can come as an unpleasant shock to even an experienced user (Hourizi and Johnson 2001). This relates to non-deterministic interfaces, which are those where 'the same user action can lead to more than one outcome' (Degani 2004).

The above described modes and mode confusion results as stated because of a mismatch between the user's mental model and the model of the system. The problem relates to poor system design. One way of minimizing the likelihood of mode confusion without changing the system behaviour could be to provide feedback of the current system status on the interface. This however, is a 'cheap and dirty' option and modifications to the system would be recommended. This shows however the importance interface design has on usability and increasingly the possibility of interaction "errors". Even if a system behaves correctly, according to the system requirements, a poor interface can have life threatening consequences in a safety-critical environment.

Over the past ten to fifteen years, several authors have become leaders in the field of interaction design providing quality principles for interface design. Authors include Jacob Nielsen and Don Norman who have provided methods for improving interfaces for better usability human-computer interaction. These usability approaches are presented in section 3.3 of Chapter 1.

3.2.2.4 Mitigating Human Interaction 'Errors'

This section is dedicated to the description of the interaction hazard identification techniques including related work in the field of software safety analysis. In (Falla 1997), it is argued that current software safety standards provide little advice regarding threats which ought to be considered when undertaking software hazard analysis such as environmental and operating conditions, logic control, real time executive, system function calls, system resources, timing and software design notations. Although all these types of hazards are important, we focus our attention on hazards relating directly to UI interpretation problems and ways of improving the UI to assist the operator. It was found in (Falla 1997), that "optimum [software safety analysis] results were obtained when techniques including Fault Tree Analysis (FTA) (see Chapter 2 Section 3.1.2), Failure modes and effects analysis (FMEA) (see Chapter 2 Section 3.1.2), and Hazard and Operability Studies (HAZOP) (see Chapter 2 Section 3.1.2), were used in a coherent fashion as part of an integrated safety assessment method" p.6 chapter 3. Although the authors apply HAZOP to an existing software design, the approach is based on data flow diagrams.

Risk assessment methods such as HAZOP, Failure Modes and Effects Analysis (FMEA), etc. have been adapted for a design orientation (Earthy 1995). Though these methods do consider human influences on failure modes, they do not deal with complex causes of human error (Hollnagel 1993a). A more systematic and theoretically grounded approach to safety design for human operation is required (Hollnagel 1993a) (Reason 1990).

In closer relation to our approach, Leveson et al., (Leveson et al. 1997b) present an approach for analysing the safety of the Center-TRACON Automation System (CTAS) portion of an Air Traffic Control (ATC) system. CTAS is a set of tools designed to help air traffic controllers manage the increasingly complex air traffic flows at large airports. The approach aims to analyse the effect the system changes have on human "errors" in order to modify the system and improve user training to reduce their impact. The approach looks at the automation as a way of evaluating its potential to contribute to human "error". The focus of this work is on automation and on mode confusion. Six "categories of potential design flaw" are identified including interface interpretation error, inconsistent behaviour, indirect mode changes, operator authority limits, unintended side effects and lack of appropriate feedback. Our approach builds and extends this kind of work as it does not focus on only mode confusion but on human-computer interaction failures because of cognitive "errors" and poor design.

Furthermore, Leveson et al., (Leveson et al. 1997) propose an approach for detecting error-prone automation features early in the development process. The approach builds on previous work by Sarter and Woods, and Degani (see Chapter 2 Section 3.2.2.3) by defining 50 completeness criteria for specifying requirements. The requirements include heuristics for identifying problems, more specifically typical HCI problems such as providing appropriate feedback. Among the 50 criteria are the six categories of potential design flaw listed in the previous paragraph. The software requirements are modelled using state machines. An analysis of is made of design constraints associated to mode confusion errors.

Sutcliff and Rugg, (Sutcliffe and Rugg 1998) have proposed a taxonomy of influencing factors that might contribute to human error. More recently, (Galliers et al. 1999) the components of the taxonomy have been combined into a causal model for error. The authors use Bayesian Belief Nets to represent their model which is used to model error influences arising from user knowledge and the task environment. With respect to human errors, the method only considers slips and lapses. Section 3.5 of this chapter will illustrate that there are far more possible errors to be considered in design.

To conclude this section, we will discuss an error analysis approach that uses formal specification of safety critical interactive systems in order to identify aspects of the user interface that could contribute to producing an accident, based on a formal system model. The approach, developed by Andrew Hussey has very similar goals to our own.

In (Hussey 2000), Hussey considers methods for analyzing interactive systems for operator errors leading to hazards. The language used is SUM (Johnston and Wildman 1999) a variant of the Z specification language for task specifications, actions and objects. Furthermore, Possum (D. Hazel et al. 1997), a model-checking CASE tool is used to explore the system state following a HAZOP guideword deviation. The proposed approach has three main steps:

1. Identify user goals based on operational scenarios (similar to THEA, see section 3.6 of this chapter)
2. For each goal, potential action sequences (tasks) are identified. Based on the tasks a model of the interactive system is constructed for hazard analysis.
3. Using HAZOP guidewords, examine tasks to identify user interface features that may cause accidents

Hussey formally models an industrial case study and demonstrates how the formal model and the task model can be used as a basis for a HAZOP analysis. The behaviour of the system is explored by looking at each action sequence identified in scenarios. Every action has corresponding failure actions associated to it based on the HAZOP guidewords. The failure actions can then be substituted with the original correct actions to analyse the potential outcome on the system.

In this section, on anticipating failures and vulnerabilities, we have presented issues relating to hardware and software failures, as well as mitigation techniques for the two via analyses. Additionally, we have related theories on human error to hardware and software failures by discussion potential interaction failures by giving an example on modes and mode confusions. We terminated this section by providing literature relating to the mitigation of such interaction 'errors'.

The following section provides a means dedicated to explicitly dealing with safety during the design of a safety critical interactive system. It can be considered a further means to anticipate failures and vulnerabilities. We present safety cases as a useful technique for documenting system safety arguments necessary for certification phases of the design process.

3.3 Means to argue safety: safety cases

This section presents safety cases as a means of explicitly dealing with safety in the development process of a safety critical system. Though there are other techniques available, we focus on safety cases since this are used as part of our framework though it is not a field we directly contribute to and therefore will not present alternative methods.

Safety cases are documentation often required by regulatory agencies before they will grant approval for the operation of complex safety-critical systems. These cases describe the arguments that a company will use to convince the regulator that their proposed application is 'acceptably safe' (Bloomfield et al. 2004). For instance, they can include information on the risk assessment techniques that have been used. They may also describe the barriers that are deployed as a result of a risk assessment.

Many countries require that safety cases demonstrate a system is ‘acceptably safe’ before they grant regulatory approval. These documents and the associated analytical techniques, therefore, provide a rich source of information about why command and control failures occurred.

Safety cases build up an argument that any risks associated with an application are as low as reasonably practicable. These arguments can be based on assumptions about the relationship between an application and other systems. They can also be based upon assumptions about the environment that a system will operate in. These conditions can be difficult to control and hence can force revisions to a safety case in order to demonstrate that the system continues to be safe within revised operating conditions. Changes in operating practices or the functional demands on a system can introduce the same need for revised argumentation.

In cases where a revised argument cannot be made then it can be necessary to introduce design changes. A continuing problem here is that it is difficult to ensure the modularity of a safety-case. Hence any changes within the design of a system sub-component will often force renewed inspection of the argumentation associated with the safety cases of other related components. If the changes cannot be accommodated within the argument then redesign may be necessary.

In order to implement safety cases, one must make an explicit set of claims about the system and produce supporting evidence. A set of safety arguments is required that links the claims to the evidence. Assumptions and judgements underlying arguments must be made clear. The main elements of a safety case are as follows:

- Claim: about a property of the system or some subsystem
- Evidence: which is used as the basis of the safety argument
- Argument: linking the evidence to the claim
- Inference: the mechanism that provides the transformational rules for the argument

Claims can include various attributes such as reliability, security, functional correctness, time responses and maintainability. While arguments can also vary from deterministic, probabilistic and qualitative arguments, information needed to produce safety cases can be obtained from sources such as the design, the development process itself or prior field experience.

3.3.1 Representing safety cases

Goal Structuring Notation (GSN) (Kelly and Weaver 2004) is a safety argument notation which provides a graphical semi-formal means of representing and reasoning about safety arguments. The notation helps explicitly represent individual elements of a safety argument. The following diagram provides an idea of the symbols used within the notation.

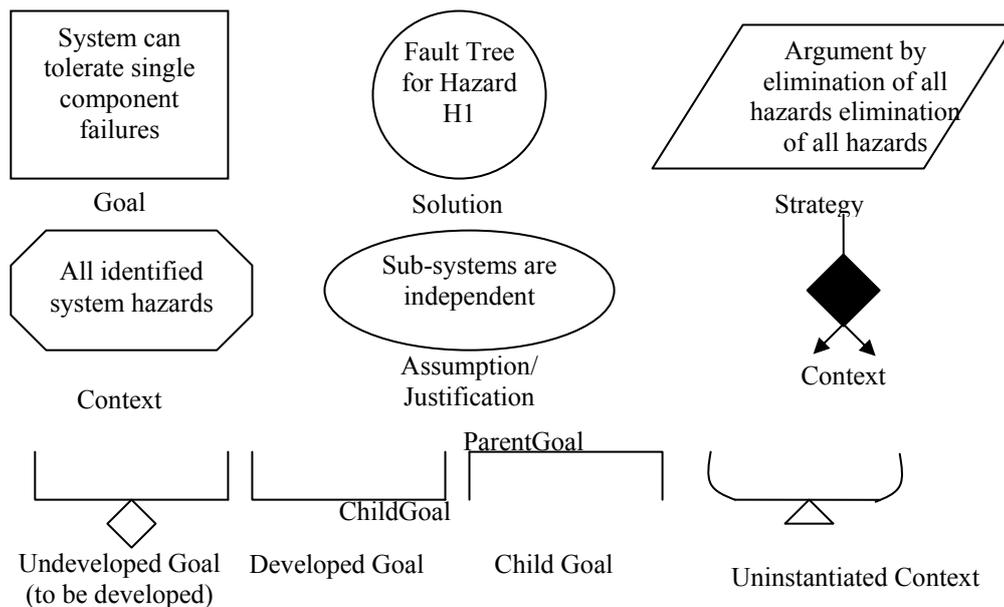


Figure 2-11. Principle Elements of the Goal Structuring Notation (taken from (Kelly 2003))

3.4 Human Factors in the Design of Safety Critical Interactive Systems

As mentioned in the system safety section (Chapter 2 Section 2.1), humans must also be considered within the development process of safety-critical interactive systems design. Clearly, humans are involved during the design phases as well as during implementation and in system operation. Mistakes in the production or installation of a system could render it dangerous, as could problems associated with its use or decommissioning (Storey 1996). We also discussed in Section 3.1.1 of this chapter that human “error” is often the root cause of incidents and accidents. In this section we will briefly introduce recent methods and approaches from the literature for accounting for human factors issues in the design process of an interactive safety critical system. Following which, a more detailed discussion on human error, error management and error analysis models is provided since this is our particular area of interest. We will then tie together task modelling (presented in Chapter 1) with approaches for mitigating or minimising the likelihood of human “error”.

The International Ergonomics Association (IEA) defines human factors as follows. Ergonomics (or human factors) is the scientific discipline concerned with the understanding of the interactions among humans and other elements of a system, and the profession that applies theoretical principles, data and methods to design in order to optimize human well being and overall system performance. Practitioners of ergonomics, ergonomists, contribute to the planning, design and evaluation of tasks, jobs, products, organizations, environments and systems in order to make them compatible with the needs, abilities and limitations of people (International Ergonomics Association (IEA) 2000).

Domains of human factors research include physical ergonomics, relating to anthropometrics and physiological and biomechanical characteristics for example. Another branch is the field of cognitive ergonomics, concerned with mental processes, such as perception, motor response and mental workload. On a larger scale, is the domain of organisational ergonomics, interested in optimizing sociotechnical systems by looking at organisational structures, processes, crew resource management etc.

Though accounting for such human factors issues is important in the design of safety critical interactive systems, we are particularly interested in issues relating to human cognition, and more specifically human ‘error’ issues as opposed to factors relating to the physical environment for example. The follow section will present established theories in the domain of human errors.

3.5 Human Error

This section presents a complementary perspective, with respect to previous sections discussing hardware, software and interaction failures by addressing psychology theories on human ‘error’. We present four theories on human ‘error’ that will contribute to approaches used within the framework of the PhD .

Human error plays a major role in the occurrence of accidents in safety-critical systems such as in aviation, railways systems, or nuclear power plants (Reason 1990). Generally speaking, human error can be tackled in two ways, identifying the problem and blaming one or several humans before terminating the investigation as this has been discussed in previous sections. Suggestions following this kind of approach could be improved training or firing of an employee. The recent rail disaster in North Korean in which more than 160 people died and 1,300 were wounded in a train blast at Ryongchon, near the Chinese border has been officially blamed by their Government on human error (<http://news.bbc.co.uk/1/hi/world/asia-pacific/3656853.stm>). A second more appropriate approach is to use the identification of human “error” as the beginning of an in-depth analysis in order to improve future design. We believe the second approach is correct and is the key to resilient systems. Interactive systems particularly those that are safety-critical need to be designed with the eventuality of human error in mind to prevent catastrophes. This means, early in the design process and as well as during the testing phase. Not only can humans have an impact incidents and accidents, they can also influence the dependability of a critical system during, the development process, the deployment process, the maintenance/management process and the operational process (Sommerville 2003).

Reason (1997) defines human error as “the failure of planned actions to achieve their desired ends – without the intervention of some unforeseeable event”. We believe the term “human error” is controversial. The term falsely implies that the human is to blame for the problem when what it really is, is a human action in an adverse situation. Hollnagel clarifies this by stating, the fundamental semantic problem is that the term “human error” has at least three different denotations, so that it can mean either the cause of something, the event itself (the action), or the outcome of the action (Hollnagel 2001). Furthermore, Hollnagel draws attention to the fact that

even the term action is underspecified. Thus based on (Amalberti (1996)), a further classification of actions is proposed by Hollnagel. These are:

- Actions for which the actual outcome matches the intended outcome, i.e., actions that seem to achieve their goal. These actions are usually regarded as correctly performed actions, hence give little cause for concern, even though it is possible that the outcome came about in other ways.
- Actions that are perceived as having been carried out incorrectly in some way, but where the discrepancy is detected and corrected. This can either happen as the action is being carried out, where typing mistakes are a typical example, or immediately after as long as the system makes a recovery possible. If the system is sufficiently forgiving, the actual and intended outcomes may still match and the action may therefore for all intents and purposes be considered as correct.
- Actions that are recognised as being carried out incorrectly, and where recovery is not possible. A recovery can be impossible for several reasons, for instance that the system has entered an irreversible state, that there is insufficient time or resources, etc. In these cases the actual and intended outcomes do not match, and the action is therefore characterised as an error.
- Actions that are recognised as being carried out incorrectly, but where the discrepancy is ignored. This usually happens because the person considers the expected consequences of the action failure as unimportant in an absolute or relative sense. This assessment may either be correct or incorrect, depending among other things on the users' knowledge of the system in question. If it turns out that the consequences were not negligible, the action is in retrospect classified as an error.
- Actions that are carried out incorrectly, but which are not detected at the time, and therefore not recovered.

The remainder of this section will detail several theories of human "error" and their classification schemes which have been widely accepted in the literature. We will also highlight that human "error" is often the result of poor misleading design even though, at times, operators may be to blame.

3.5.1 Human Error Theories

The theories presented in this section will be used to structure our arguments on reasoning about human error in the task modelling and system modelling phases of design and will also directly contribute to approaches within the framework of the PhD.

The main theories accepted in the literature and often referred to are Rasmussen's SRK (Rasmussen 1983) theory, Reason's GEMs model (Reason 1990), Norman's Slips and Lapses (Norman 1988) and Hollnagel's phenotypes and genotypes (Hollnagel 1991). Each is detailed below.

3.5.1.1 SRK

Rasmussen (Rasmussen 1983) distinguishes between three levels of human processes each with its associate error types in his Skill-based, Rule-based and Knowledge-based (SRK) theory: (1) skill based level, for activities performed automatically, (2) rule based level, for circumstances in which our intuition provides an applicable response, and (3) the knowledge based level, for new situations in which there are no rules. The terms skill, rule and knowledge based information processing refer to the degree of conscious control exercised by the individual over his or her activities.

It can be summarised as:

- **Skill-Based:** Unintended deviations from the procedures that are conducted automatically by the individual. This refers to the execution of highly practiced, mostly physical actions for which conscious monitoring is not necessary. Performance is smooth, automated, and consists of highly integrated patterns of behaviour in most skill-based control (Rasmussen 1990).
- **Rule-Based:** Associated with those activities in which the individual has to consciously choose between alternative courses of action. A rule-based behaviour is characterised by the use of rules and procedures to select a course of action in a familiar work situation (Rasmussen 1990). The rules may have been learnt via operation of a system, or via training or by learning from colleagues. The level of consciousness required can be considered in between knowledge and skill based modes.
- **Knowledge-Based:** The individual attempts to define a new procedure on the basis of knowledge about the system they are using. The human carries out the task in a conscious manner. A knowledge-based behaviour represents a more advanced level of reasoning (Wirstad 1988).

Figure 2-12 illustrates the placement of the three performance levels with respect to the amount of consciousness required and the likely situations. Rasmussen introduced the SRK model as a tool for systematic description of how human intervention performs in the environment of high risk work domains, such as power plant control (Albrechtsen et al.).

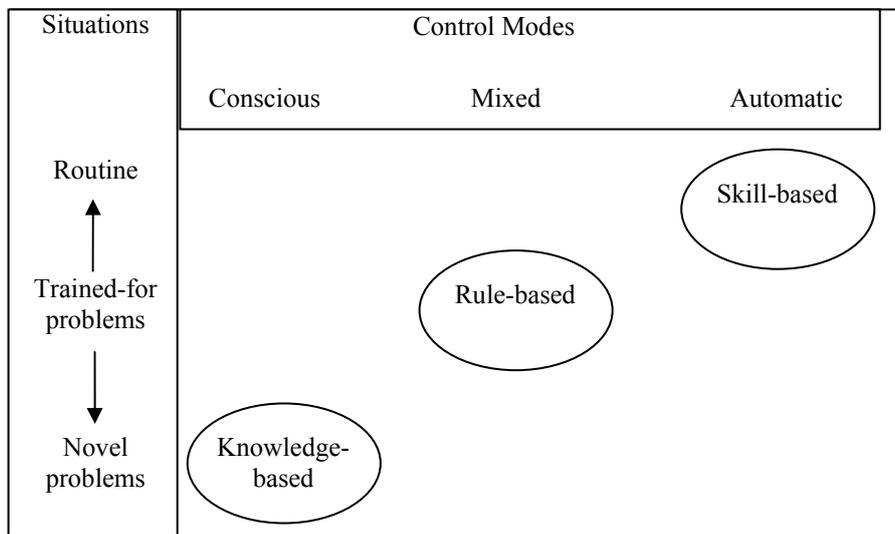


Figure 2-12. Location of the three performance levels within an ‘activity space’ defined by the dominant mode of action control and the nature of the local situation (from (1997))

3.5.1.2 GEMS

Based on Rasmussen’s SRK theory, Reason developed the Generic Error Modelling System (GEMS) (Reason 1990) model which relates typical error types with error-shaping factors.

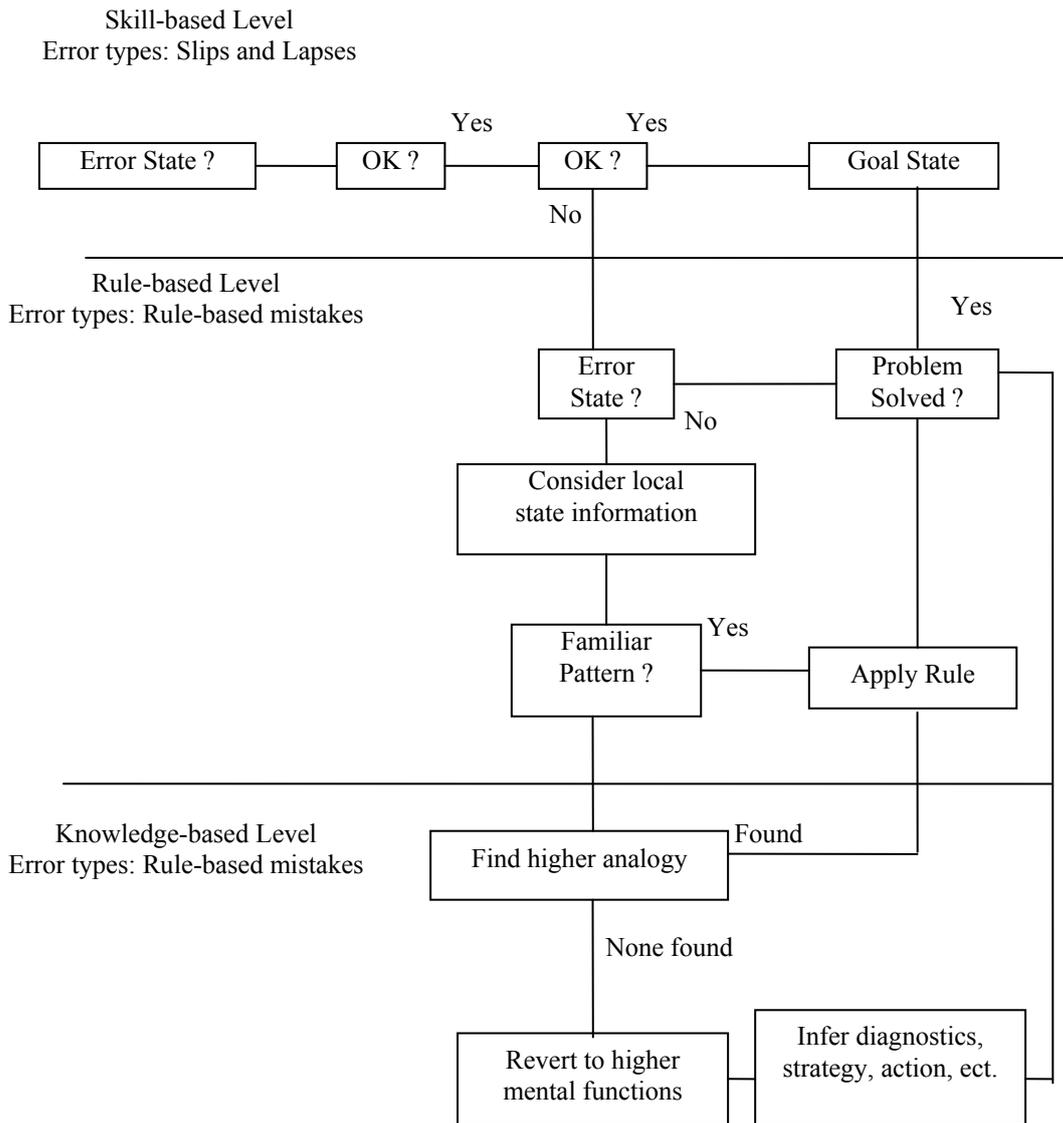


Figure 2-13. The Generic Error Modelling System (GEMS) (Reason 1990)

The GEMs framework aims to show how users switch between the SRK information processing types while undertaking tasks. The approach does not consider environmental factors or any other context-related factors, rather cognitive factors in humans. The process of switching information processing types is presented within a hierarchy of three levels of cognitive involvement (see Figure 2-13). GEMS also models the points in which slips, lapses and mistakes typically occur and the reasons for breakdowns in interaction. As shown in Figure 2-13, slips and lapses occur during skill-based interactions, while mistakes occur during rule and knowledge based interaction. For further explanation, Figure 2-14 presents the key features of the GEMS framework. The table provides an overview of the types of activity that occur during skill-based, rule-based and knowledge-based errors, the predictability of errors, their ease of detection etc.

DIMENSION	SKILL-BASED ERRORS	RULE-BASED ERRORS	KNOWLEDGE-BASED ERRORS
TYPE OF ACTIVITY	Routine actions	Problem-solving activities	
FOCUS OF ATTENTION	On something other than the task at hand	Directed at problem-related issues	
CONTROL MODE	Mainly by automatic processors (schemata)	(stored rules)	Limited, conscious processes
PREDICTABILITY OF ERROR TYPES	Largely predictable "strong-but-wrong" errors (actions) (rules)		Variable
RATIO OF ERROR TO OPPORTUNITY FOR ERROR	Though absolute numbers may be high, these constitute a small proportion of the total number of opportunities for error		Absolute numbers small, but opportunity ratio high
INFLUENCE OF SITUATIONAL FACTORS	Low to moderate: intrinsic factors (frequency of prior use) likely to exert the dominant influence		Extrinsic factors likely to dominate
EASE OF DETECTION	Detection usually fairly rapid and effective	Difficult, and often only achieved through external intervention	
RELATIONSHIP TO CHANGE	Knowledge of change not accessed at proper time	When and how anticipated change will occur unknown	Changes not prepared for or anticipated

Figure 2-14. Key Features in GEMS ((Reason 1990)p62)

Using SRK and GEMS, specific failure modes such as mistakes and slips and failure types can be obtained from the models. An evaluation can be performed on the effect a human error can have and in which type of error model.

3.5.1.3 Phenotypes and Genotypes

Hollnagel's (Hollnagel 1991) phenotypes and genotypes are an error taxonomy based on terms from biology. Professor John Blamire of the Department of Biology at Brooklyn College, New York, provides the following definitions of phenotypes and genotypes (Blamire 2000) just as the term "resilience" is borrowed from the field of biology.

Phenotype: The "outward, physical manifestation" of the organism. These are the physical parts, the sum of the atoms, molecules, macromolecules, cells, structures, metabolism, energy utilization, tissues, organs, reflexes and behaviours; anything that is part of the observable structure, function or behaviour of a living organism.

Genotype: The "internally coded, inheritable information" carried by all living organisms. This stored information is used as a "blueprint" or set of instructions for building and maintaining a living creature.

In terms of Hollnagel's definitions, the genotype of an error relates to its origin. The source or genotype of an error lies within the cognitive process and not with the implementation of the task. The error *genotype* denotes the mental mechanism assumed to underlie the observable erroneous behaviour. Conversely, the phenotype of an error relates to its physical outward manifestation when the genotype is translated into observable action. The observable phenomena, he states, make up the empirical basis for error classification. He refers to these behavioural descriptions as the *phenotype* of human error. The phenotype is fully observable and potentially measurable. Hollnagel identifies eight simple phenotypes and seven complex phenotypes (Hollnagel 1993a).

Eight simple phenotypes

- Repetition of an action
- Reversing the order of actions
- Omission of actions
- Late actions
- Early actions
- Replacement of one action by another
- Insertion of an additional action from elsewhere in the task
- Intrusion of an additional action unrelated to the task

Seven complex phenotypes

- Restart
- Jumping
- Undershoot
- Side-Tracking
- Capture
- Branching
- Overshoot

Based on the defined phenotypes, Hollnagel presents a taxonomy of phenotypes of erroneous actions. The taxonomy is presented in Figure 2-15.

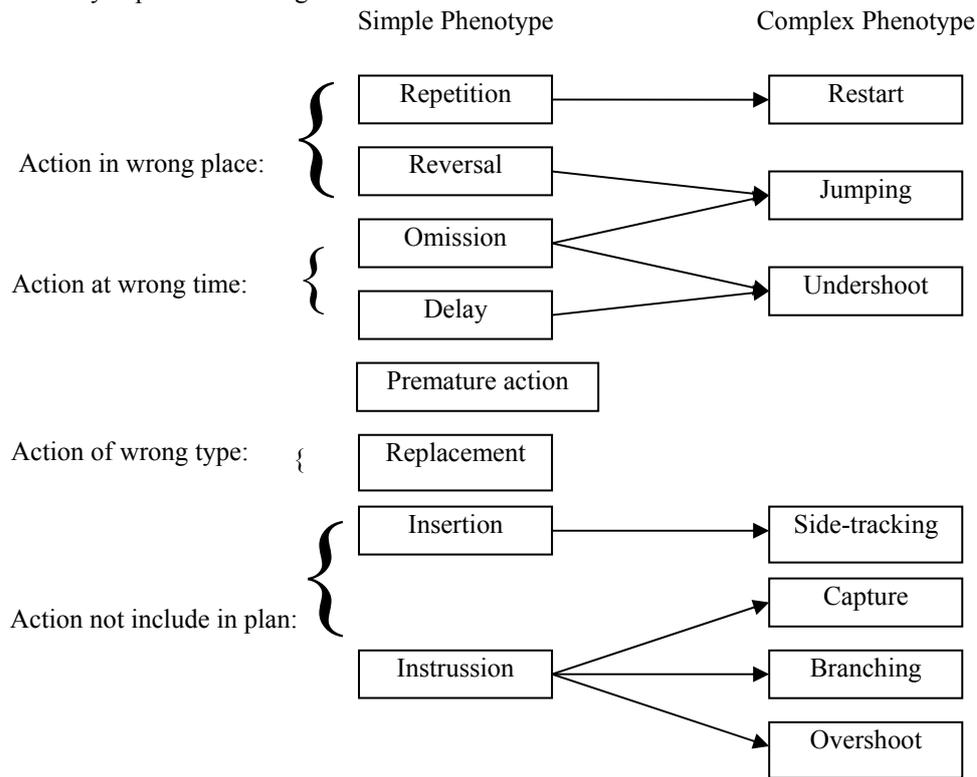


Figure 2-15. Hollnagel's taxonomy of phenotypes (from (Hollnagel 1993a) p.76(Hollnagel 1991))

3.5.1.4 Slips and lapses

Don Norman, distinguishes between two fundamental categories of errors, slips and mistakes. Slips result from automatic behaviour, when subconscious actions that are intended to satisfy our goals get waylaid en route. Mistakes results from conscious deliberations. Slips are almost always small things such as misplaced actions while mistakes can be major events and are difficult if not impossible to detect since the action performed is appropriate for the goal (Norman 1990) p.105 (originally (Norman 1981)).

In an extension to the slips and mistakes definition, Reason defines slips, lapses, mistakes and violations (1997). The difference between a slip and a lapse is that a slip is observable and a lapse is not. Figure 2-16 provides an overview of these error types in relation to the skill-based, rule-based and knowledge-based errors defined earlier in this section. The diagram does not include violations which can be either deliberate or erroneous. With respect to deliberate violations, Reason defines three major categories of safety violation, routine (corner cutting), optimizing (to serve a variety of motivational goals) and necessary violations (non-compliance is seen as necessary to get a job done).

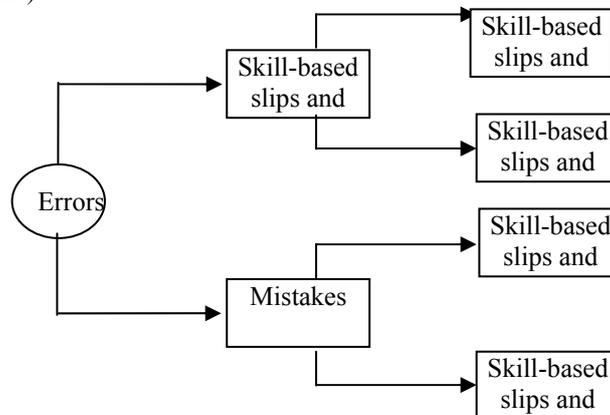


Figure 2-16. Summary of the principle error types (from (1997) p.72)

Hollnagel states that the distinction between phenotypes and genotypes is often emphasised by these common definitions of slips and lapses, and mistakes (Hollnagel 1993a).

3.5.2 Error Management

When designing safety-critical interactive systems, we should design to avoid error, to detect errors and to recover from errors. Of course this means both technical and human “errors”. Here we focus on human “error” management for the design of error tolerant and error resistant systems. This section looks at techniques and methods from the literature for managing human “error”.

Recently, Fujita wrote “Humans in systems (e.g. operators, maintenance people) are in general, adaptive and proactive. These are admirable qualities, but of limited scope. Adaptive and proactive behaviours can change systems continuously, but humans at the front end alone may or may not be able to recognise the potential impact that the changes can have on the system, especially the impact when several changes are put into effect simultaneously. Humans at the back end (e.g. administrators, regulators) tend to have sanguine ideas such as that the system is always operated as planned, and that rules and procedures can fix the system at an optimal state. Mismatches caused by these two tendencies constitute latent hazards, which may cause the system to drift to failures” (Fujita 2006).

The above paragraph introduces many interesting and relevant concepts. The fact that a system is constantly changing, the fact that operators have a misconception of what rules and plans can do to help and the fact that these problems can lead to system drifts to danger. These concepts and ideas will be discussed in this section.

3.5.2.1 Error Resilience and Error Tolerance

Dearden and Harrison (Dearden and Harrison 1995) make clear the difference between human error resistance and human error tolerance. According to the authors, a system is defined as human error resistant if it ‘reduces the probability that a human operator interacting through the interface will attempt to perform actions that endanger the safety of the system’. A system is defined as human error tolerant if ‘the consequences of a human operator performing erroneous actions through the interface are limited to some acceptable level’. The terms limited and acceptable level are subjective and would depend greatly on the system in use.

We now present techniques that have been developed for considering human error in the design process. These include human reliability assessment techniques, human error prediction techniques and methods used for analyzing human error following an incident or accident. Our particular interest is in methods for use post incident or accident. This is because of our multidisciplinary approach to safety-critical interactive systems design and the importance information from accident investigation reports have for our approach.

3.5.3 Error Analysis Approaches

Before presenting established error analysis approaches, we detail a number of potentially preventative measures to consider during the design of a safety-critical interactive system. Of the various measures, based on (Peters and Peters 2006) “countermeasures”, some can be considered more closely related to the human or to the system. However, since the discussion is centred on interactive systems, the boundary will not always be clear and there will be overlap between the two. The measures proposed by Peters and Peters have been represented as a graph which illustrates the proximity of the measure with respect to the human or the system and approximately where in the development process it is likely to be implemented.

Several technical terms are used in the table, which are further clarified below based on explanations provided by Peters and Peters (Peters and Peters 2006).

- Channalization: A means for reducing errors by providing guidance information to facilitate appropriate choice behaviour
- Sequestration: Providing sufficient distance between source of harm
- Guides and stops: Physical guides that serve to counter foreseeable problems (machinery)
- Automation: Removing the human from the loop
- Redundancy: Provide more than one means to accomplish a task
- Failsafe: Equipment should fail in a manner that creates no significant risk
- Enhancements: Providing humans with leverage for example, to assist with task
- Inactivity: When a machine/system is off, it should pose no harm to humans (i.e. electrical shock)
- Behaviour modification: Interventions to improve job satisfaction, interpersonal relationships, company cultures.
- Hardware and software barriers (see section 3.1.2.1)

- Hardware and software interlocks and lockouts (see section 3.1.2.2)

It can be seen from the table that the majority of countermeasures apply to systems and there is a lack of support for human errors and failures.

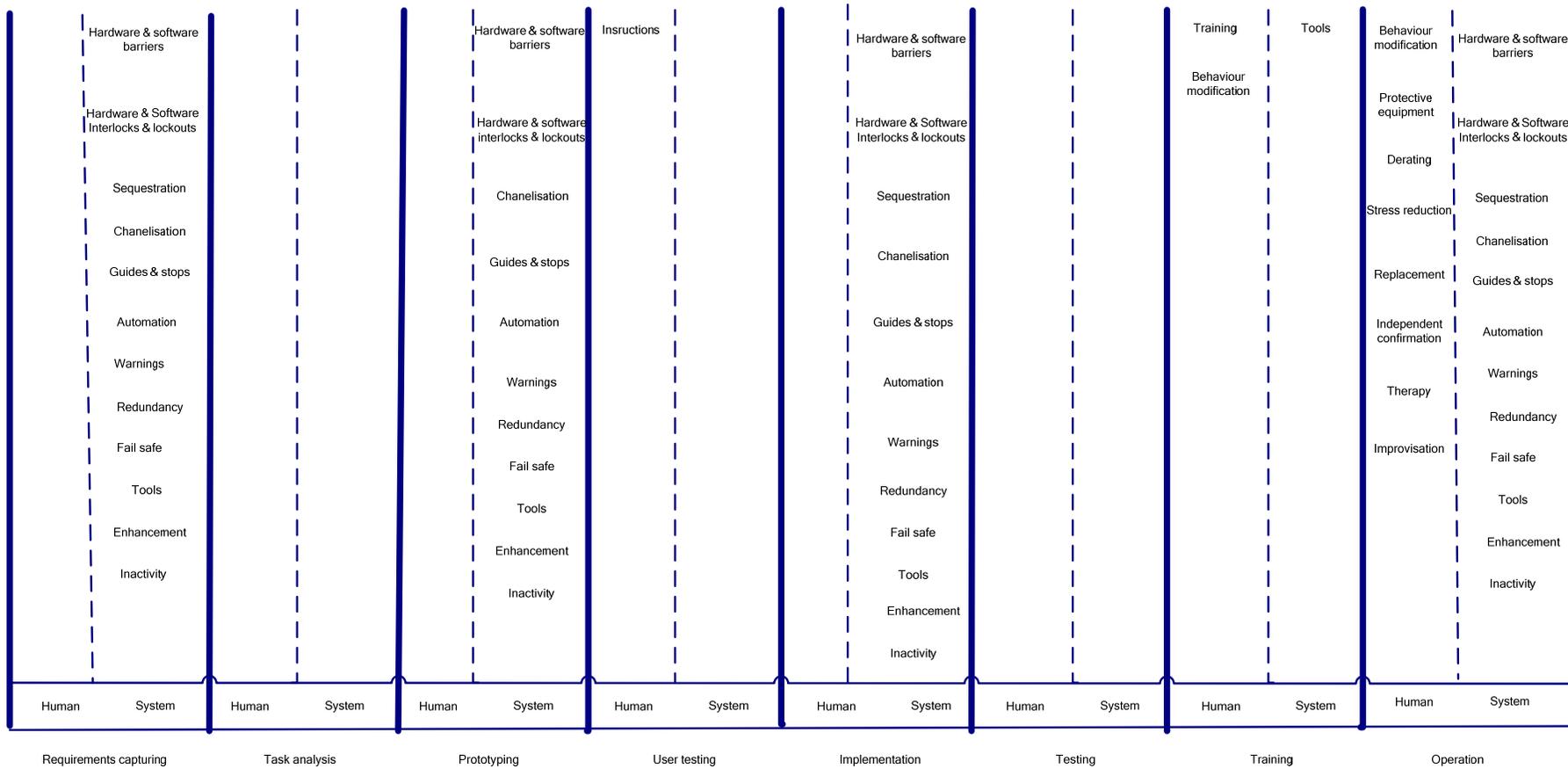


Figure 2-17. Peters & Peters' Countermeasures (Peters and Peters 2006) mapped to development process

3.5.3.1 Human Reliability Analysis

While previous sections presented safety analysis techniques relating to systems as a whole, the following two approaches deal explicitly with humans.

Human Reliability Analysis (HRA) is the method by which the probability of a system-required human action, task, or job will be completed successfully within the required time period and that no extraneous human actions detrimental to system performance will be performed. The results of HRAs are often used as inputs to the PRA process (Jeffcott and Johnson 2002a). HRA can be used to predict human performance issues and to identify human contributions to incidents before they occur and can be used to design safe and reliable systems (Basnyat et al. 2006). Referring once again to the NRC and nuclear industry, human reliability needs to go beyond being a diagnostic tool to become a prescriptive tool. The NRC are looking at new designs for control rooms and want plants designed with human reliability in mind, not simply verified after the design is completed. In (Basnyat et al. 2006), Ron Boring proposes a triptych (see Table 2-4) which illustrates the best achievable practices human reliability.

Table 2-4. The Human Reliability Design Triptych (from (Basnyat et al. 2006))

Design (including hardware and software engineering)	Testing (including equipment and human subject testing)	Modelling (including PRA, HRA and simulations)
Compliance with applicable standards and best practices documents	Controlled studies that avoid confounds or experimental artefacts	Compliance with applicable standards and best practices documents
Consideration of system usability and human factors	Use of maximally realistic and representative scenarios, users and/or condition	Use of established modelling techniques
Iterative design-test-redesign-retest cycle	Use of humans-in-the-loop testing	Validation of models to available operational data
Tractability of design decisions	Use of valid metrics such as statistically significant results for acceptance criteria	Completeness of modelling scenarios at the correct level of granularity
Verified reliability of design solutions	Documented test design hypothesis, manipulations, metrics and acceptance criteria	Realistic model end states

Though many further techniques for analysing human reliability exist we do not focus explicitly on such techniques in the framework of the PhD. However, presenting the above Triptych will help show later how our framework fits in with suggested design methods to deal with human reliability.

3.5.4 Accident Models

In this section we present two accident models and a classification of human factors issues that relate closely to the contribution of accidents. Such models and classifications are useful for a number of reasons. The models give a clear, pictorial impression of how our framework fits in with current research work. As will be seen, the accident models proposed take the form of layers, which have a knock on effect on each other if one fails, and furthermore, can contain holes in these layers. Our framework can be considered as a means of reducing these holes and reducing the likelihood that these holes will align thus allowing a hazard to reach a potential target.

3.5.4.1 Domino Model

Heinrich's Domino Model of Accident Causation (Heinrich et al. 1980) is a classic example of a sequential accident model which depicts an accident as a set of dominos that fall because of a unique initiating event. Heinrich proposed that accidents are caused either by an unsafe act, an unsafe condition or both. The falling dominos represent action failures which standing dominos representing normal events. The model can be considered deterministic since the outcome is deemed a necessary consequence of a specific event. By tracing the chain of causes and events back, it is assumed the investigator could find the original cause of the event and establish liability of the entire chain of events.

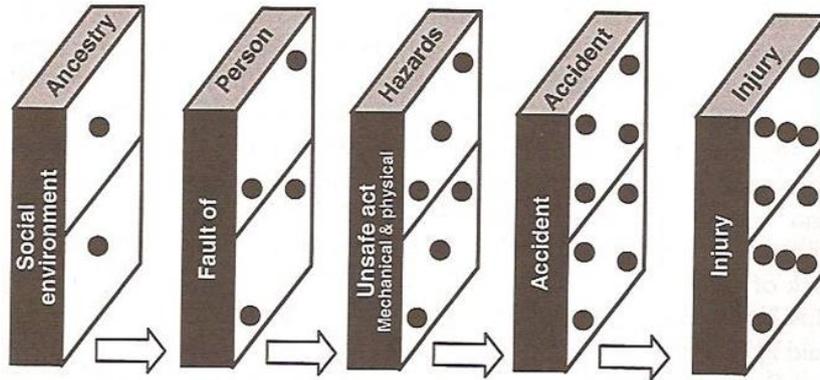


Figure 2-18. The Domino Theory (based on (Heinrich et al. 1980) taken from (Hollnagel 2004))

3.5.4.2 Swiss Cheese Model

An influential model, developed by James Reason is the “Swiss Cheese” model describing organisational accidents. This model is based on the idea that, rather than blaming the operators who directly control complex applications processes, there has been an increasing tendency to identify the underlying latent and distal causes of adverse events, such as poor safety management (Reason 1990). Reason’s theory shows that the precursor to an accident, the unsafe act (errors and violations (Reason 1990)), is the result of imperfections in the organisation/local environmental conditions within which a team or an individual operates. These organisational and environmental flaws are known as latent conditions which mean they may arise in advance of the accident and remain dormant in the system. The unsafe act results in an accident because of problems with the organisation’s defences. Again, the flaws relating to the organisation’s defences may be the result of organisational and local factors. They may also remain dormant in the system without detection. Figure 2-20 illustrates the Swiss Cheese model from the perspective of an ideal world where the layers of defence clearly prevent a danger or hazard from reaching a target, resulting in a potential loss. Figure 2-20 illustrates, that an accident occurs when the holes, or imperfections in the layers of organisational defences line up allowing the hazard to reach a target and cause an accident.

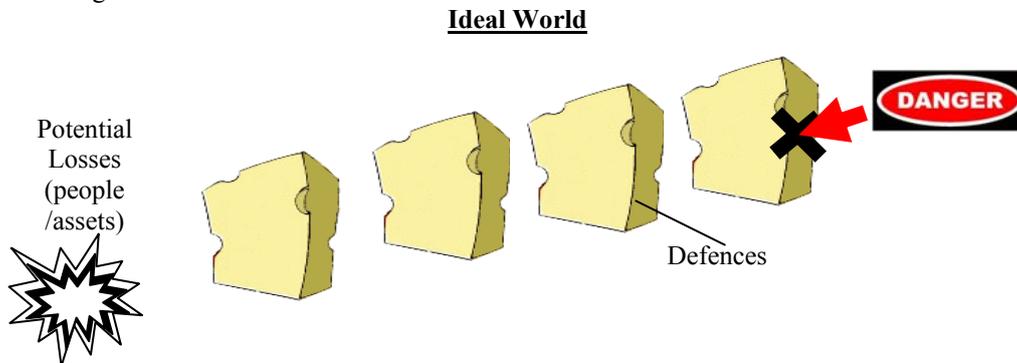


Figure 2-19. “Swiss Cheese” Model in an Ideal World

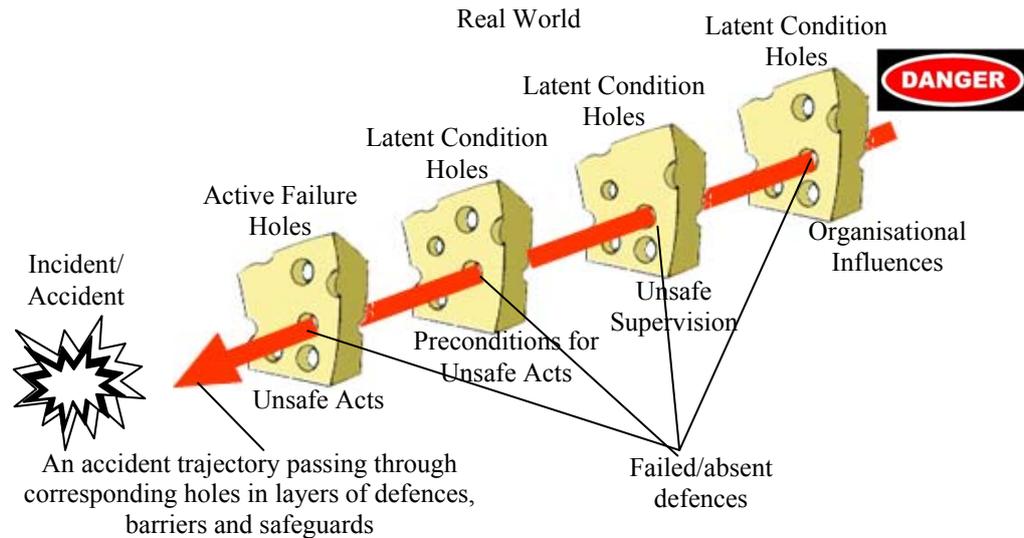


Figure 2-20. James Reason's "Swiss Cheese" Model

The model has four levels of human failure each influencing the next. Working backwards in time from the accident, level 1 is known as unsafe acts of operators. This concept originates from the nuclear power plant industry. This is where many accident investigations have focused their efforts, though we described in section 3.1.1 of this chapter that it is a complex combination of contributory factors (human and technical) that together cause systems (humans and machines) to fail and not just the unsafe act of an operator. The other three levels of human failures related to latent failures. Level 2, is known as preconditions for unsafe acts. It involves the condition of the operators (such as mental fatigue) and how their condition affects their performance. Level 3 is known as unsafe supervision which is again a form of latent failure involving for example breakdown of good communications which can be traced back to instances of unsafe supervision. The Swiss Cheese model does not terminate at a supervisor level, the model shows that the organisation itself can impact performance at all levels. This is represented in level 4, organisational influences.

3.5.4.3 Human Factors Analysis and Classification System (w.r.t Swiss Cheese)

A problem with Reason's Swiss Cheese model is that there are few details on how to apply it to the real world, and that it does not define what the holes actually are. The Human Factors Analysis and Classification System (HFACS) attempts to describe the holes in the cheese by bridging the gap between theory and practice, particularly for the aviation industry. HFACS is based on the Taxonomy of Unsafe Operations framework which was based on naval aviation accidents obtained from the U.S Naval Safety Centre (Shappell and Wiegmann D.A 1997). The naval taxonomy was refined using input and data from other military and civilian organisations such as the US National Transportation Safety Board (NTSB) and the US Federal Aviation Administration (FAA) resulting in the HFACS. The HFACS is based entirely on Reason's four levels of failures but relates strongly to the aviation industry. Though Shappell and Wiegmann argue that unsafe acts (errors and violations) do not provide the level of granularity required of most accident investigation and have expanded on these two categories of unsafe acts to include three basic error types and two forms of violations (see Figure 2-21).

We present this classification since classifications of human behaviour play a strong role in this PhD. Though we will not go beyond individual human 'errors', this classification is interesting as it provides a bigger overview of problems relating to management and organisational issues.

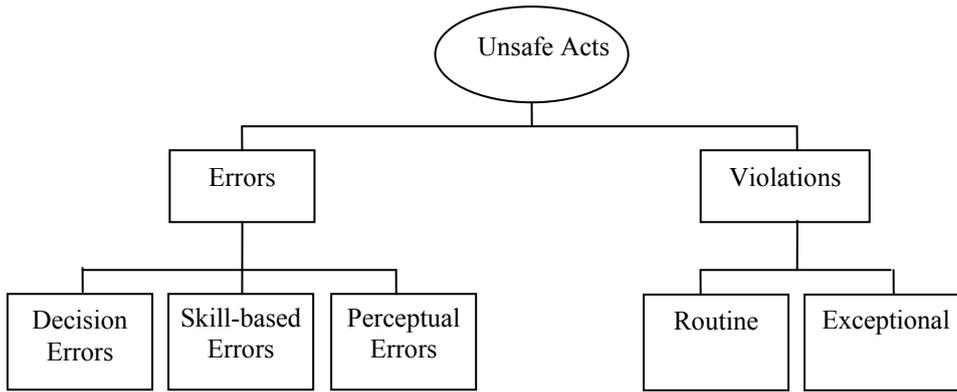


Figure 2-21. Categories of unsafe acts committed by aircrews (from (Shappell and Wiegmann D.A 2000))

The further three levels of defence relating to latent failures have also be further categorised by Shappell and Wiegmann. See Figure 2-21, Figure 2-22 and Figure 2-23 for details.

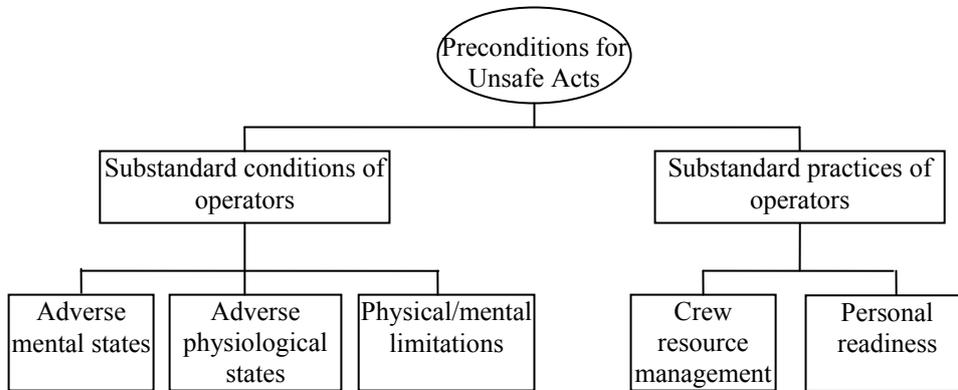


Figure 2-22. Categories of preconditions of unsafe acts (from (Shappell and Wiegmann D.A 2000))

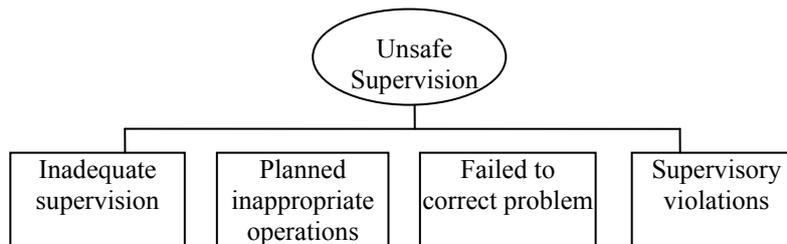


Figure 2-23. Categories of unsafe supervision (from (Shappell and Wiegmann D.A 2000))

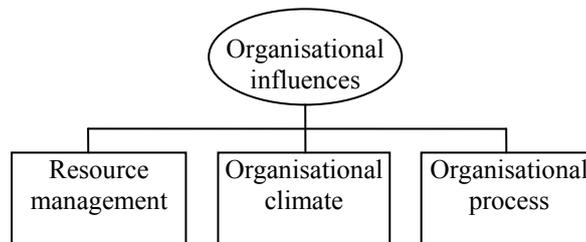


Figure 2-24. Organisational factors influencing accidents (from (Shappell and Wiegmann D.A 2000))

We have taken the HFACS classification and plotted them on Reason’s Swiss Cheese model in order to clearly show how the two theories relate.

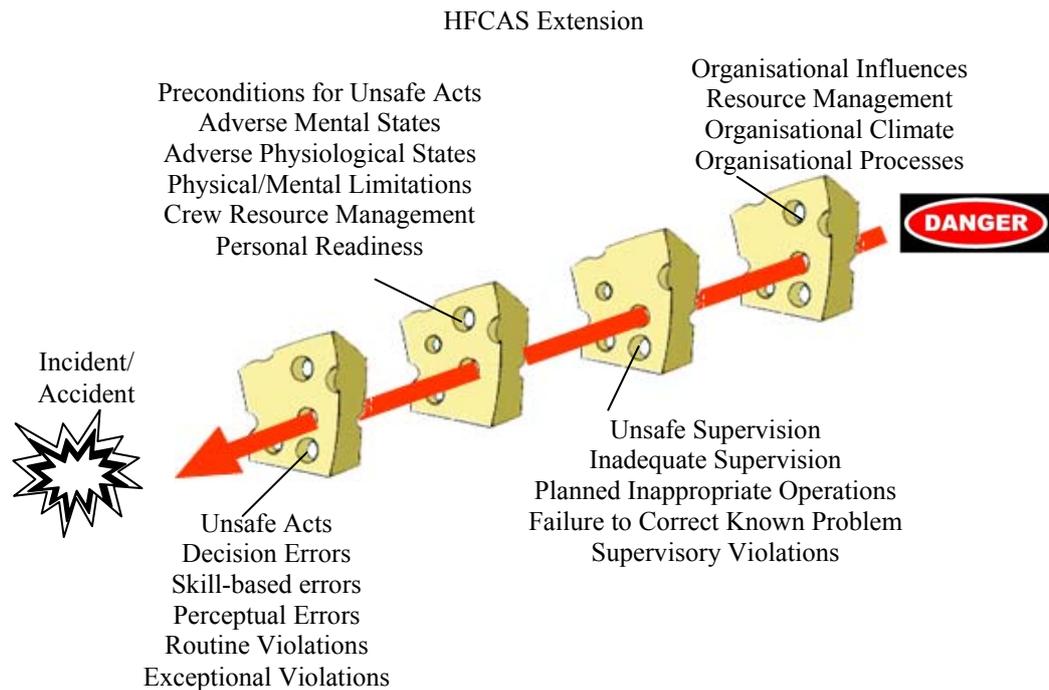


Figure 2-25. HFCAS Extension to Reason's Swiss Cheese Model

The problem with the Domino and Swiss Cheese models is that they do not account for gradual loss in safety that may also lead to safety (Hollnagel and Woods 2006a). The “drift to danger” model (presented in (Hale and Heijer 2006)) introduces the a dynamic aspect, though Hollnagel and Woods make it very clear that one should not take the term drift literally nor refer to it as a model for the following reasons.

- The distance of a drift cannot be measured, ‘Drift’ then only refers to how a series individual actions or decisions have larger, combined and longer term impacts on system properties that are missed or underappreciated (Hollnagel and Woods 2006a).
- Some parts of an organisation may be safe while others may be unsafe. In other words, parts of the organisation may ‘drift’ in different directions(Hollnagel and Woods 2006a).
- There are no external forces that... push an organisation in some direction ... to steer it clear of danger. What happens is rather that choices and decisions made during daily work may have long-term consequences that are not considered at the time directions(Hollnagel and Woods 2006a).

In this section we have seen two established accident models and a classification of human factors issues relating to unsafe operations. The models provide an overview and help place our framework within existing ideas. Our approach aims to reduce the holes in the Swiss Cheese model by taking into account additional information in the design process relating to safety, reliability and human error.

3.6 Tasks and user errors

This last subsection of the “Achieving Interactive Systems Safety by Resilience Engineering” section makes a link between Chapter 1 which introduces task modelling techniques as a means of User Centred Design, with theories presented in this chapter on human error. As mentioned previously, this is a multidisciplinary thesis, touching aspects of multiple domains, though we show here how closely these different factors are related. The PhD is heavily based on task modelling and human error as well as system modelling. Here we make explicit the work that has been carried out in the field of tasks and user errors, and incorporating the two as a means of identifying potential human errors. We show how the contribution to the PhD (part 2, Chapter 4) can extend the current state of the art and connect to it.

Hazard analysis and error identification are systematic processes applied to an emerging or completed design to gather information about the design’s dependability. There are a number of such methods and they have features in common. For example, in all cases some form of design representation is used as a starting point. This representation could be a process architecture of the system or a procedural description of how the system behaves for example. The second common feature is some kind of procedure for asking structured questions

systematically about the designed system. These questions are used to identify significant parts of the system where dependability is problematic. None of the methods involves quantification. They involve a variety of possible techniques including for example HAZOP (see Chapter 2 Section 3.1.2) and Technique for Human Error Assessment (THEA) as examples.

When modelling user behaviour, an error-free perspective is usually employed. It is normally during the testing phase of the system development cycle that errors are realised and taken into account. Task modelling as yet, does not allow for the description, representation and analysis of any unexpected eventualities that may occur including human error. Since the task model influences the system design it is important to understand how to manage and overcome possible errors.

TAFEI

In their paper, Baber and Stanton (Baber and Stanton 2004) propose that a system image implies a set of routines that a person may use and that the selected routine will be based on the user's goal and previous experiences. They suggest a need to represent interaction between user and product to consider possible mental model mismatches between a given system image and user representation during initial design activity. The proposed technique, Task Analysis for Error Identification (TAFEI) is based on the assumption that interaction is goal-oriented and passes through a series of states. The TAFEI approach consists of three stages, a Hierarchical Task Analysis (although any technique is acceptable), construction of a State Space Diagram (SSD) mapped with the HTA plans, finally construction of a transition matrix to display state transitions during device use. This work tries to address the same goal as ours. However, there are three main differences. Our work, exploiting a classification of user error provides a systematic way of dealing with possible user errors. Our proposal to define and exploit error patterns of user errors provides a way of coping in an efficient and reliable way with the complexity of the task models. The authors consider as erroneous only the paths in user tasks provided by the system which do not support the achievement of the user's goal. We consider user error in a broader sense including errors such as mistakes and slips.

Task Deviations

Paternò and Santoro (Paterno and Santoro 2002) also suggest that a goal is a desired modification of the state of an application. Their work describes how task models can be used in an inspection based usability evaluation for interactive safety-critical applications. A key aim to their method is to provide designers with help in order to systematically analyse what happens if there are deviations in task performance with respect to what was originally planned during the system design. Building upon the HAZOP family of techniques a set of predefined classes of deviations are identified by guidewords such as "none", "other than" and "ill-timed". The method incorporates three stages; 1) Development of the task model of the application considered, 2) Analysis of deviations related to the basic tasks 3) Analysis of deviations in high-level tasks.

The analysis, which is recommended be carried out by interdisciplinary groups, follows a bottom-up approach considering basic and then high-level tasks. Documentation of the analysis takes the form of tables. Paternò and Santoro (Paterno and Santoro 2002) mention that the interpretation of every guideword for every task of the prototype addresses completeness issues however has the drawback of taking time and effort. This issue of cost/benefit analysis will be dealt with extensively in the conclusion section of Part 2 when the proposed framework of the PhD has been presented.

This work tries to address the same goal as ours. However, there are two main differences. Our proposal to define and exploit task patterns for user errors provides a way of coping in an efficient and reliable way with the complexity of the task models. The authors consider error analysis based only on a set of high-level guidewords (such as "other-than") and thus leaving detailed analysis to the analyst's discretion. Our proposal is much more concrete and grounded on previous work in the field of user error identification and classification. For instance we have more than 20 types of errors that would fit within the "other-than" guideword (for instance "branching error", "environmental capture", "order error" ...).

THEA

The Technique for Human Error Assessment or THEA (Pocock et al. 2001) is aimed at helping designers of interactive systems anticipate human errors resulting in interaction failures. As we suggest, this is also a technique intended for use during the initial phases of the development lifecycle. With its foundations laying in human-reliability analysis (HRA) (Peters and Peters 2006) it aims to establish requirements for "error resilient" system design. In their paper, it is noted that errors can be regarded as failures in cognitive processing (Pocock et al. 2001). The process of analysing a system's vulnerability to human error is performed by posing provided questions about a given scenario, identifying possible causal factors of potential problems identified and finally

identifying consequences and their impact on task, work, user, system, etc. The results are recorded in tables for further analysis.

Like human HAZOP (see Chapter 2 Section 3.1.2), THEA (Pocock et al. 2001) also begins with an explicit description of the work that is under analysis in this case scenarios because they provide a richer cultural context for the analysis of the system

The THEA approach shares a common perspective with our work, that is, we aim to assist designers to produce usable error tolerant interactive systems based on using systematic techniques. However, our work differs in that we intend the task patterns to be applicable to more than one system design, possibly of various domains. This means that once a sound solution has been modelled, there will be less repetition of work.

Deriving Human-Error Tolerance Requirements from Tasks

In (Wright et al. 1994) the authors present a form of analysis based on the concept of trace in order to reason about potential human errors based on a task specifications. The aim of the approach is to define error tolerant requirements. The tasks are specified from both a user and system perspective. The software engineering notation, ‘*Communicating Sequential Processes*’ (CSP) (Hoare 1985) is used to bridge the theory between human error and the practice of design and implementation. The approach takes a task specification and then applies error classes to determine the consequence of these potential errors. The approach has four main phases as described below:

1. Identification of potential human errors: Based on detailed operator, task and error data a system analysis is performed resulting in classifications of three classes of errors, omissions, commissions and reversals.
2. Selecting significant errors: In this phase, the relevant identified errors are extracted. That is, those that lead directly to an undesirable event based on the system analysis, the likelihood of the failure and of other conditions necessary for the error.
3. Detailed analysis of significant errors: Qualitative information is gathered (complexity, time available, time necessary etc) for each action to derive an estimate of its likelihood
4. Integration into a system model: Using cause consequence trees, human error analysis is integrated with the system model to study the dependence between human errors and system failures.

Our approach differs from this as we try not to combine the task specification and the system specification in the same model. We propose to use dedicated notations for each model allowing more accurate and concise specifications. Since the errors are more human related than system related (is we assume that the human actions cause the problems rather than the system events) we apply the human error analysis to sub tasks of the task model. Our aim for analyzing human errors in task models is to inform the system model making it potentially more error tolerant by taking into account this additional information early in the development process.

4 Certification Phase

This section briefly describes the certification phase of the development process of a safety-critical system. Though we do not directly contribute to this area, we make use of standards which have a strong implication on such certification phases. The case study section of this thesis makes use of the ARINC 661 standard.

The certification phase of the development of a safety-critical system is what makes the development process unique from that of a ‘normal’ (non safety-critical) system. In many highly critical projects the final state is to convince some external regulating body that the system is safe, and thereby to achieve certification (Storey 1996). Certification is the process of issuing a certificate to indicate conformance with a standard, a set of guidelines or some similar document (Storey 1996). The aims of certification are to improve safety of critical systems, to increase the awareness of the implications of system performance on safety, to enforce minimum standards of design and manufacture within the relevant industry and to encourage a structure of professional responsibility (Storey 1996).

In part 2 of the ARINC 661, a standard which aims to normalize the definition of a Cockpit Display System (CDS), and the communication between the CDS and User Applications (UA), it is stated that it is widely recognised that software qualification and system certification costs dwarf all other aspects of developing, installing and updating a Cockpit Display System (CDS) (ARINC 661-2 2005).

For a safety-critical system, safety aspects must be considered and the development processes must be able to address them in a balanced and consistent way. While these concerns are typical of the development process of any type of software systems, safety critical ones put a special emphasis on the certification phase. Indeed, developers of such systems are responsible for demonstrating that the system is ‘acceptably safe’ before certification authorities grant regulatory approval.

Storey (Storey 1996) indicates that there are three typical aspects to the certification of safety-critical systems:

- A demonstration that all important hazards have been identified and dealt with, and that the integrity of the system is appropriate for the application.
- Evidence of compliance with some particular standard
- A rigorous argument to support the claim that the system is sufficiently safe and will remain so throughout its life

5 Conclusion

This chapter has introduced the concept of resilience for interactive systems by detailing individual methods and approaches that could be used together to assist in producing more resilient safety-critical interactive systems. Methods such as incident and accident investigation techniques, safety analysis techniques, methods for anticipating hardware and software failures and the incorporation and consideration of human factors and human error in the design process have been described. Some of the techniques discussed are based on the assumption that a problem, such as an incident or accident has previously been identified and that improvements can be made to future designs. However, even a new system, for which there is no previous accident data will in large be similar to an existing system and will therefore benefit from past experiences, hence the section on learning from experience. We believe past experiences are a particularly pertinent source of data to feed into the design process.

We have also focused on our area of interest which is formal specification techniques; in particular the Petri net formalism which has been used in the past to represent incident and accident investigation data for safety analysis. However, none of the literature aims at utilising this powerful method of information representation to inform the design of a system. Our approach differs from the described literature, in that we are not attempting to model incident and accident reports using Petri nets. One of our main aims is to show in our approach how the use of Petri nets and data from incident and accident investigation reports can be used to inform the model-based design of a system in order to provide a safer safety-critical interactive system specification. We firmly believe that the many individual experts needed for the design of a safety-critical interactive system, including accident analysts, risk assessment professionals, system requirements developers, system specification experts etc, should continue to use techniques and methods they are familiar with and have experience in. What is vital is that each model applied within the development process should contain enough, but not too much relevant information. That is to say, a system model should not contain information relating to user performance shaping factors, and a task model should not contain information relating to system behaviour.

Part 1 Conclusion

We have shown in Chapters 1 and 2 that there is a considerable amount of research in the task modelling domain, system modelling domain and in the analysis of human error. However, little research has been dedicated to modelling these aspects together for error-tolerant systems design with emphasis on safety critical interactive systems.

We conclude part 1, describing relevant state of the art with respect to the interests and framework of this thesis, by summarising in Table 2.5 techniques for considering human error in either task modelling or system modelling that have been identified in the literature review.

The table highlights a gap in the domain for explicit handling of all types of human errors that have been identified in the literature review and listed in the table. For example, Palanque and Bastide (Palanque and Bastide 1997) have considered HAZOP type errors in system modelling by means of Petri nets but no other types of human errors. From a task modelling perspective, Paterno and Santoro (Paterno and Santoro 2002) considered HAZOP guidewords but no cognitive errors.

There is currently no approach to take into account all of these elements in either task modelling or system modelling. Designing a system only taking into account part of such information, will necessarily produce either an unsafe, unusable or unreliable system or a combination of all these factors. Our framework accounts for these issues for both task modelling and system modelling phases by proposing methods, tools, techniques and processes for accounting for multiple types of data aiming to design more usable, reliable and safe interactive systems.

Part 2 will present a generic integrated modelling framework that will try to piece together contributions in these various domains in order to go beyond current practice.

Table 2-5. Summary and comparison table of state of the art

Error Classification Name	Based on	THEA see Chapter 2 section 3.6) (Pocock et al. 2001)	TAFEI see Chapter 2 s section 3.6) (Baber and Stanton 2004)	Petri-nets see Chapter 1 section 4.1.1.3) (Palanque and Bastide 1997)	User deviations in task performance see Chapter 2 section 3.6) (Paterno and Santoro 2002)
		System modelling Semi-Formal	Task modelling Formal	System modelling Formal	Task modelling Semi-Formal
Categories of slips and lapses (see Chapter 2 section 3.5.1.4) (Norman 1990)		x	x	x	x
Mode confusion (see Chapter 2 section 3.2.2.3) (Palmer 1995)		x	x	x	x
GEMs (see Chapter 2 section 3.5.1.2) {Reason 1990 #430}		x	x	x	x
Failure modes {Reason 1990 #430}		x	x	x	x
Common mechanisms {Reason 1990 #430}		x	x	x	x
Phenotypes & Genotypes (see Chapter 2 section 3.5.1.3) (Hollnagel 1991)		x	x	x	x
Skill-based Human Errors (see Chapter 2 section 3.5.1.1)(Rasmussen 1983)		x	x	x	x
HAZOP Causes of deviations (see Chapter 2 section 3.2.2.2) (MOD 1996)		√	x	√	√

X: Not taken into account in the approach

√: Taken into account in the approach

PART 2 – THE APPROACH

Chapter 3

Design Process for Safety-Critical Interactive Systems

*“Intellectuals solve problems, geniuses prevent them.”
(Albert Einstein, 1879 - 1955)*

Chapter Summary

We have seen in the state of the art current model-based design methods for designing systems. Such methods are limited in that they do not tackle interactive systems aspects, nor explicitly integrate human factors issues, or erroneous technical and human-related behaviour, and most (except MEFISTO) are not dedicated to safety-critical interactive systems. For this reason, we present in this chapter the generic integrated modelling framework, the core on which the thesis is founded which tackles the limitations of other model-based design approaches for safety-critical interactive systems.

1 Introduction

This chapter presents issues relating to the complexity of designing a safety-critical interactive system and provides justification for the need of an integrated modelling framework which provides modelling consistency. The integrated modelling framework will be presented. The principle phases of which will be discussed in subsequent chapters.

In order to demonstrate our ideas, this chapter and further chapters take the example of an Automated Teller Machine (ATM) for explanatory purposes. ATMs have often been used to demonstrate task analysis since they are widely used systems demonstrating clear human-computer interaction. We are using the ATM as an example because we are focusing on repetitive, highly structured, situated based systems involving less decision making, however errors can still occur.

Human-Computer Interaction and related disciplines have argued, since the early days, that interactive systems design requires the embedding of knowledge, practices and experience from various sources. For instance, user centred design (Norman 1988) advocates the involvement of human factors specialists, computer scientists, psychologist, designers ... in order to design useful and usable systems. While designing interactive software, the use of formal specification techniques is of great help as it provides non-ambiguous, complete and concise models.

In the state of the art chapters, we argued that it is important to consider all stakeholders during the design process. The consideration for all stakeholders leads systems designers and analysts to look at the same system (the one to be designed) from multiple perspectives. Such perspectives come from, but are not limited to domains such as human factors, product development, training, product management, marketing, the customers, design support, system engineers and interface designers. A number of these domains will be discussed more in detail hereafter and more precisely describing the roles they have in supporting interactive safety-critical systems design.

Our work relates closely to the ideas of Burns's (Burns 2000) Framework for the Formalisation for Accident Reports, introduced in the state of the art which was described as very high level and not systematic. We will show how based on a very similar idea, we can provide a far more systematic approach to identifying potential human errors, by also using accident reports, human error classifications and task analysis methods.

Furthermore, in relation to the Human Reliability Design Triptych presented in Chapter 2, we show here, how our approach fits in with and relates to existing design suggestions. The approach presented in this thesis can be said to focus on design and modelling issues. The testing issues described in the triptych are not considered, though model verification/validation is considered. Table 3-1 and Table 3-2 describe how the triptych recommendations for human reliability design are taken into account in our approach. Case study 1 refers to the

mining accident presented later in Chapter 8 and case study 2 refers to the multipurpose interactive application described later in Chapter 9.

Table 3-1. Generic Integrated Modelling Framework Approach With Respect to Design Column of The Human Reliability Design Triptych

Design	Considered in the proposed Integrated Modelling Framework
Compliance with applicable standards and best practices documents	Case study 1: compliance with manufacturer operating guidelines. Case study 2: compliance with the ARINC 661 standard for flight deck display interface.
Consideration of system usability and human factors	Case Study 2: usability heuristics imposed. Case studies 1 and 2: human "error" analysis studied.
Iterative design-test-redesign-retest cycle	The Integrated Modelling Framework is a cyclic iterative process.
Tractability of design decisions	Not considered in the Integrated Modelling Framework
Verified reliability of design solutions	Case study 1: Model testing, validation/verification with respect to data in accident investigation report. Case study 2: Model testing, validation/verification with respect to imposed software barriers.

Table 3-2. Generic Integrated Modelling Framework Approach With Respect to Modelling Column of The Human Reliability Design Triptych

Modelling	Considered in the proposed Integrated Modelling Framework
Compliance with applicable standards and best practices documents	Case study 1: compliance with manufacturer operating guidelines. Case study 2: compliance with the ARINC 661 standard for flight deck display interface.
Use of established modeling techniques	Case studies 1 and 2: Established task modeling technique and established system modeling technique applied
Validation of models to available operational data	Case study 2: The models can be validated by running the rendered user interface. Though for case study 1, it is not possible, sine have not constructed a full mining plant.
Completeness of modeling scenarios at the correct level of granularity	Case studies 1 and 2 have been modeled at a level of abstraction necessary to represent the information we are interested thus an appropriate level of granularity. If further information was to be considered, i.e. environmental factors, operator stress etc, the models would not be of a sufficient level of granularity.
Realistic model end states	Case studies 1 and 2 have been modeled using Petri nets which means that every possible state (with respect to the abstract model) is detailed. This means every state is an end state.

Due to the large number of domains involved in the design of a safety critical interactive system, it is highly unlikely that the data gathered, analysed and documented will be represented in the same way. For example, it is unlikely that the system engineers will take into account all information provided by human factors analysts (for instance about work practice and users). This is not only because of time constraints and the amount of data involved, but also and mainly, because the kind of notation that they are used to employing cannot record that information efficiently. This can have serious effects on the reliability, efficiency and error-tolerance of a system. For example, if a task is represented in a task model by a human factors expert and if that information is not represented (in one way or another) in the system model by a systems engineer there is no means to ensure and check that the system will support this task.

It is clear that there is a need for formalising not only the process of gathering this mass of data, but also for refining and modelling it when necessary in order to provide valuable input to the system design. The following

sections discuss the need to provide means for gathering, formalising and refining multi type data of multi sources.

1.1 Gathering Information

The phase of gathering information for the design of a new system is crucial for the success of the end product. If performed incompletely, inaccurately or indeed ignored, gaps are left in understanding the scope, concept and function of the new system. The process of experts gathering data from various domains for input into the system design has been studied as part of the Mefisto Method (see section 6.1.3).

Gathering information is not a goal per se. The result of this activity must be used to feed other phases in the design process. This feeding cannot be left informal nor at the discretion of those responsible for these other phases. In addition, not all types of information are closely enough related to build useful bridges between them. On the other hand, some sources of information are so close that, not merging and cross validating them would certainly result in poorly designed and inconsistent systems.

For instance, scenarios and task models (as discussed in the state of the art) both convey information about user activities. It is thus possible to check that scenarios and task models (for the same activity) convey not only the same information but also the same sequencing of operations.

Similarly scenarios and system models (also discussed in the state of the art) both deal with the same operational system and thus must contain compatible and coherent information and this must be checked at all stages of the development process. These examples have not been chosen randomly. Indeed, scenarios are the perfect candidate as the corner stone of the consistency process.

1.2 Formalising Information

We have seen in the state of the art, that there is a significant amount of literature on the design process for interactive systems design the more referred to being the ones including prototyping activities and evaluations (Dix et al. 1998)p206-207 and (1993)p102. However little research exists on formalising the process of 1) documenting the information such that experts of other domains can understand and reuse information for their analysis, 2) refining the information to share only what is necessary and 3) embedding data from one domain to another, all for input into the system design

2 Multi-Type Data

The data obtained and analysed by various domain experts can be considered as multi-type data. We have distinguished between two main types of data, pre-design data and post-design data. That is, data that is available before a system has been designed, and data that is available after a system is designed. This distinction and its impact on systems design is explained in more detail in the following sections.

2.1 Pre-design data

Data can be obtained throughout the design process before the system has been developed. Of course, much of this data can be made available and used for evaluation purposes, once a system has been designed. However; we have labelled it pre-design data because the techniques can be applied without the need of the current system. Within this category of pre-design data, data can be further classified according to the properties of the data obtained. That is, formal or informal, complete or incomplete for example. Figure 3-1 illustrates on a three-dimensional cube, four examples of techniques that can be applied to obtain data before the system has been designed. By formal and informal we mean whether there is only one interpretation of the models or not. Complete and incomplete refer to the fact that the model contains either a sub set of the relevant information or deals exhaustively with it. Finally, high and low-level data refer to the level of abstraction at which the information is dealt with.

To illustrate the complexities surrounding multi-type data, we have provided an example of seven techniques positioned in the Multi-Type Data Cube. Some of the examples are presented in more detail later in this chapter. This type of 3D presentation is used because of the overlapping properties of the techniques. For example, a Petri-net is considered in this thesis as formal, complete and precise enough so it is possible to use them to represent any basic or complex data.

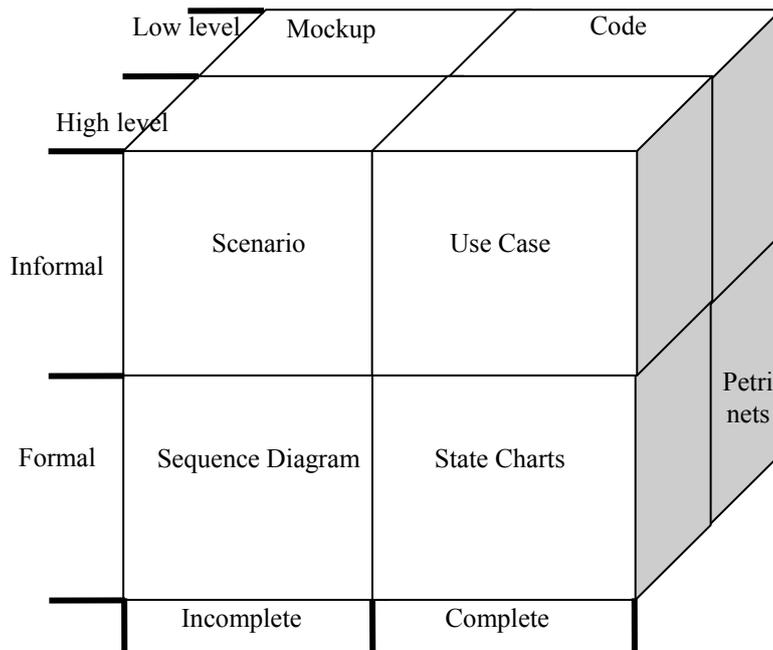


Figure 3-1. Multi-Type Data Cube

Figure 3-2 provides an example of a formal low level and complete data modelling using a simple Petri-net. The model describes the behaviour of a one-attempt PIN entry system. If the PIN is correct, a token is deposited in place Success, while if it is incorrect, a token is deposited in place CardLocked.

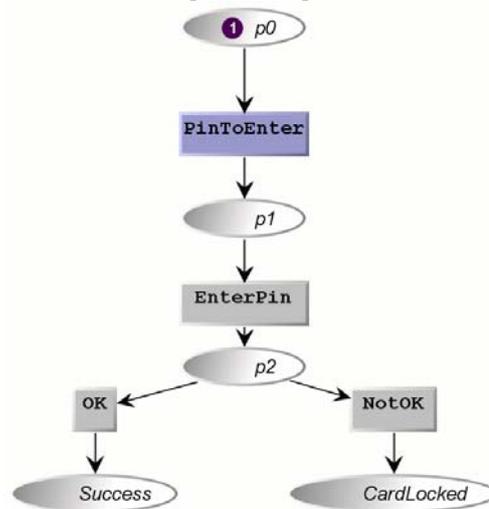


Figure 3-2. Formal low level and complete data modelling using Petri-nets

An example of incomplete, informal and low level data, as shown on the 3D Multi-Type Data Cube is a scenario. In safety-critical interactive systems design, scenarios can be used to elucidate the particular chain of events that lead to an accident but can also be used to identify alternate failure scenarios that might cause future adverse events. In the case of an ATM machine, an example may be,

“Mr X enters his card in the cash machine; a task he does not frequently perform, and fails to enter the correct PIN on 3 attempts thus loses his card”

One of the problems associated with ensuring consistency, reliability, efficiency and error-tolerance in the design of an interactive safety-critical system, lies in the probable limited use of fruitful information. Scenarios can be used in line with many techniques, such as task modelling, a priori and a posteriori i.e. for design or evaluation activities. A careful identification of meaningful scenarios allows designers to obtain a description of most of the activities that should be considered in the task model(Paterno F and Mancini 1999).

An example of incomplete, formal and high level data using an ATM as a case study is provided in Figure 3-3. This event-based sequence diagram can be used to map out what happened in the lead-up to an adverse event as well as to represent sequences of events that could potentially occur in a new design.

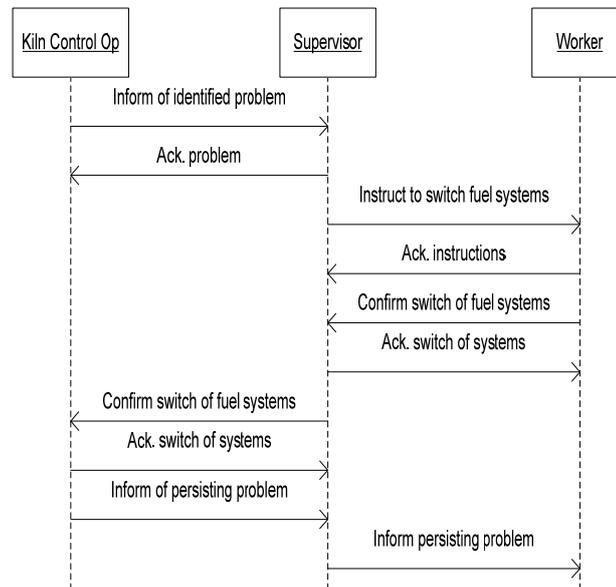


Figure 3-3. High-level data, communication sequence diagram from UML

2.2 Post-design data

The second distinction of data we have made is post-design data. By this, we mean data that can only be obtained once the system in mind has been designed. Examples of such are usability analysis, incident and accident reports or the use of metrics for risk analysis (Fenton and Neil 1999)

The design of a safety-critical interactive system must be grounded on concrete data, of which may be of multiple source and of multiple type. However, an additional way to compliment and enhance a system's safety is to take into account as much information from previous real life cases. One such type of data is an incident or accident report. To date, input to a safety-critical interactive system design from an incident or accident report has not been considered in a systematic way. We believe these reports can be extremely fruitful to the design of safer safety critical systems. In most cases, these reports are used by assigned experts to analyse why an incident or accident occurred and what could be changed to prevent future similar scenarios from occurring. In contrast, we suggest using the reports to improve future design. To be more concrete, we have implemented this approach on one of the two case studies presented in this thesis, the fatal mining accident. The reports allow us to achieve two things, 1) obtain and 2) deduce important information that could be embedded into future systems as that which was considered.

3 Multi Source Data

The data gathered and analysed for input into a safety-critical interactive system design is collected by multiple specialists of a wide-array of domains. This is due to the nature of safety-critical but also to the variety of information that has to be gathered and the fact that this information stems from multiple domains of expertise. This combination of diverse specialists and diverse domains adds to the complexity of design of a safety-critical system. This section describes several such specialists and domains and the input they have on the design.

Human factors is a discipline which aims to put human needs and capabilities at the focus of designing technological systems to ensure that humans and technology work in complete harmony, with the equipment and tasks aligned to human characteristics (<http://www.ergonomics.org.uk/ergonomics/definition.htm>). Examples of human factors specialists are production engineers, health and safety- practitioners and interface designers. These are just a number of experts in the human factors field who all bring advantages to the design of the system. However, the complexity increases when considering the background of these experts and the ways in which their analyses will vary according to their backgrounds.

Occupational Health and Safety (H&S) practitioners are trained in the recognition, evaluation and control of hazards which place people's safety and health at risk in both occupational and community environments. Techniques employed by H&S practitioners include risk assessments, postural analysis, legal and organisational factors, work equipment. As with most occupations, health and safety practitioners also have wide ranging educational backgrounds such as psychology, anthropometry or physiology. This results in multiple perspectives and methods of working on the same system.

An Interface Designer is responsible for the presentation of the interface part of an application. Although the term is often associated to computing, the interactive part of a system can include controls and displays in many domains such as military aircraft, vehicles, audio equipment and so on. The educational background of an interface designer can be varied, computer science, graphics design or again psychology. It is probable that a psychologist and a computer scientist will base their interface designs on different principles. Stereotypically, for example, a psychologist may wish to ensure correct colours are used, whereas a computer scientist will want to employ the latest programming techniques with a flashy interface. Both perspectives can be advantageous to the overall design.

Systems engineering is an interdisciplinary process referring to the definition, analysis and modelling of complex interactions among many components that comprise a natural system (such as an ecosystem and human settlement) or artificial system (such as a spacecraft or intelligent robot), and the design and implementation of the system with proper and effective use of available resources (<http://sydewww.uwaterloo.ca/SystemsDepartment/WhatIsSystems/whatissystems.html>). In the fatal mining accident case study that we will present, mechanical and automation engineers were involved. However, other types of engineers include hardware, software and systems engineers. The combination of these engineers assists in the system development process. Further classifications of systems engineering can include hardware and software engineers, mechanical and automation engineers. A mechanical engineer can have a variety of responsibilities such as, the design and improvement of machines and mechanisms, organisation and maintenance of computer controls for production processes or even selection and installation of equipment for indoor environment control. Automation engineers design, build and test various pieces of automated machinery. This can include electrical wiring, tooling, software debugging etc. One of the main fields of an automation engineer is to design automation systems from a collection of single components of different distributors.

Furthermore, incident and accident analysts are interested in understanding system 'failures' and human 'error' using accident analysis techniques and incident reporting techniques. [<http://www.dcs.gla.ac.uk/research/gaag>]. Such analysts have varying educational backgrounds in computer science for example.

4 Ultimate Goals

The above mentioned issues increase complexity in the design of safety-critical interactive systems due to the necessary ultimate goals of embedding reliability, usability, efficiency and error tolerance with the end product. Without such ultimate goals the development process would be far less cumbersome. This is a very important aspect of the work presented in this thesis as it points out the issues that are specific to the type of applications we are considering, i.e. large hardware based interactive systems as well as embedded software applications, and thus less relevant to others more commonly considered.

4.1 Consistency

Consistency is a means to achieve reliability, efficiency, usability and error-tolerance of a system. This can only be achieved by means of systematic storage of gathered information into models and the development of techniques for cross models consistency checking.

4.1.1 Model Consistency

One of the problems associated with safety-critical interactive systems design is the lack of consistency between multiple viewpoints and therefore multiple design models, of the same world. We believe there should be consistency between these design models to reduce the likelihood of incidents or accidents in the safety-critical systems domain.

Some work on model-based approaches has tried to address these issues but there is still a lot to do before design methods actually provide a framework to support this critical activity. Indeed, it is still not commonly agreed that there should be a framework for multiple models as some current research argues that models of one kind could

be generated from models of the other kind. For instance (Paterno et al. 1999) proposes to generate user interfaces from task models while (Lu et al. 1999) proposes to generate task models from system models.

When both task and system models have been built, their consistency must be ensured on the lexical, syntactic and semantic levels. The use of Petri nets for the design of these models is of great help in order to check this consistency. Indeed, the same mathematical verification that is used for proving the correctness of a model can also be used to prove their mutual consistency.

Lexical Consistency: This kind of consistency is related to the demand of actions from the task model towards the system model and vice versa. For example if at some point in the task model the user has to press a push-button it is mandatory to ensure that this button exists in the system (and the system model).

This kind of consistency can be automatically proven by checking the action part of the transitions for each model.

Syntactic Consistency: This kind of consistency is related to the sequencing of actions supplied by the models and their related demand. Indeed, as the lexical consistency corresponds to the existence of a requested action, the syntactic one is related to the actual availability of the action at the moment when it is requested. That is to say that the system can perform any sequence of actions requested by the task model. The analysis is based on the calculation of the *languages* corresponding to the task and system models. Indeed, a Petri net can be considered as an acceptor for a language in the same way that a finite state automaton can be. The *supply* of a system model is the language defined by its user transitions and the *demand* of a task model is the language defined by the transitions where a service request is made of the system. Syntactic consistency thus consists of proving that the demand of the task model is included in the supply of the system model. This proof can be performed automatically by building the marking graph of the Petri nets.

Semantic Consistency: To ensure semantic consistency, some meaning has to be ascribed to the places of the models, and it has to be checked that this meaning is preserved in the interaction between task and system models. The semantic of the places is usually apparent from the name of the places. The formulas to be checked have to be given by the designer (in terms of place invariant) but the checking itself can be carried out by automatic means.

Semantic consistency can be checked after the merging of task and system models. This merging is possible because the protocol for the communication between two Petri nets is itself defined in terms of Petri nets (if the task model and system model are modeled using the same notation). It is therefore possible to merge the nets pertaining to the task and system models and to perform analysis on the merger. It is important to consider firstly that this global model may be constructed by completely automated means, and secondly that it is not meant to be read by humans, but merely to be processed by automatic tools, in order to perform analysis and prove properties. The result of this analysis on the reconstructed net allows us to be sure that the models have been built in such a way that they can cooperate together without altering their inner behaviour.

.Using the tools, it is possible to simulate tasks, or different paths to achieve a task, on the task model (a standard or error explicit task model). This scenario can be recorded and manually run on the system model using the Petshop tool. For example, if the simulated task (based on a cash machine example) was:

1. Insert Card
2. Recall PIN
3. Enter PIN

The same tasks could be simulated on the system model, the Insert Card task, an interactive task between the human and the system would be checked by ensuring that a transition representing this external human event is available at the required time. Recalling the PIN is a cognitive task and therefore will most likely not be represented in the system model. The Enter PIN task, is again an interactive task requiring the user to press buttons. These external events for when the user is acting on the interface must be represented as transitions in the system model. The same applies for error explicit task model and system model consistency and coherency checking. The system model should support erroneous human behaviour, such that for example, the system returns to a previous state, blocks the users card etc.

5 A Generic Framework for Ensuring Consistency

Although highly beneficial, it is unlikely that all techniques from all domains of all types of experts will be applied to the design of any given system. This is an unfortunate reality and this is why we must focus on providing a unifying framework to ensure that data of multiple domains can be gathered, refined and embedded

into the design of the system. As previously mentioned, formalising this unified procedure is the only way to ensure that there are no ambiguities, that the description of the models and information is precise, that the framework allows reasoning about the system and to ensure consistency throughout the design and development process.

5.1 Design Process Centred on Formal Models

The framework we propose is grounded on Navarre’s “Design process centred on formal models” (Navarre 2001)p.90. The original framework is presented here in Figure 3-4.

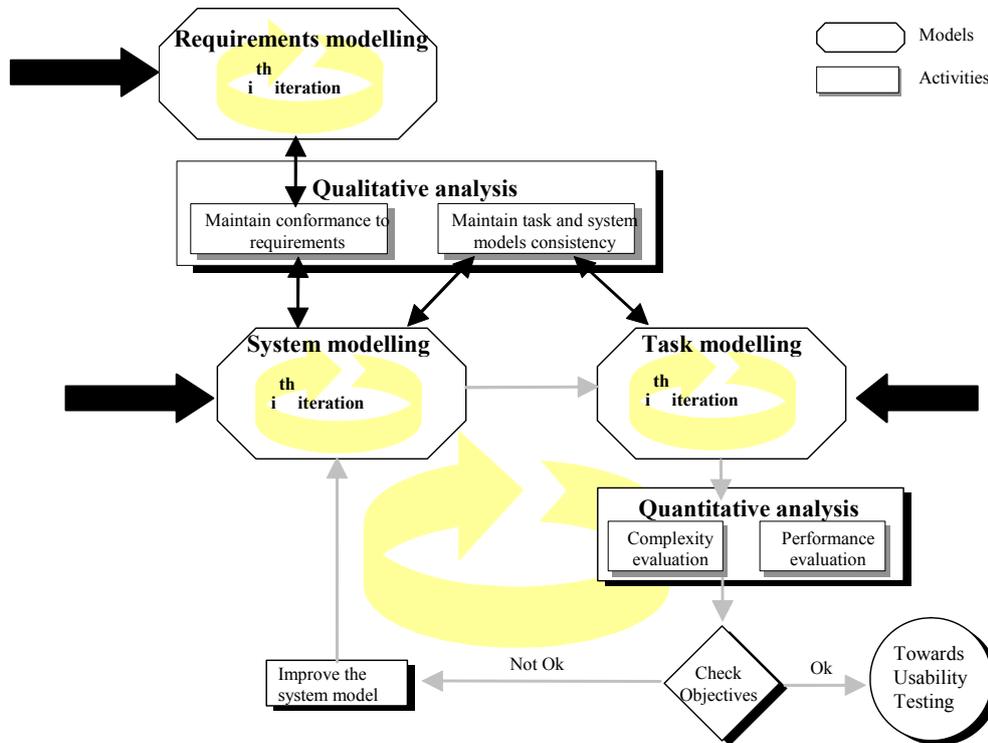


Figure 3-4. Design process centred on formal models (Navarre 2001)

The large black arrows correspond to starting points of the design process of an interactive system. The design process can therefore be initiated either from:

- the requirements specification, if this was carried out before hand
- a task model
- a system model

An iterative design cycle at a high level is therefore initiated where, for each iteration, the system models are constructed in accordance with the task models and where the complexity of these models is analysed in a quantitative way. The designers can therefore propose to modify the system model while aiming to decrease the complexity of the task model. The task model is then modified in accordance with the system model. This permits the designers of an interactive system to propose new interaction techniques to improve the users’ activities. These must then be tested for usability.

Once the first model has been constructed, the process of property verification is implemented in order to determine if the construction is correct. For example, to prove that the system model accurately describes a system that is live, reinitilisable and does not have any potential blockages, or proves the particular states are available according to a task model.

5.2 Generic Integrated Modelling Framework

The generic integrated modelling framework we propose presents a way of bringing consistency between the task model and system model while taking into account human error, safety issues and barriers in order to mitigate the occurrence of erroneous events in safety-critical interactive systems.

Table 3-3 positions our contributions with respect to the state of the art. This is presented by adding a column (describing our work) to the Table 2.5 presented in the conclusion of part 1 page 92.

The additional column shows that our framework integrated tasks and system modelling using both semi and formal notations covering and integrating knowledge from all the human error theories that have been presented in the literature review.

Table 3-3. Positioning our generic integrated modelling framework with respect to the state of the art

Error Classification Name	THEA see Chapter 2 section 3.6) (Pocock et al. 2001)	TAFEI see Chapter 2 s section 3.6) (Baber and Stanton 2004)	Petri-nets see Chapter 1 section 4.1.1.3) (Palanque and Bastide 1997)	User deviations in task performance see Chapter 2 section 3.6) (Paterno and Santoro 2002)	Generic integrated modelling framework
	Based on System modelling Semi-Formal	Task modelling Formal	System modelling Formal	Task modelling Semi-Formal	Tasks and systems modelling Semi-formal and formal
Categories of slips and lapses (see Chapter 2 section 3.5.1.4) (Norman 1990)	x	x	x	x	√
Mode confusion (see Chapter 2 section 3.2.2.3) (Palmer 1995)	x	x	x	x	√
GEMs (see Chapter 2 section 3.5.1.2) {Reason 1990 #430}	x	x	x	x	√
Failure modes {Reason 1990 #430}	x	x	x	x	√
Common mechanisms {Reason 1990 #430}	x	x	x	x	√
Phenotypes & Genotypes (see Chapter 2 section 3.5.1.3) (Hollnagel 1991)	x	x	x	x	√
Skill-based human errors (see Chapter 2 section 3.5.1.1)(Rasmussen 1983)	x	x	x	x	√
HAZOP Causes of deviations (see Chapter 2 section 3.2.2.2) (MOD 1996)	√	x	√	√	√

X : Not taken into account in the approach

√: Taken into account in the approach

Our research focuses mainly on task and system modelling due to the background of our laboratory and our ability to contribute to the domain. Task modelling and system modelling are often performed by experts of specialist domains. It is unlikely that a human factors specialist will design the task model and go on to design the system model. Therefore, it is probable that there will be a mismatch between the two, i.e. a task may have been incorporated into the task model that cannot be performed with the system model. We therefore suggest that it is essential to provide consistency between the two models to test the system complacency and ability to handle all human tasks necessary for achieving a given goal to reduce the possibility of any accident or incident occurring.

Figure 3-5 presents (in no particular order) the various ingredients of the system part as described in section 2.3 of the state of the art (i.e architecture, environment, presentation, platform, dialogue, device and domain models). This component is reproduced in Figure 3-6 (label 1), where interaction with other models is emphasised. Though we consider all of these models belong to the category of system modelling, our contribution to system modelling does not cover the environmental modelling.

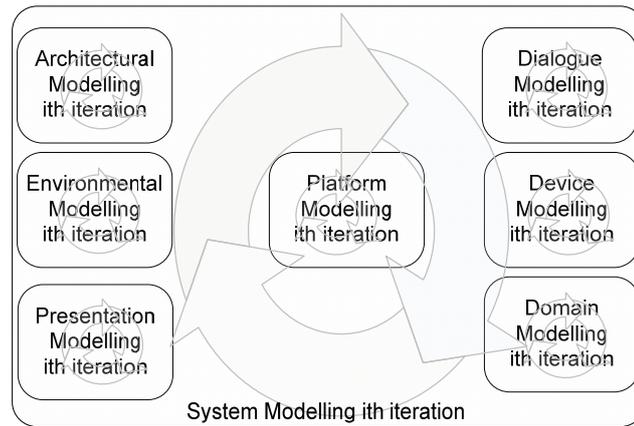


Figure 3-5. Ingredients of the system model

Our research on a multi perspective method for the design of safety critical interactive systems has a goal to propose model based design techniques that allow taking into account and managing erroneous human and system behaviour. The general idea is to bring together, in a unified framework, three aspects of User Centred Design, for reliable interactive systems design, i.e. task analysis and modelling, a formal description of the system, and human factors issues.

We present a generic integrated modelling framework to attain this goal. By generic, we mean that the approach can be applied to safety-critical interactive systems other than those used for our research. Figure 3-6 presents the generic integrated modelling framework which presents as a summary and in a single diagram the set of information, data and processes required for the design of safety safety-critical interactive systems.

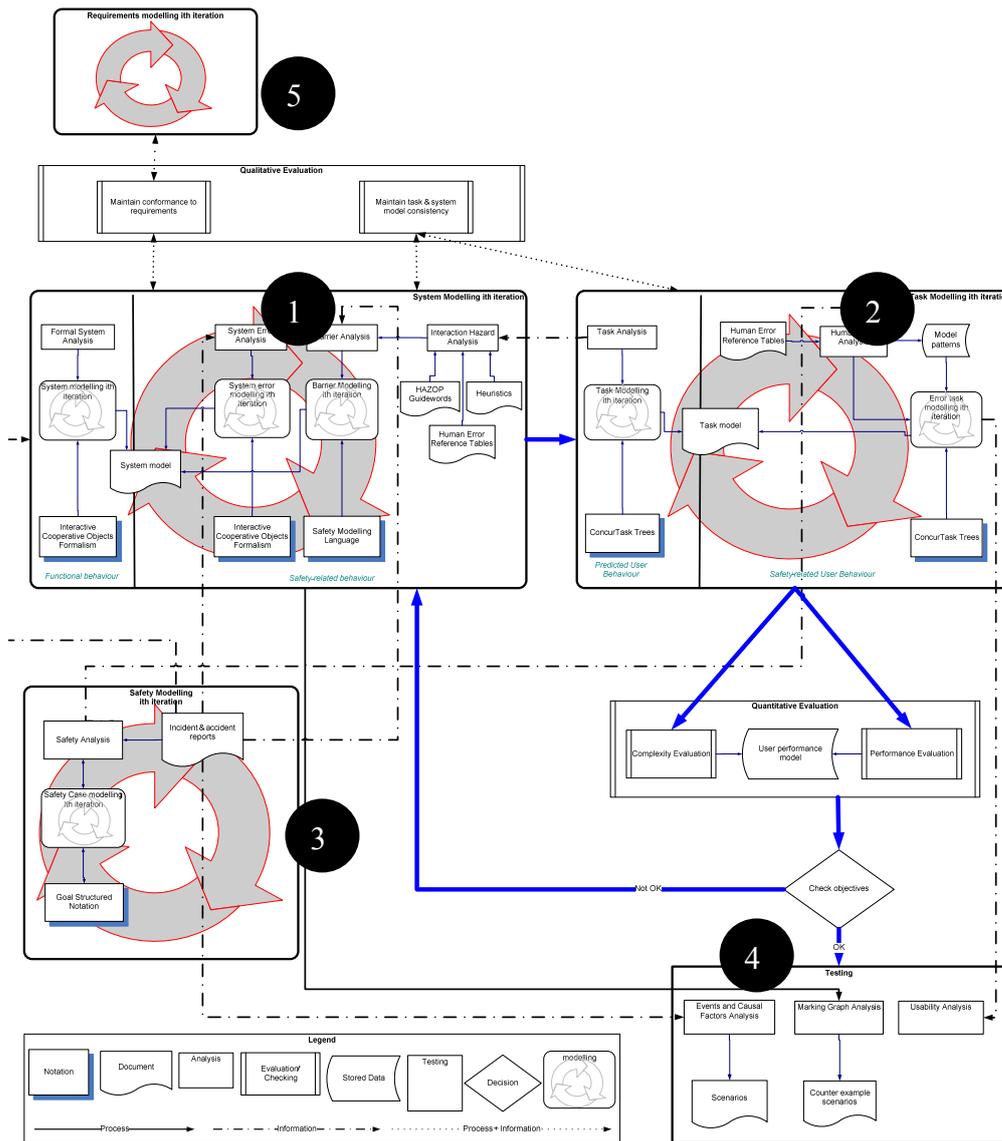


Figure 3-6. Generic Integrated Modelling Framework

The framework represents a cyclic and iterative process for design. As in Navarre’s (Navarre 2001) design process centred on formal models, the process can commence from either the requirements capturing phase, the system modelling phase or the task modelling phase. The emboldened arrows illustrate the iterative procedure that is to be followed though additional processes such as qualitative evaluation and the safety modelling phase can be used to inform the procedure. The solid lines represent processes while the dotted/dashed lines represent information. The only-dotted lines represent processes including information.

The following chapters of this thesis will be based largely on the four principle phases of the framework which are the four principle phases addressed in this thesis. The fifth phase (label 5 in Figure 3-6), requirements capturing has been included in the framework as it is also an important modelling phase of systems design, as well as for safety-critical interactive systems design. However, requirements modelling is a research field within itself and is considered out of scope of this PhD. The interested reader can refer to the IEEE Requirements Engineering Conference.

Before discussing each phase in detail in the subsequent chapters, the following paragraphs provide an overview in the four principle phases with larger diagrams of each phase for increased legibility. The arrows connecting various phases and specific analyses cannot be explained here until the full contents of the phases have been explained in subsequent chapters.

Phase 1 System Modelling (Figure 3-7): For the system modelling we have used the Petshop environment (Navarre et al. 2001a) and the Interactive Cooperative Objects (ICO) formalism (Navarre et al. 2003). The phase is divided into two parts, the modelling of functional behaviour and of safety-related behaviour. The modelling of functional behaviour has been demonstrated using the first case study of a fatal mining accident presented later in Chapter 9 and is also available for the second case study, presented later in Chapter 9. After modelling the system, we perform a socio-technical barrier analysis based on accident report data and a technical barrier analysis based on user interaction hazard analysis on the system model to eventually reduce the possibility of erroneous events. The barriers are also modelled using the ICO notation and Petshop environment. This coupling of barriers and system and what it brings to increase the reliability of an interactive system are presented in subsequent Chapter 7 and is exemplified on the two case studies, the fatal mining accident and the cockpit Multi Purpose Interactive Application.

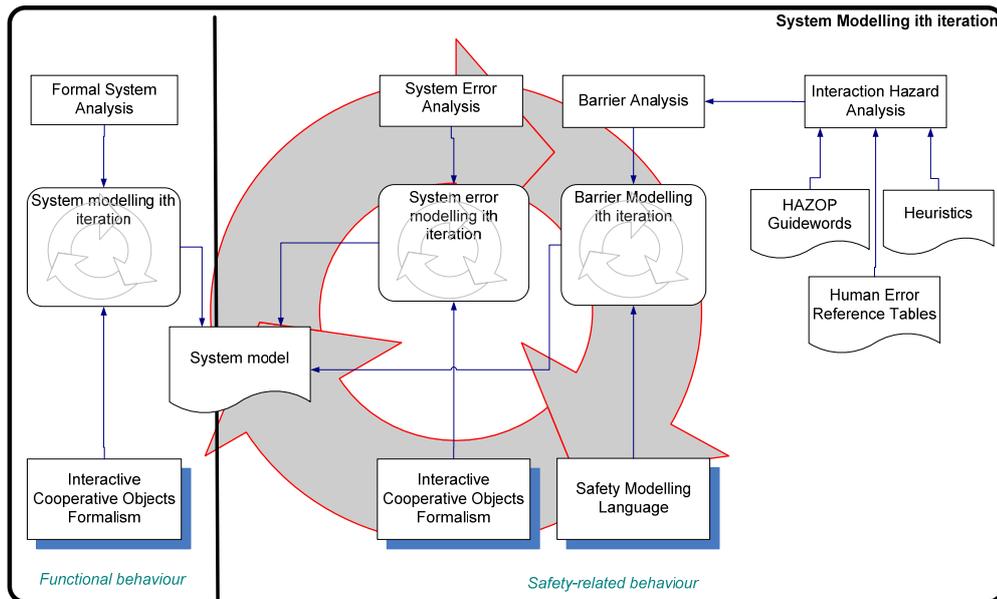


Figure 3-7. System modelling phase of the generic integrated modelling framework

Phase 2 Task Modelling (Figure 3-8): This phase is also divided in two, the first describes the predicted behaviour of the user and results in an analysis of user tasks and activities. The second concerns the modelling of user deviations with respect to predicted behaviour. For this phase, we use the ConcurrentTaskTree (CTT) (Paterno 1999). To take into account erroneous user behaviour, our task modelling is extended by reusable subtasks (called error patterns) resulting from human error analyses. The method for integrating “standard” task models and task models representing potential deviations is described in Chapter 4 and is applied to the fatal mining accident case study.

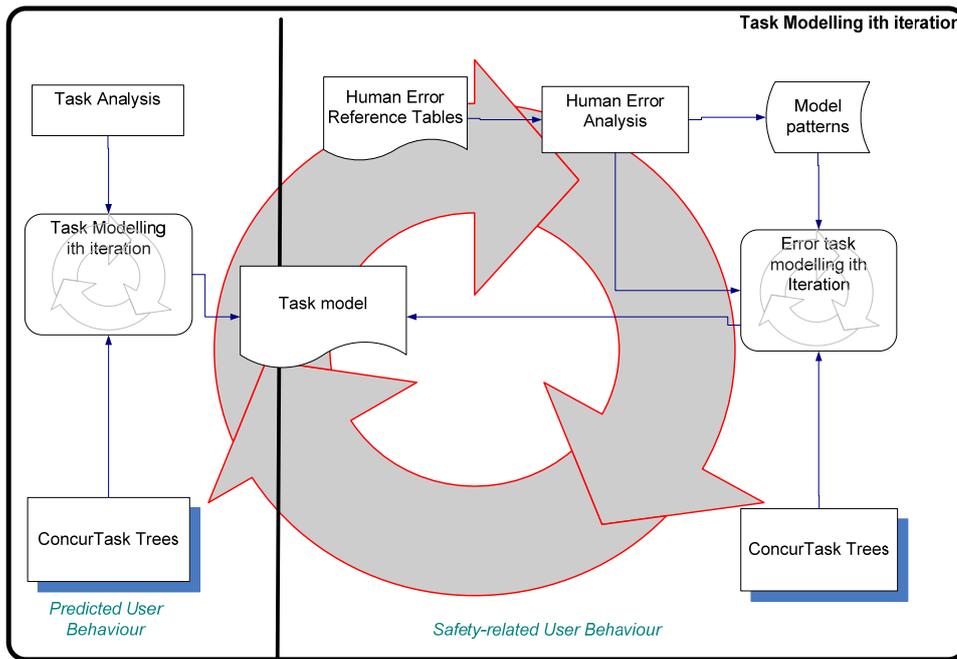


Figure 3-8. Task modelling phase of the generic integrated modelling framework

Phase 3 Safety Modelling (Figure 3-9): In this phase, we propose a safety analysis by exploiting the Safety Cases using the Goal Structuring Notation (GSN) (Kelly 1998) in order to inform the system model. Phase 3 exploits further information including incident and accident reports. It is clear from the number of arrows (4) exiting the incident and accident report block of the framework, that this information is central to our proposed approach since a significant aspect of safety is to make sure successive versions of a system do not allow the same incidents or accidents to be reproduced. The use of safety cases in the proposed approach is described in Chapter 5 and it exemplified on the fatal mining accident case study.

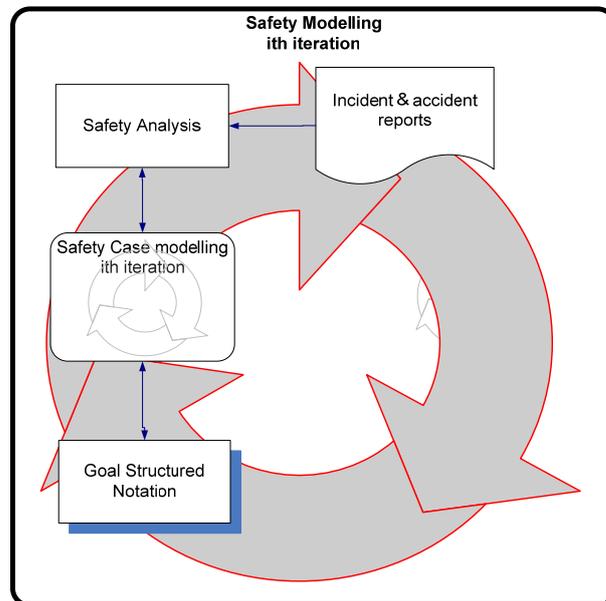


Figure 3-9. Safety modelling phase of the generic integrated modelling framework

Phase 4 Testing (Figure 3-10): This phase concerns the verification of prototypes and models. We use incident and accident reports to perform events and causal factors analyses. We also use the ICO system models to perform marking graph analyses. The graphs are used to systematically explore all of the possible scenarios leading to an accident which described in terms of states in the system model. The techniques serve two purposes. Firstly, to ensure that “modified” system model does not allow the same sequence of events that led to the accident and secondly, our approach can be used to reveal further scenarios that could eventually lead to non-desirable system states. We discuss this concept in Chapter 5 and illustrate it on the fatal mining accident case study in Chapter 8.

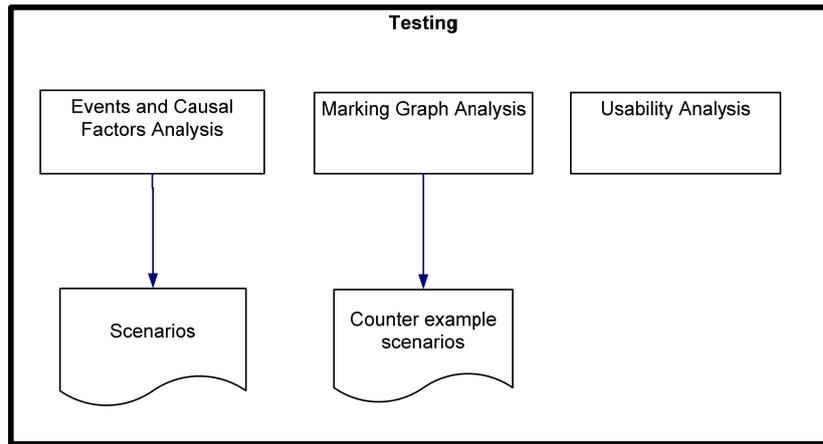


Figure 3-10. Testing phase of the generic integrated modelling framework

6 Supporting Integration

Since the phases have not yet been described in detail, it is difficult to discuss the way in which the phases, methods, techniques, processes, notations and tools are integrated. For this reason, each of the following 5 chapters will have a dedicated section, towards the end of the chapter detailing how the phase presented fits in with other corresponding phases.

7 Conclusion

The proposed framework advocates the use of models for the design of interactive safety critical systems. It claims that the issues raised by the design of such systems require the use of systematic ways to support the gathering, refinement and storage of data. This data is, by nature, multi-disciplinary and thus requires a multi-notation approach to support individually each discipline.

However, this multi-notation approach calls for additional means in order to support additional activities such as verification of models consistency. Besides, in order to alleviate the burden for developers and designers, software tools supporting their activities are also at the core of the applicability of such an approach.

We have studied and propose methods for integrating the necessary models for safety-critical interactive systems design. We have devised approaches for integrating task models and system models while using various techniques to take into account erroneous human and system behaviour. Our approach relies heavily on scenarios to bridge the two principle modelling phases. These concepts are explained in subsequent chapters which details each of the principle phases.

The framework provides support for the gathering, refining and embedding of multi-source and multi-type data while tolerating experts of their multiple domains to perform analyses and use techniques they are accustomed to. All of these components, we believe, are essential for the design of safety-critical interactive systems design.

Chapter 4

Task Modelling & Error Management

“Anyone who has never made a mistake has never tried anything new.” (Albert Einstein, 1879 - 1955)

Chapter Summary

In this chapter we present our contributions to the task modelling phase of the model-based design of safety-critical interactive systems. We propose the notion of “model patterns for error tolerance” as a means of systematically taking into account erroneous user behaviour in a task model. The method and techniques used are grounded on previous work in the fields of task modelling and human error analysis and identification. In order to design the model patterns, a human error analysis must first be performed on a “standard” task model. This is achieved using our Human Error Reference Tables (HERTs) which classify a significant number of known human “errors”. Model patterns for error tolerance are necessary to deal with the complexities that arise when trying to consider human deviations in task models, such as model explosion which makes models unreadable.

1 Introduction

Previous chapters, particularly the state of the art has argued that formal specification techniques are advantageous for the design of safety-critical interactive systems because they provide non-ambiguous, complete and concise notations. However, formal specification of interactive systems (even though aiming to produce reliable software) often does not address the issues of erroneous user behaviour. Indeed, the focus is mainly on describing ‘normal’ behaviour of users while difficulties mainly arise due to unexpected or invalid (according to the system’s description of valid behaviour) user actions performed.

Although task modelling is recognised as a useful part of the design process of interactive systems, the construction of accurate task models is still a challenging task. There is no general consensus on the best level of detail to be used in task models, and in particular on the way human error should be considered in task modelling. Introducing error handling in task models usually results in an explosion of the complexity of the models, without any clear benefit, since the models become too complex to be readable.

Furthermore, task analysis and modelling approaches have always focussed on standard behaviour of users leaving user error analysis for later phases in the design processes (Baber and Stanton 2004). This is part of the rationale underlying task analysis which is to provide an *exhaustive* analysis of user behaviour. This comprehension (of analysis) is critical as it is meant to provide the basics for a global understanding of user behaviour and tasks that will serve as a basis for driving evolutions of the interactive system. However, practice shows that reaching this exhaustiveness is very difficult in terms of availability of resources and economy. These aspects drive people responsible for task analysis and modelling to focus on most frequent and standard activities, thus leaving the infrequent or erroneous ones unconsidered. However, this is precisely where the emphasis should be placed, in order to deal efficiently with error tolerance.

We extend task models and cooperative task models (describing standard user behaviour) with erroneous user behaviour. Without appropriate support, incorporating erroneous user behaviour in task models requires much effort from task analysts. We thus propose the definition of model patterns for error tolerance in task models as a way of dealing exhaustively with potential user errors. Our approach builds upon previous work in the field of task analysis, task modelling, human error analysis and identification.

The model patterns can be directly reused within a task model in order to represent potential deviations of user behaviour. The model patterns are systematically applied to a task model in order to build a task model covering both standard and erroneous user behaviour.

The notion of model patterns proposed herein refers to the seminal work of Gamma et al. (Gamma et al. 1995) who popularized the notion of Design Pattern as a higher-level reusable structure for software systems. A design

pattern presents a piece of Software Engineering knowledge in a form that makes the recurring problem it solves easy to understand and to reuse while adapting it to the system under design. In the same vein, our goal is to present human errors in an easily communicable and processable form that can be reused to investigate the potential hazardous effects human errors can have in task models.

The task models (including representations of possible user errors) can then be exploited towards system models to provide a systematic way of assessing both system compliance to user tasks and to verify that the system under development is able to tolerate such erroneous user behaviour.

The principle is similar to the work presented in (Fields et al. 1999) in terms of objective. However, our approach focuses not only on ways of identifying erroneous user behaviour, but also on representing such behaviour and on integrating it with standard user behaviour.

This precautionary approach will minimise the likelihood of erroneous events by designing the system to deal with such events. This can be by designing the system to return to a safe state after a problematic event has taken place.

The approach proposed relies on several premises:

1. Our work is geared for the design of a new interactive system (whether starting from an existing one or from scratch), rather than for the investigation of an existing system, where task modelling is merely used to faithfully describe and record the activity of users actually using the system.
2. We assume a design process where task models are produced during the initial phases of development. In this usage of task modelling, task models are essentially useful to provide an abstract view of the user's activity, exploring their goals as well as the temporal and causal structure of their interactions with the system. In particular, we assume that the results of task analysis will be used as an input to an Interaction Design phase, where the user interface specialist will strive to design an interface that is best suited to the user task, while taking into account the limitations inherent to the target platform for the interactive system. Since task models are produced before the user interface is designed, they should be devoid of references to user interface specifics (such as "The user clicks on the radio button"), but should instead express abstract user actions (such as "The user selects from the available options", that do not unduly constraint the forthcoming design of the user interface.

Considering these premises, we believe that task models should remain of manageable size, to better convey their intended meaning to the interaction designer. This precludes a very fine-grained level of detail, where every possible user error and the corresponding system reaction would be described. However we do not want to disregard user errors altogether, and to postpone error handling to the implementation phase, where the programmer would be responsible of managing user errors without the high-level human-factors knowledge that is required.

Our work aims at finding an appropriate compromise between these two constraints: enabling the construction of simple and useful task models, while taking into account human error at the right phase, i.e. during requirements gathering. When error-free task models are transformed by the injection of model patterns for error tolerance, the resulting error-laden task models can be used at several stages of the design process. During early stages, the error-laden models can be used to exercise early mockups or prototypes of the user interface to investigate how the system should best react to potential user errors. In the later stages, error scenarios can be used to test that the final software is error-tolerant, by using the CTTe tool to generate usage scenarios.

This chapter therefore focuses on the task modelling phase of the generic integrated modelling framework presented in Chapter 3 where our contribution lies in the design and implementation of model patterns for error tolerance. The model patterns are grounded on established theory in the field of cognitive psychology. The proposed approach and techniques tackle the problem of not taking into account erroneous user behaviour by applying a systematic way of dealing with such important data, via our classification of human errors, referred to as Human Error Reference Table.

The approach and techniques presented here could fit any of the task modelling notations discussed in the state of the art. The choice of CTT to serve as the notation for task modelling in this work is motivated by the fact that it is a widely used notation, supported by a tool that allows for graphic edition and execution of the models.

2 Task modelling

The state of the art introduced the concept of task modelling, though before we can demonstrate the model patterns for error tolerance, we will first present a basic task model using an ATM as an example. Following which, we will define our contribution on the classification of human errors for the systematic application to subtasks of a task model allowing us to introduce the notion of model patterns.

We show, on a simple ATM case study, how this approach can be used and what it brings to the design and verification of error-tolerant safety critical interactive systems.

With respect to the generic integrated modelling framework, a standard task model without the consideration of erroneous human behaviour is represented on the left hand side of the task modelling phase highlighted with dashed lines in Figure 4-1. The left hand side is labelled “predicted user behaviour” as this is what a standard task model represents. It assumes the user will behave in a correct, predicted and modelled way. The upper layer is the task analysis which feeds into the task modelling block. The task modelling is achieved using the CTT notation represented in the lower part of the diagram. The task model itself, considered a document, is represented as overlapping the left hand side (predicted user behaviour) of the task modelling phase and the right hand side (safety-related user behaviour). This is because the standard task model produced will receive input from the error modelling discussed later in this chapter.

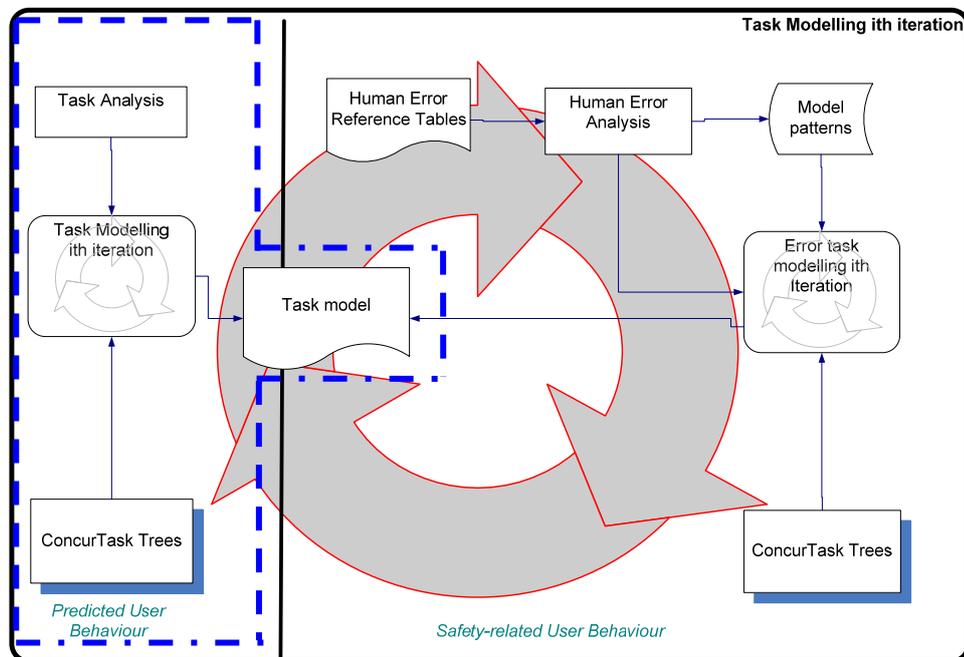


Figure 4-1. Task modelling phase of the generic integrated modelling framework highlighting predicted user behaviour

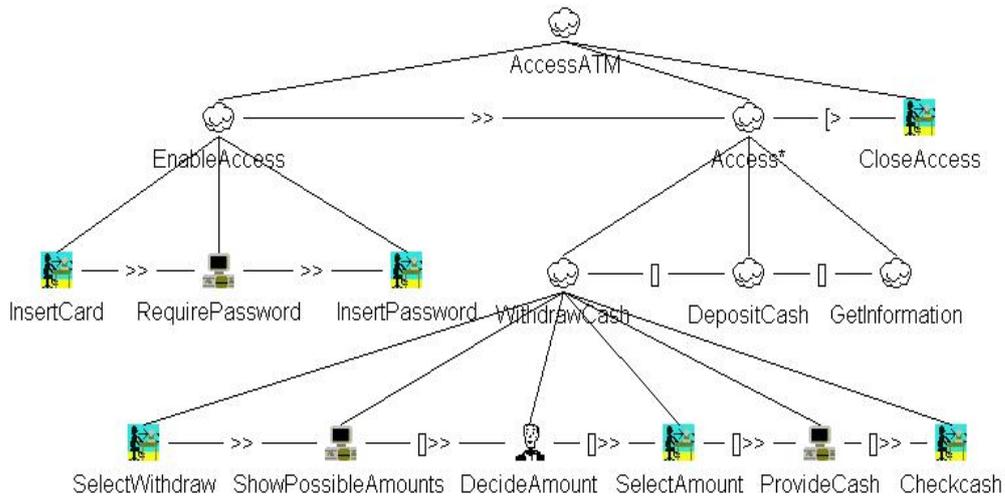


Figure 4-2. CTT ATM example provided within CTTe

Figure 4-2 shows, at a level of description of most task analysis methods, with addition of temporal operators, the task of accessing an ATM. The diagram is read from left to right. A task is not complete until its necessary sub-tasks are addressed. Thus in the above example, EnableAccess must first be completed by performing of its subtasks. That is, the user inserts the card (an interactive task) which enables the request of the PIN (a system task) which in turn enables the entering of the PIN (an interactive user task). The success of the EnableAccess task enables the Access task which is composed of WithdrawCash or DepositCash or GetInformation. Please note however, that the Access task is iterative which means that each of the Access subtasks can be performed in any order. The iteration of the Access task is aborted by the interruption of the CloseAccess task.

3 Error Management

As we have seen in the state of the art, human “error” plays a major role in the occurrence of accidents in safety-critical systems such as in aviation, railways systems, or nuclear power plants (Reason 1990). The cause for many recent disasters is portrayed in the press as being human “error” with news headlines such as “Human error caused Helios crash” <http://news.bbc.co.uk/2/hi/europe/6036507.stm> (The Cypriot Helios Airways Boeing 737-300 crashed near Athens in August 2005, killing all 121 people on board), or “Cable crash 'human error' claim” http://news.bbc.co.uk/2/hi/uk_news/scotland/highlands_and_islands/5189324.stm (A cable car collision in the Scottish Highlands in which five people were injured on July 13 2006) and “Human error caused aircraft crash” http://news.bbc.co.uk/2/hi/uk_news/england/southern_counties/5342376.stm (A light aircraft crash which killed a 25-year-old pilot in Surrey, 13th September 2006).

3.1 Designing with error in mind

Interactive systems particularly those that are safety-critical need to be designed with the eventuality of human error in mind to prevent catastrophes. This means, early in the design process as well as during the testing phase. Our approach, which tackles the inclusion of human “error” during the task modelling phase is an initial phase of the development cycle. We are not claiming to be able to account for *all* eventualities. When designing, there is information we are certain about and there is information we are not certain about, including potential human-computer interaction failures. It is obviously more difficult to take into account information that we are unsure of though we believe that by at least considering some potential cognitive failures we will improve the safety of the design.

Although the term “human error” is controversial, theories of human errors such as Rasmussen’s (Rasmussen 1983) SRK, Hollnagel’s (Hollnagel 1991) Phenotypes and Genotypes and Norman’s (Norman 1988) classification of slips, (all of which presented in the literature review) are considered widely acceptable. In the next section we present our Human Error Reference Table (HERT) which is grounded on the skill-based part of Rasmussen’s theory while also grouping together many user oriented error types appropriate for our studies.

Human behaviour and therefore erroneous human behaviour is a subcomponent of the resilience engineering concept (as well as organisational and managerial issues). The view in this thesis is that by accounting for human error, technical errors, interaction failures, software, hardware and socio-technical barriers will improve the resilience of a system. Though we recognise that human error may compromise resilience, understanding how users and even systems recover from erroneous behaviour in order to sustain resilience has not been considered.

The Skill-based HERT as well as the heuristic evaluation and HAZOP analysis for interaction hazard analyses are methods for accounting for erroneous behaviour while interacting with a system. The considered errors are forms of deviations such as omitting an action, branching errors, too early, too late etc. We state however, that the aim of the approach to consider erroneous behaviour in the design of safety-critical interactive systems is not to account for all eventualities. The difficulty is making explicit things we do not know. Though we do not know which erroneous behaviour may occur during system interaction, established theories such as those mentioned provide examples of errors that are known to occur. We provide a means for accounting for things we know about in the hope of making a system more resilient to such errors. Furthermore, by anticipating potential failures, we can improve the system making it more error tolerant.

Dearden and Harrison (Dearden and Harrison 1995) make clear the difference between human error resistance and human error tolerance. According to the authors, a system is defined as human error resistant if it ‘reduces the probability that a human operator interacting through the interface will attempt to perform actions that endanger the safety of the system’. A system is defined as human error tolerant if ‘the consequences of a human operator performing erroneous actions through the interface are limited to some acceptable level’. The terms limited and acceptable level are subjective and would depend greatly on the system in use. Our approach tackles the second definition in that we attempt to improve the system such that known erroneous behaviour is considered in the design. These considered errors allow us to improve the system to deal with them. We do not however tackle resistance because we have no direct impact on the users using the systems so far. We are not able to influence users, via training or guidance for example to reduce the likelihood of them performing erroneous actions. Furthermore, the concept of resilience should again be separated from the concepts of error tolerance and error resistance which contribute to a more resilient system.

The framework only accounts for non-intentional, non-malicious erroneous operator behaviour and barrier violation. For example, in the fatal mining accident, the pump priming procedure was violated in addition to the manufacturer guideline of not opening valves while the motors were in operation. These barrier violations were without intention to damage the system, not with the intention of improving any other criteria, but simply because the operators were unaware of these barriers/manufacturer guidelines. Furthermore, the skill-based errors and interaction hazards considered when interacting with software are non-intentional behaviour. Part of the skill-based HERT includes “Over-Motivation: Can come from being too zealous, eg: completing a job too quickly just to please a supervisor. Working fast can lead to shortcuts and risk-taking” which is along the lines of violation without intent to damage the system but with intention to satisfy another criteria. Analysing intentional violations with respect to barriers or deviations etc are outside the boundaries we set for the PhD. This is not to say that they are important, and form the basis of future work

3.2 Human Error Reference Tables

To remind the reader of which section of the generic integrated modelling framework we are referring to, the diagram is reproduced (see Figure 4-3) this time highlighting the right hand side, safety-related user behaviour.

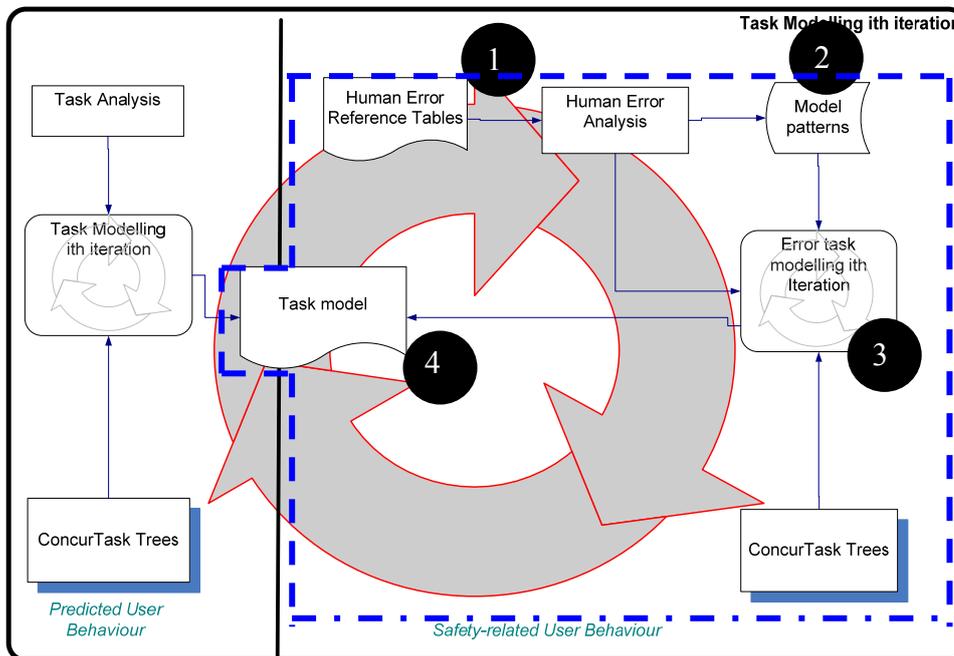


Figure 4-3. Task modelling phase of the generic integrated modelling framework highlighting safety-related user behaviour

In order to perform a human error analysis based on task models, we have devised three reference tables of human “errors” called Human Error Reference Tables (HERTs) (number 1 in Figure 4-3). Taxonomies are an important tool for description, categorization and analysis of human errors in critical domains. The table is based on Rasmussen’s (Rasmussen 1983) Skill-based error types. We selected this theory as the highest level of classification as we believe and show that most human “error” theories and classifications fit into these three categories.

The Human Error Reference Table (HERT) has been limited to the representation of skill-based errors since determining whether an erroneous action was the result of lack of knowledge or application of the wrong rule would require explicit representation of knowledge and belief that is beyond the scope of this PhD. In addition to the representation of the knowledge a person has gathered over time, there is also a need to represent the current “value” of that knowledge at the execution time of the tasks. Integrating such information in our framework would involve additional activities more related to user cognitive processes. In addition validation of such information would require intensive activities on each user of the system. For these reasons we have followed a pragmatic approach focussing on skill-based errors that do not require interpretation. It is however clear that such knowledge-based and rule-based error might have a significant impact on safety.

Figure 4-4 illustrates (as a screenshot) the skill-based error table. The table is not easily legible though has been shown to provide the reader with an idea of a complete table. Broken down versions of the table can be found in Appendix. It must be made clear that the research in this thesis has focused on the use of the skill-based human error reference table only (i.e. not the rule-based or knowledge-based tables). This is because we have not performed analyses with real system operators. In order to apply and understand rule-based or knowledge-based errors, we would first need a sound understanding of the operators. Further research into potential violations of rules would be necessary and is out of scope of this thesis.

The table is composed as follows and is read from left to right. The first left hand column entitled “Reason’s GEMS & Rasmussen’s SRK Classification” defines the highest level of classification of this table, i.e. the skill-based level. Within this category, James Reason has defined two main failure modes (2nd column of the skill-based HERT), errors of inattention (omitted checks) and errors of over-attention. The third column of the table, entitled “Reason’s Common Mechanisms” form the classification of both the inattention and over-attention errors, comprising of six mechanisms. These are double-capture slips, omissions, reduced intentionality,

perceptual confusion, interference errors and mistimed checks. Each common mechanism is followed by a definition and reference. Though within the third column, one can observe that additional text has been added (red if printed in colour). This is because, Reason's definition of double-capture slips correlates with Norman's (Norman 1990) 1st category of slips. The fourth column is labelled "Error name and definition". The previous common mechanisms have been further broken down into fine grained error types. Where available, definitions of these are given with references. The fifth column, which we believe will be particularly useful for those using the tables and performing the analyses of task model subtasks, provides a day-to-day example of each type of error, unless the error name in itself is deemed clear enough. The final column provides a reference for each of the fine grain error types listed in column four.

This phase of our multidisciplinary model-based approach for the design of safety-critical interactive systems allows us to determine the types of errors that could occur during each subtask of a single user (which in combination could be applied to a cooperative task model) task model.

The tables provide a tool for the measurement and the study of human errors. We believe that understanding the types of error and the causes of errors are vital to evaluating the root causes of potential problems and fixing them.

Chapter 4 - Task Modelling & Error Management

Reason's GEMS & Rasmussen's SRK Classification	Reason's Main Failure Modes	Reason's Common Mechanisms	Error Name & Definition	Examples of errors	Error Type Ref	
S k i l - b a s e d I e v e l	I n a t e n t i o n (O m i t t e d c h e c k s)	Double-capture slips <i>Involve two distinct, though causally related, kinds of capture. (Reason, 1990) (Norman's 1990 1st category of slips)</i>	Strong-habit intrusion <i>The unintended activation of the strongest action scheme beyond the choice point. (Reason, 1990) Hollnagel 1993, simple phenotype, intrusion. Error mode: action not included in current plans. Corresponding complex phenotype, Hollnagel, 1993 capture, branching and overshoot.</i>	Unintended activation of the strongest action schema beyond the choice point. (Reason, 1990)	Reason, 1990	
			Strong-habit capture	On starting a letter to a friend, I headed the paper with my previous home address instead of my new one. (Reason, 1990)	Reason, 1990	
			Strong-habit exclusion	I intended to stop on the way to work to buy some shoes, but "woke up" to find that I had driven right past. (Reason, 1990)	Reason, 1990	
			Branching errors <i>An initial common action sequence leads to different outcomes and the attentional check at the choice-point is omitted. (Reason, 1990)</i>	I brought the milk in to make myself a cup of tea. I had put the cup and saucer out previously. But instead of putting the milk into the cup, I put the bottle straight into the fridge. (Reason, 1990)	Reason, 1990	
			Overshooting a stop rule	I meant to get my car out, but as I passed through the back porch on the way to the garage, I stopped to put on my wellington boots and gardening jacket as if to work in garden. (Reason, 1990)	Reason, 1990	
			Program Counter Failures Counting <i>An extraneous event as a part of the intended sequence - causing an omission or failing to count a relevant action causing repetition which is less common (Reason, 1984)</i>	I went to my bedroom to change into something more comfortable for the evening, and the next thing I knew I was getting into my pajama trousers, as if to go to bed. (Reason, 1990)	Reason, 1990	
		O m i t t e d c h e c k s)	Omissions following interruptions <i>Failure to make an attentional check is compounded by some external event. (Reason, 1990)</i>	Trips	I picked up my coat to go out when the phone rang. I answered it and then went out of the front door without my coat. (Reason, 1990)	Reason, 1990
				Detached Intentions <i>The action element of the sequence of events either is neglected or performed with another object or trigger from an external distraction becomes incorporated into the desired sequence.</i>	A trip is a forcible interruption to an ongoing motor program, as in a trip while walkin. (Smith, 2002)	Reason, 1997
				Environmental Capture	I intended to close the window as it was cold. I closed the cupboard door instead. (Reason, 1990)	Reason, 1990
				I-should-be-doing-something-but-I-can't-remember-what	I went into my bedroom intending to fetch a book. I took off my rings, looked in the mirror and came out again - without the book. (Reason, 1990) (Norman's 1990 5th category of slip. Loss of activation error)	Reason, 1990
				Fumbles	(Norman's 1990 5th category of slip. Loss of activation error)	Reason, 1990
				Post-completion error <i>There is a precondition to the conceptual operation that achieves the main goal, but satisfying the precondition perturbs the state, and a clean-up action is needed after achievement of the main goal (Blandford, 2002)</i>	A fumble is a poorly executed motor program, as in a fumbled catch at cricket. (Smith, 2002)	Reason, 1997
	I e v e l	Reduced Intentionality <i>Some delay intervenes between the formulation of an intention to do something and the time for this activity to be executed. (Reason, 1990)</i>	Under-motivation <i>Stemming from boredom or lack of morale also can lead to shortcuts and risk-taking. However, it is more likely to cause latent errors, discovered later by someone else (Bill Mostia, 2003)</i>	Making copies on a photocopier but forgetting to retrieve the original and forgetting to take change from a vending machine (Curzon & Blandford, 2004)	Byrne & Bovair, 1997	
			Description error <i>When the correct action is carried out on the wrong object. It tends to occur when one is distracted. (Preece, 1994)</i>	Putting the salad in the oven and the casserole in the fridge (Preece, 1994) (Norman's 1990 2nd category of slip. Description error)	Norman, 1989	
			Input or Misperception errors <i>With errors of this type, the information needed to make a decision or perform a task might be misunderstood, perhaps because it has been presented in an overly complex or misleading way. (Mostia, 2003)</i>	The input data are incorrectly perceived, an incorrect intention is formed, and the wrong action is performed; that is, an action other than what would have been intended, had the input been correctly perceived (Green, 2003)	Mostia, 2003	
			Data driven error <i>When unconscious processing of external data interferes with what you had intended to do. (Preece, 1994)</i>	Saving a file with the name of the file that is displayed in the adjacent window instead of the name you intended. (Preece, 1994) (Norman's 1990 3rd category of slip. Data driven error)	Norman, 1989	
			Associative-activation error <i>When internal thoughts interfere with what you were supposed to be doing. (Preece, 1994)</i>	Saving a file with the name of the person you were thinking about rather than the name you had intended. (Preece, 1994) (Norman's 1990 4th category of slip. Associative-activation error)	Norman, 1989	
			O v e r t e n t i o n	Mistimed Checks <i>When an attentional check is omitted, the reins of action or perception are likely to be snatched by some contextually appropriate strong habit or expected pattern. (Reason, 1990)</i>	Omission <i>Occurs if an action has been omitted from the sequence. (Hollnagel, 1993) (Hollnagel's 1993 simple phenotype. Error mode: action at the wrong place. Corresponding complex phenotypes, jumping and undershoot.)</i>	Omit a necessary step like putting tea in pot, or switching on kettle. (Reason, 1990) Part 1a of HAZOP causes of deviation
	Repetition <i>Means that an action has been carried out twice (Hollnagel, 1993) (Hollnagel's 1993 simple phenotype. Error mode: Action in the wrong place. Corresponding complex phenotype, Restart</i>	Setting the kettle to boil for a second time. (Reason, 1990)			Hollnagel, 1993	
	Reversal <i>Means that two actions have been reversed in their sequence. (Hollnagel, 1993) (Hollnagel's 1993 simple phenotype. Error mode: Action in the wrong place. Corresponding complex phenotype, Jumping.</i>	I intended to take off my shoes and put on my slippers. I took my shoes off and then noticed that a coat had fallen off a hanger. I hung the coat up and then instead of putting on my slippers, I put my shoes back on again. (Reason, 1990)			Hollnagel, 1993	
	Insertion <i>Occurs if the action does not belong to the action sequence and if it does not disrupt it. (Hollnagel, 1993) (Hollnagel's 1993 simple phenotype. Error mode: Action not included in current plans. Corresponding complex phenotype, side-tracking.</i>	Part 1b of HAZOP causes of deviation			Hollnagel, 1993	
	Replacement <i>The action is a proper substitute for the expected action. (Hollnagel, 1993) (Hollnagel's 1993 simple phenotype. Error mode: Action not included in current plans. Corresponding complex phenotype, side-tracking.</i>				Hollnagel, 1993	
	Premature Action <i>An action that occurs when no action was expected. (Hollnagel, 1993) (Hollnagel's 1993 simple phenotype. Error mode: Action at wrong time.</i>				Hollnagel, 1993	
	Order Errors <i>There are circumstances in which doing A then B has an indifferent effect from doing B then A, that the actions can be performed in either order, and that the user may not be aware of the order constraint. (Blandford, 2001)</i>				Blandford, 2001	
Over-Motivation <i>Can come from being too zealous, eg: completing a job too quickly just to please a supervisor. Working fast can lead to shortcuts and risk-taking. (Mostia, 2003)</i>		Mostia, 2003				

Figure 4-4. Screenshot of Skill-based Human Error Reference Table

We propose that a HCI specialist will use Human Error Reference Tables (HERTs), to systematically work through the list of fine grain error types and analyse sub-tasks of task models to determine the impact of the ‘error’ on the task in hand. If the development process is at an appropriate stage, where for example prototypes of the system are available, the analyst can also determine what the impact the error would have on the system. Perhaps the system tolerates the error or perhaps the error has not at all been considered and would have serious implications.

The benefit of producing such reference tables enables the exact identification of very precise types of ‘error’ while analysing human behaviour associated to particular subtasks of a task model.

The aim of these reference tables is not to guarantee a comprehensive and exhaustive identification of every possible eventuality but to provide investigators with systematic ways of exploring likely and reoccurring user deviations.

Section 4 provides an illustrative walk-through of the procedure of analysing a subtask of a task model for Skill-based human “errors”.

4 Model Patterns for Error Tolerance

We propose a model-based approach to integrate human error analysis with task modelling, introducing the concept of model patterns for error tolerance. This extends standard task modelling by incorporating human error analysis. Due to its complexity, we propose the definition and use of model patterns for dealing with complexity and repetitions that frequently appear when modelling erroneous user behaviours as a means of reducing workload. Model patterns are prototypical deviations from abstract task models, expressed in a semi-formal way by a model transformation. A collection of typical errors taken from the literature on human errors is described within our framework and are represented in the HERTs as shown in the previous section.

The intent is that the human factors specialist will produce the task models taking an error-free perspective, producing small and useful task models. The specialist will then choose from the collection of model patterns for error tolerance, and selectively apply these patterns to parts of the original task model, thus producing a transformed model, error-laden model exhibiting erroneous user behaviour.

This transformed task model can be used at various stages of the design process, to investigate the system’s reaction to erroneous behaviour or to generate test sequences. For example, the extended task model can be systematically exploited on its corresponding system model to determine any system flaws. If necessary, the system model can be iteratively redesigned to ensure both system compliance to user tasks and system tolerance to user errors. This will be illustrated later in Chapter 5 after having presented the interactive systems modelling phase.

We propose the notion of model patterns for error tolerance, designed to be easily usable in a Model-Driven Engineering (MDE) framework such as the generic integrated modelling framework at the core of this thesis. We give a general, technology independent definition of model patterns: a model pattern for error tolerance is a semi-formal description for a class of domain-independent user errors belonging to a structured classification of errors.

4.1 Relating types of user “error” to model patterns

This section describes how to relate the fine grain human “errors” to the subtasks of task models. Since the HERTs have been decomposed to the level of an example for each type of error, it is possible to relate every classified error to a particular subtask. Thus for each task of a task analysis model, it is possible to determine, by means of elimination, which human errors are applicable.

Once specific error-related situations for each task of a task model have been identified, it is possible to re-design the model to support and make explicit these types of errors. At this stage, if there is an existing system in place, its constraints will need to be considered. This ensures, that when the system or future system is modelled, for example as a Petri-net, the task model can be tested over the system model with the human errors already identified. This would avoid the discovery of problems during later testing phases of the development lifecycle.

Finally, the re-designed error explicit parts of the task model, the model patterns, can be ‘plugged in’ to the relevant areas of the task model. We anticipate that the model patterns can be applied to other domains involving the same or similar activities.

Since the possible interactions with an ATM may sometimes be complex and plentiful, we focus our attention on producing a task model for the process of withdrawing cash using the CTT notation (see Figure 4-5).

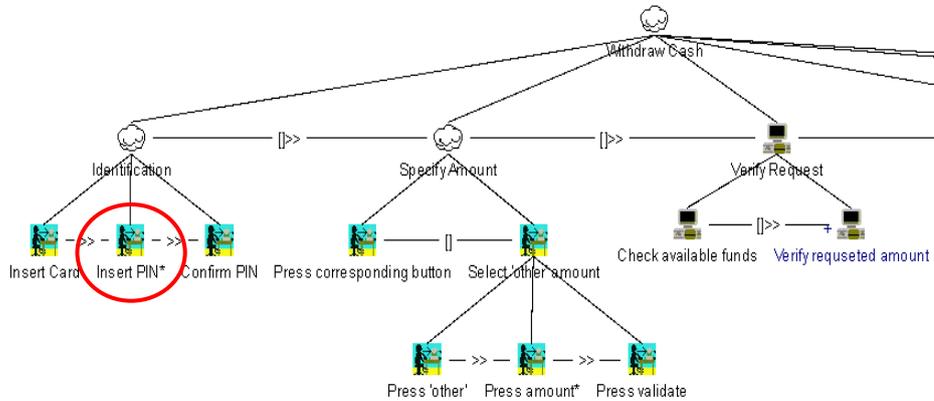


Figure 4-5. Sub-section of CTT Task Analysis for Withdrawing Cash at an ATM highlighting the “Insert PIN” activity for analysis

The CTT diagram for cash withdrawal at an ATM shows part of the process of being identified, selecting an amount, through to the withdrawal of the money. We are referring to this CTT as a general representation of task analysis because it describes error free behaviour and “normal” flow of activities. For example, if the same task was analysed using the HTA approach, the resulting textual or graphical representation would be somewhat similar. However CTT allows for temporal operators to be described, which is particularly useful for modelling interaction with an ATM, resulting in a more realistic model in terms of possible user activities.

4.1.1 Identification of possible deviations

Following the task analysis and modelling, each low-level subtask can be considered for possible human error events that may occur (number 2 in Figure 4-3). This can be done by referring to the HERTs and determining whether or not each particular type of error could occur to each CTT task.

As an example, we present the systematic analysis of possible human deviations while interacting with an ATM based on the “insert PIN” sub-task of the Identification task highlighted in Figure 4-5 and using the Skill-Based HERTs only. Several, Human deviations, relevant to the Insert PIN activity, have been identified. This can be seen in Figure 4-6.

Ideally, the results should be as closely related to the human task as possible. Though as with most task analyses, there is always a problem of separating the user task from the system. For example “insert PIN for card used more frequently” is considered task related as it does not refer to the system or interactions with the system. A result such as “Correct PIN entry, validate before appropriate, press key not associated to insert PIN process” refers to the system interface.

The results in Figure 4-6 have significant references to the system interface itself. This may be because the example of an ATM is a common device, though in theory (as we shall see later in Chapter 5 on system modelling), a well-known system such as an ATM can behave in a completely unanticipated way and still be efficient (depending on requirements of stakeholders).

It must be noted however, that although many possibilities of human “errors” have been considered, we have sensibly limited our analysis to avoid considering natural disasters.

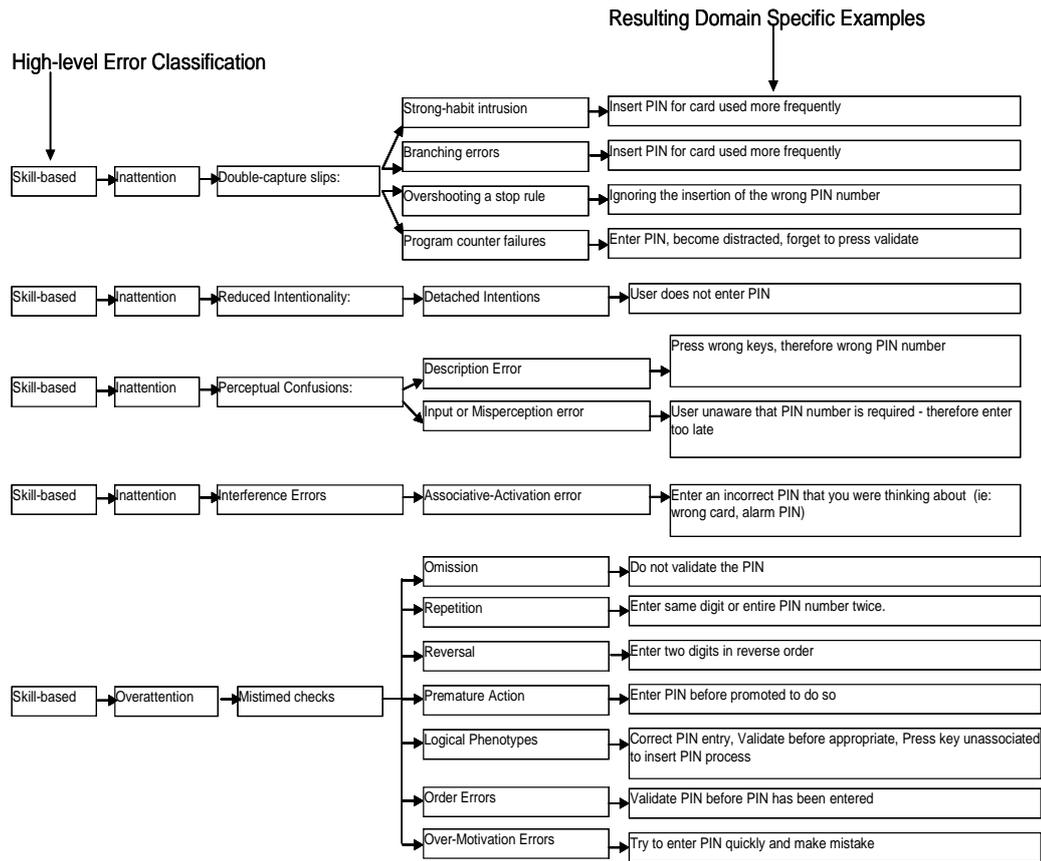


Figure 4-6. Results of the systematic analysis of possible deviations while inserting a pin at an ATM based on a CTT task analysis model and the skill-based HERT

4.1.2 Including model patterns for error tolerance in task models

We suggest that model patterns can be used as a means of incorporating erroneous behaviour into a standard task model as a method for making explicit such errors. We produce our model patterns using the results of the systematic error analysis of each subtask. In this case, the “insert PIN” subtask. This is achieved by first grouping results that are the same and then by grouping results that would have the same impact on the system. For example, ‘Insert PIN for card used more frequently’ and ‘Press wrong keys, therefore wrong PIN number’ can both be considered as entering the wrong PIN. The grouping procedure for the ATM example is presented in Table 4-1.

Table 4-1. Grouping ATM Human Error Analysis Results

Domain specific example	System Impact
Insert PIN for card used more frequently	Incorrect PIN
Ignoring insertion of wrong PIN number	Incorrect PIN
Enter PIN, get distracted, forget to validate	Too long entering PIN
User does not enter PIN	System timeout
Press wrong keys, therefore wrong PIN number	Incorrect PIN
User unaware that PIN is required, therefore enter too late	Too long entering PIN
Enter an incorrect PIN user was thinking about (i.e. wrong card, alarm PIN)	Incorrect PIN
Do not validate the PIN	Too long entering PIN
Enter same digit or entire PIN twice	Incorrect PIN
Enter two digits in reverse order	Incorrect PIN
Enter PIN before prompted to do so	Incorrect PIN
Correct PIN entry, validate before appropriate, press key unassociated to insert PIN process	Too long entering PIN
Validate PIN before PIN has been entered	Too long entering PIN
Try to enter PIN quickly and make mistake	Incorrect PIN

4.1.3 Error modelling

The errors grouped in Table 4-1 are now considered in combination. For example, the user could first enter the PIN incorrectly, and then enter it correctly, which means the user’s goal of identifying one’s self is still attainable. We graphically represent and provide textual descriptions of the ATM errors using the CTTe icons. This simplifies the explanation of later plugging the model patterns into the task models. This part of the procedure corresponds to label 3 in Figure 4-3 of the generic integrated modelling framework.

We have determined that with a “standard” cash machine (note that when the same procedure is applied to the case studies of this thesis, there are no ambiguities of the system behaviour since we have modelled the systems ourselves. Here we simply assume the behaviour of an average cash machine), there are four possibilities of interaction when entering a PIN. These four possibilities are represented as the first, second, third and fourth in Figure 4-7. They are abstract icons, which mean each of the subtasks of the Insert PIN task will contain subtasks (illustrated with dashed arrows).

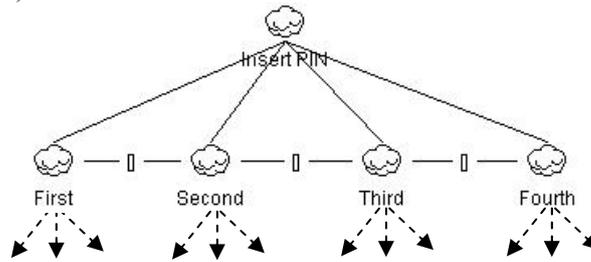


Figure 4-7. Four possibilities of interaction when entering PIN

The abstract tasks (first, second, third and fourth) consist of combinations of five errors models (in relation to system impact) labelled P1, P2, P3, P4 and P5 which have been devised based on the error analysis results. These error models can be seen in Figure 4-8 - Figure 4-10.

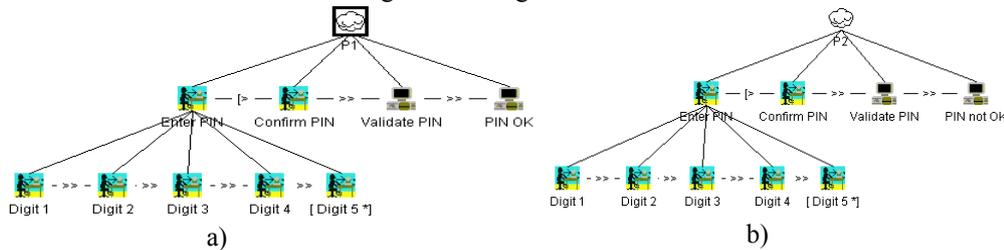


Figure 4-8. a) p1: PIN OK, b) p2: PIN not ok

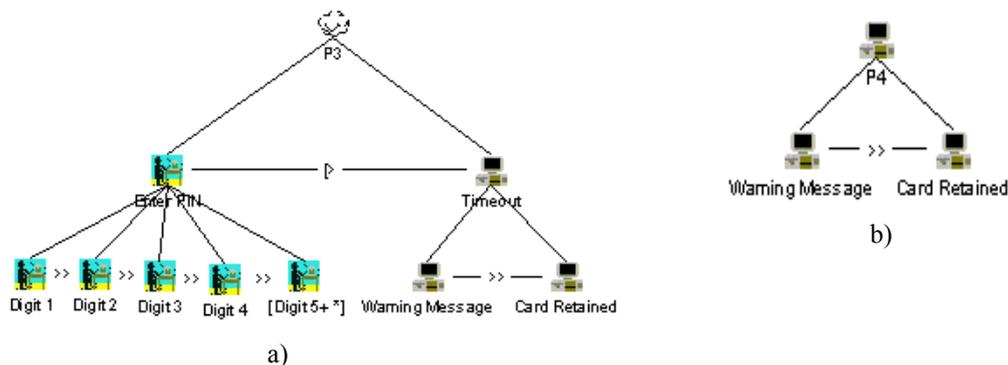


Figure 4-9. a) p3 Too long entering PIN, b) p4: Simple Timeout

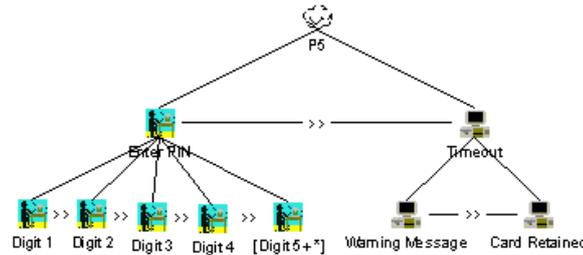


Figure 4-10. p5: Timeout on PIN confirmation

Referring back to Figure 4-7, the first of four possible interactions is what we will call the first model pattern for error tolerance. Model pattern 1 (modelled abstractly in Figure 4-11, i.e not showing the full leaves of the tree) can be defined textually as:

1. The PIN is correct on the 1st try or,
2. The user takes too long to enter the PIN and the system times out or,
3. The user does not enter a PIN at all and the system times out or,
4. The user takes too long confirming the PIN and the system times out

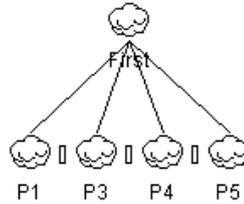


Figure 4-11. Model pattern 1, showing abstract tasks only

The complete model pattern 1 is depicted in Figure 4-12. It is clear, that even for a rudimentary task, such as entering a PIN, and considering only Skill-based errors, the models can become large and complex. For this reason, we anticipate that certain patterns will be reusable, such as PIN entry patterns while others will require customisation.

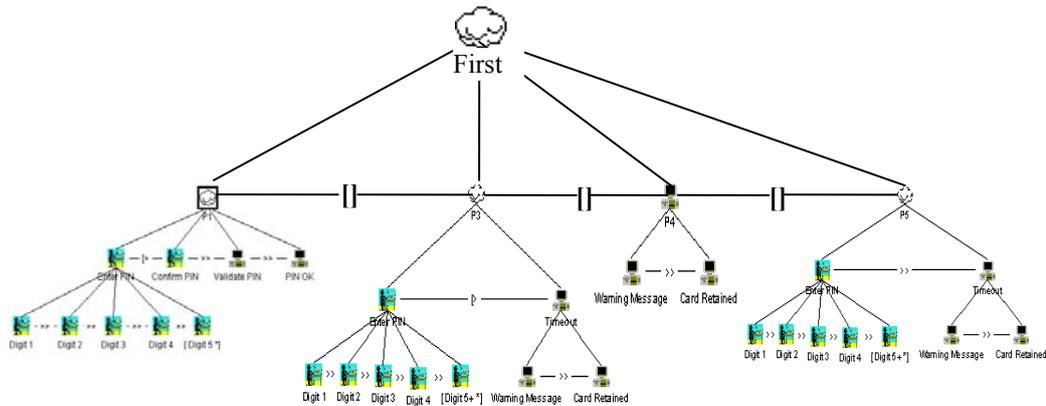


Figure 4-12. Model pattern 1, showing all leaves

The second possibility of interaction (refer to Figure 4-7) can be defined textually as:

1. The PIN is incorrect on the 1st try and correct on the 2nd try or,
2. The PIN is incorrect on the 1st try and the user takes too long to enter the PIN on the 2nd try therefore the system times out or,
3. The PIN is incorrect on the 1st try and the user does not enter a PIN at all on the 2nd try and therefore system times out or,
4. The PIN is incorrect on the 1st try and the user takes too long confirming the PIN on the 2nd try and the system times out

The abstract model of this potential interaction, model pattern 2 for error tolerance, is presented in Figure 4-13. The full model pattern, including all error models, is presented in Figure 4-16a.

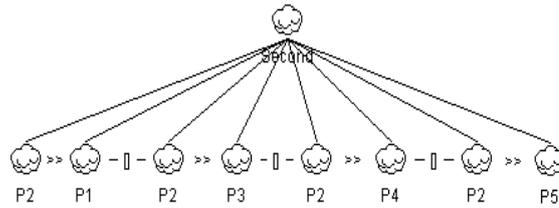


Figure 4-13. Model pattern 2, showing abstract tasks only

The third of four possible erroneous interactions with the ATM according to the error analyse is textually represented as:

1. The PIN is incorrect on the 1st try, incorrect on the 2nd try and correct on the 3rd try or
2. The PIN is incorrect on the 1st try, incorrect on the 2nd try and the user takes too long to enter the PIN on the 3rd try therefore the system times out or
3. The PIN is incorrect on the 1st try, incorrect on the 2nd try and the user does not enter a PIN at all on the 3rd try and therefore system times out or
4. The PIN is incorrect on the 1st try, incorrect on the 2nd try and the user takes too long confirming the PIN on the 3rd try and the system times out.

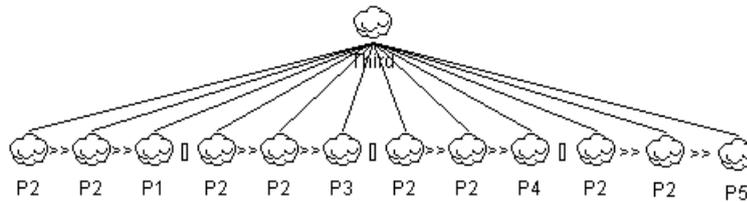


Figure 4-14. Model pattern 3, showing abstract tasks only

Please not, the complete model pattern 3, including all leaves of error models is not presented as it would be illegible.

For the fourth and final possibility of interaction, model pattern 4 is defined as:

1. The PIN is incorrect on the 1st try, incorrect on the 2nd try and incorrect on the 3rd try

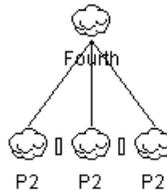


Figure 4-15. Model pattern 4, showing abstract tasks only

The complete model pattern 4, including all leaves is presented in Figure 4-16b.

It can be seen from previous models that taking into account possible erroneous behaviour increases significantly the size of the task models. For instance there are 21 leaves in the full task model for normal user behaviour (some of which is represented in Figure 4-5) and 234 leaves in the task model describing both erroneous and normal user behaviour (merging of models illustrated in Figure 4-8 to Figure 4-15).

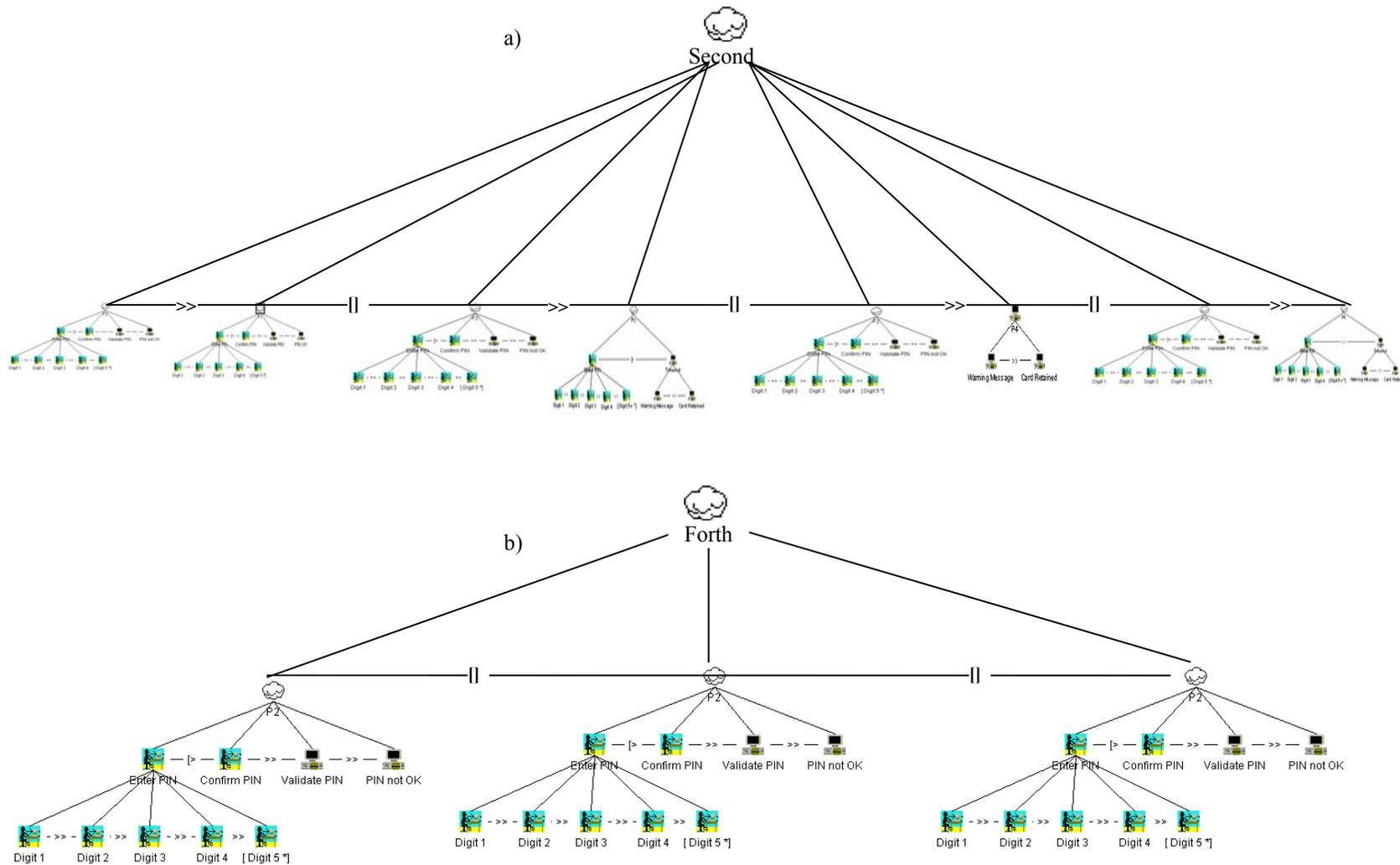


Figure 4-16. a) model pattern 2, showing all leaves, b) model pattern 3, showing all leaves

When errors are introduced in a task model (thus making it an error-laden model), some generated scenarios can lead to a failure to reach the goal. It could be debated whether this can still be called a task model, since making errors is certainly not part of the user's goal. Regarding Hollnagel's approach (Hollnagel 1993b), the error-free and error-laden models concern the error's phenotype, i.e. its observable manifestations. The informal description is especially valuable to describe the error's genotype (psychological process that led to the error).

4.2 “Plugging in” model patterns for error tolerance

The final stage of accounting for erroneous user behaviour in the task model is to “plug in” the model patterns in the place of the subtask being analysed. This relates to label 4 of Figure 4-3 of the safety-related user behaviour part of the task modelling phase of the generic integrated modelling framework. For illustrative purposes, we provide a diagram to give the reader an idea of this procedure and to illustrate how the model suddenly becomes complex, though there is currently no tool support for the process.

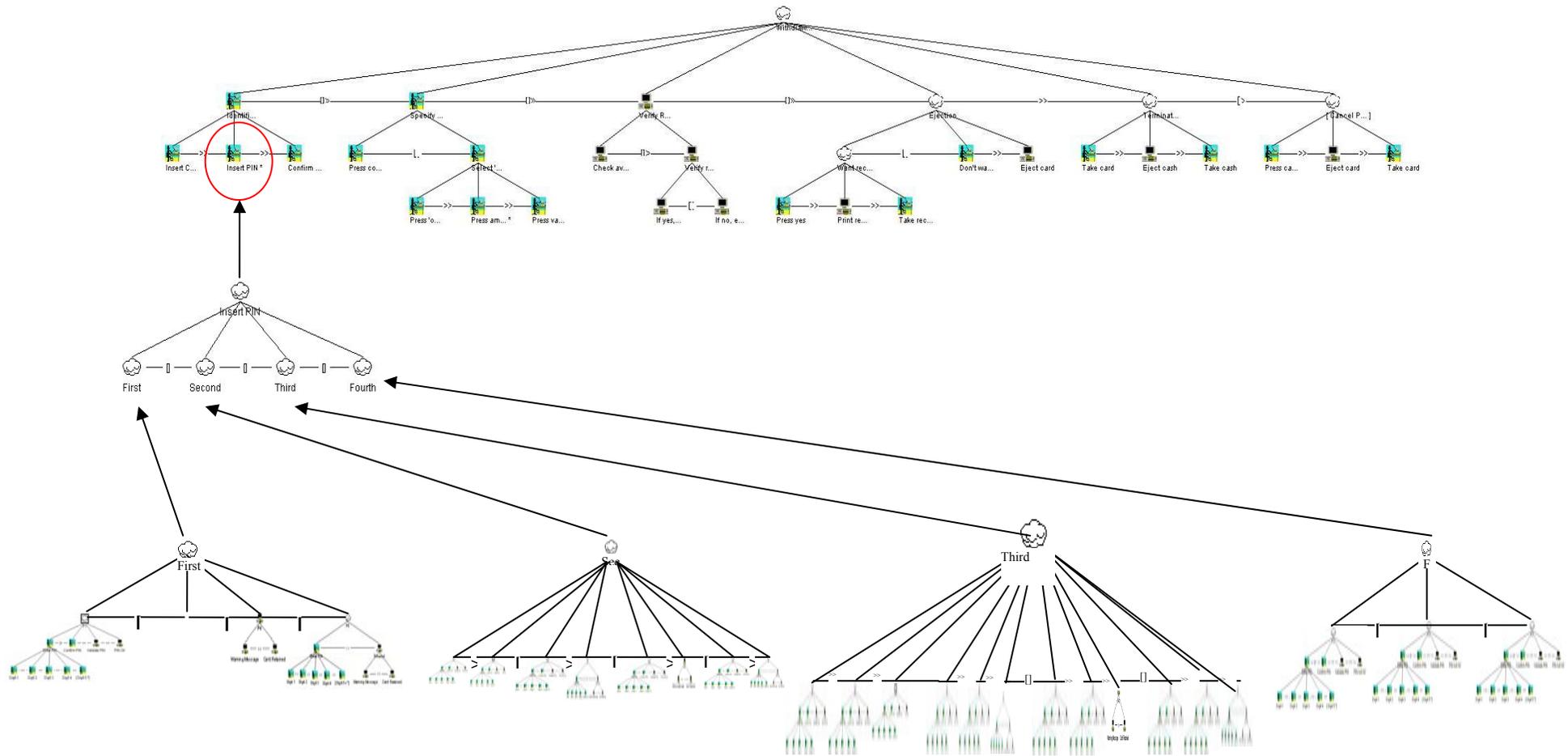


Figure 4-17. Illustration of “plugging in” model pattern for error tolerance of “insert PIN” subtask

5 Supporting Integration

In this section, we discuss how the phase presented in this chapter, Task Modelling and Error Management is integrated within the generic integrated modelling framework. Full comprehension of this will require further reading of corresponding chapters. The phase diagram, presented in Figure 4-18 includes annotations referring to sections and diagrams within this chapter to show where the processes have been presented and discussed. The framework supports the designer by advocating that the process of design or redesign can commence from any of the phases. Thus the task modelling phase presented in this chapter may not necessarily be the first performed. Once the task modelling phase has been performed, it is connected to the quantitative evaluation phase. Though this phase is not a key contribution to the thesis, it is a necessary part of the framework ensuring that the tasks modelled are realistically achievable by its intended end users. For example, in (Palanque and Bastide 1997) it was demonstrated that assessing the complexity of a task based on end user capabilities could influence the system model by potentially making the system more simple. Furthermore, Lacaze et al., (Lacaze et al. 2002) applied Fitt's Law (Fitts 1954) to a system model to evaluate potential user performance.

Depending on the current state of development (if there is a system model or not), the process can continue to the testing phase, which comprises of system model testing for validation and verification purposes, as well as system usability testing, keeping inline with User Centred Design (UCD) approaches. So far, we have discussed the general flow of information from the task modelling phase. This task modelling phase also shared particular information with other phases. The human error analysis part of the task modelling phase may (if appropriate/necessary) receive information from the safety modelling phase. Task analysis can be used to feed into the interaction hazard analysis of the system modelling phase for barrier analysis and modelling. Furthermore the task modelling may feed directly into the usability phase to ensure a system is able to handle identified user tasks including erroneous behaviour. The research presented in this chapter on task modelling and error management has been published and can be found in (Palanque and Basnyat 2004) and (Bastide and Basnyat 2006).

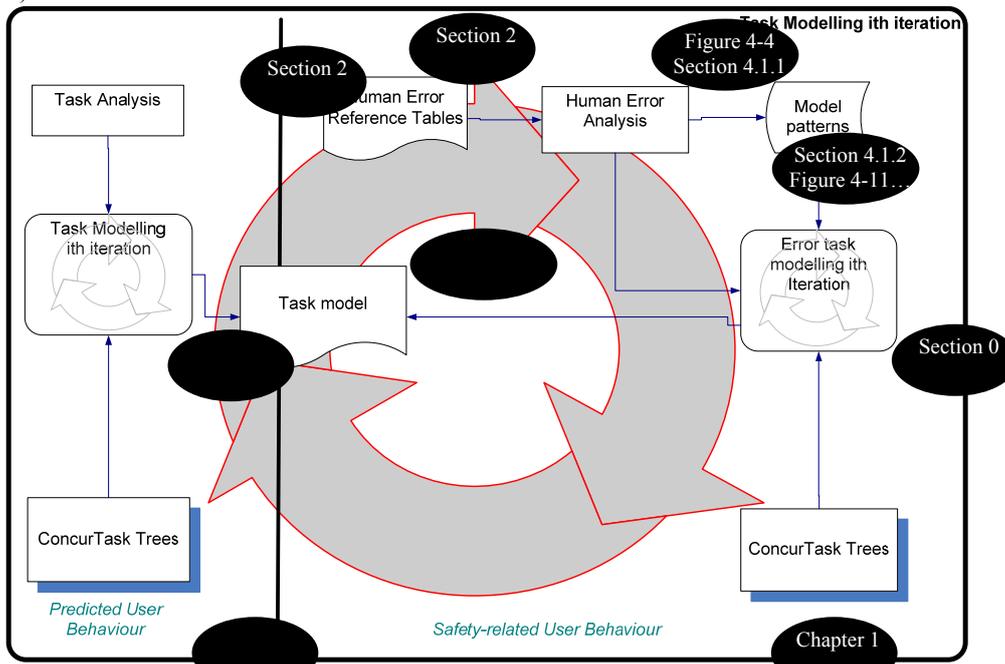


Figure 4-18. Task modelling phase of the generic integrated modelling framework

6 Conclusion

In this chapter we have presented a systematic way of taking into account erroneous user behaviour in task modelling. This work builds upon previous work in the field of task analysis, task modelling, human error analysis and identification. We have proposed the definition and use of model patterns for dealing with complexity and repetitions that frequently appear when modelling erroneous user behaviours. We have shown on a simple case study how task patterns have been identified and how we have modelled them using CTT notation and its related tool CTTE. Due to their intrinsic nature, patterns are good candidates for known and previously encountered problems. This is the reason why they have been successfully exploited on the ATM case study.

The notion of model patterns for error tolerance have been given for use in the process of transforming a standard task model to a more complex one, making domain specific human errors explicit that could also be applied to other domains. We have given a definition of this concept in terms of model transformations. We are in the process of building a collection of these patterns by investigating the literature on human errors, basing our work on the HERTs classification.

Our model patterns are designed to put human factors knowledge about human errors in a form easily usable in a model-driven systems development process. Model patterns are put into use after an initial error-free task model has been built for the system under design. It is then the human factors specialist's task to investigate which errors are likely to occur at different locations in the task tree. The application of the corresponding model pattern will produce a transformed task model that exhibits this specific error.

It is intended that the idea of identifying human errors early in the development process will improve the likelihood of designing an error-tolerant system. By exploiting these HERTs in a proactive manner we are also able to identify further errors that should be taken into account for future designs of a similar system.

The key benefit of this human "error" analysis however, is that analysts are guided in their investigation by the structure of operator tasks that are embedded in the task notation. A further benefit of this style of analysis is that not only does it help to identify potential interaction problems that led to a particular incident, it can also be used to identify other alternate problems that might jeopardise the future safety of an application but which did not arise in this specific accident.

This form of analysis can help to identify a large potential set of interaction problems even for relatively simple sets of tasks. It is, therefore, often necessary to continue the analysis by grouping potential interaction problems into more generic types.

Future chapters will show how the CTT task models, including model patterns for error tolerance can be related to formal models of the system's behaviour based on Petri nets, using the ICO notation. A practical interest is that usage scenarios can be automatically computed from CTT task models (in our case, error-laden models), and one can further investigate how the system reacts to these error-laden scenarios.

Though we have provided a method for systematically identifying areas for potential cognitive human "errors", further levels of potential failures that may cause adverse outcomes may also exist which have not been considered. These include communication failures (between humans and/or systems), coordination and cooperation failures. Our method and techniques for analysing human errors, producing model patterns and applying them also indicates a strong need for dedicated tool support.

Chapter 5

Interactive Systems Modelling & Error Management

*“Everything should be made as simple as possible, but not simpler. The important thing is not to stop questioning. Curiosity has its own reason for existing”
(Albert Einstein, 1879 - 1955)*

Chapter Summary

Previous chapters have argued the advantages of formal specification techniques for interactive systems modelling. Though the contribution in this thesis is not directly the modelling of such systems, we must first present how to model interactive systems before illustrating how to account for human “errors” and erroneous events in the formal specification of an interactive application. In order to discuss our approach on task model and system model consistency, and other aspects of the proposed generic integrated modelling framework such as barrier analysis and modelling, it is important for us to demonstrate how to apply the proposed system modelling techniques. This chapter provides a discussion and illustrative example for the application of the Interactive Cooperative Objects (ICO) formalism on an ATM cash machine. The aim of this chapter is to show that the ICO formalism is capable of supporting the integration of human “errors” in the formal specification of an interactive system.

1 Introduction

We have seen in the literature review that though a lot of research has focused on the formal specification of user interfaces and applications, few explain the practical advantages and uses of formal modelling of errors in the design of interactive applications.

We believe safety-critical interactive systems should be designed for error avoidance, error detection and error recovery. Our approach tackles these properties. We include error avoidance (not all possible errors) by incorporating known human “errors” into the formal specification of a system. Additionally, we include protection barriers in the system model as a further means to avoid known errors (this is presented in the subsequent chapter). Our approach also provides a means for detecting errors, though these errors must be made explicit within the system model. For example, if the system state changes from a safe state to a hazardous state, the model could trigger alarms. We also consider error recovery, again for errors that have been explicitly modelled. This can take a number of forms, for example returning the system to a safe state following a hazardous event. Returning to a safe state may involve triggering events that automatically shutdown a system, or reinitialise a system, or it could take form of undo/redo/cancel features.

To model systems for this research, we use the Interactive Cooperative Objects (ICO) (Navarre 2001) formalism, an object-oriented, Petri net-based formalism dedicated to the modelling and construction of highly interactive distributed systems. We have a tool, PetShop (Navarre et al. 2003) dedicated to editing, simulating and analysing ICOs specifications.

Three error things

We discuss conflicts between applying usability principles and meeting safety and security requirements. We show that safety requirements may induce restrictions on human error tolerance, for example by imposing timing constraints on users or restricting the number of authorised attempts at PIN entry.

We illustrate how to associate the results of the HERTs analysis on a subtask of the ATM task analysis in the previous chapter to modelling erroneous events in the system model in this chapter.

The presentation of the modelling of erroneous events is organised according to three types of errors, data entry errors, sequencing errors and timing errors. In addition to these, we also present modelling techniques to make explicit PIN entry for the wrong card and post-completion errors.

This approach and techniques presented in this chapter relate to the functional behaviour and part of the safety-related behaviour of the system modelling phase of the generic integrated modelling framework. Following on from this, the subsequent chapter presents our approach to barrier analysis and modelling for safety-critical interactive systems to complete the explanation of the system modelling phase.

The aim of this chapter is to show that the ICO formalism is capable of supporting the integration of human “errors” and erroneous events in the formal specification of an interactive system.

2 Modelling Interactive Systems on the ATM example

We present in this Section 2, the formal specification of an ATM model in order to illustrate that the capabilities of ICOs are sufficient for our needs. Another aim is to present various modelling aspects related to the design of interactive systems. We discuss issues relating to the design of interactive systems with respect to HCI principles, such as user centred design, HCI design guidelines, usability as well as safety and security issues. The discussions will highlight, that there are often constraints when attempting to satisfy both HCI principles and safety issues. The first model presented in this section uses basic Petri nets to represent the behaviour of an ATM. It illustrates two modelling features that impact HCI. The model is mostly sequential (i.e. leaving little choice for user interaction) though includes optional behaviour, for example when deciding whether or not a receipt is required. The aim here is to illustrate a basic system model and then discuss why in our opinion it is under-specified, particularly in the field of safety-critical interactive systems.

Referring back to the literature review, disadvantages in terms of using Petri nets for accident analysis were found (Kontogiannis et al. 2000) because they become “unwieldy” for complex scenarios and that tools should be improved to focus on providing capabilities for multiple levels of represented. We show here how our supporting CASE tool, Petshop can be used for the edition, design and modelling of Petri nets for ‘real life’ large systems.

With respect to the generic integrated modelling framework, the modelling of systems is represented on the left hand side of the systems modelling phase, presented in Figure 5-1. This side of the phase refers to the functional behaviour of the system.

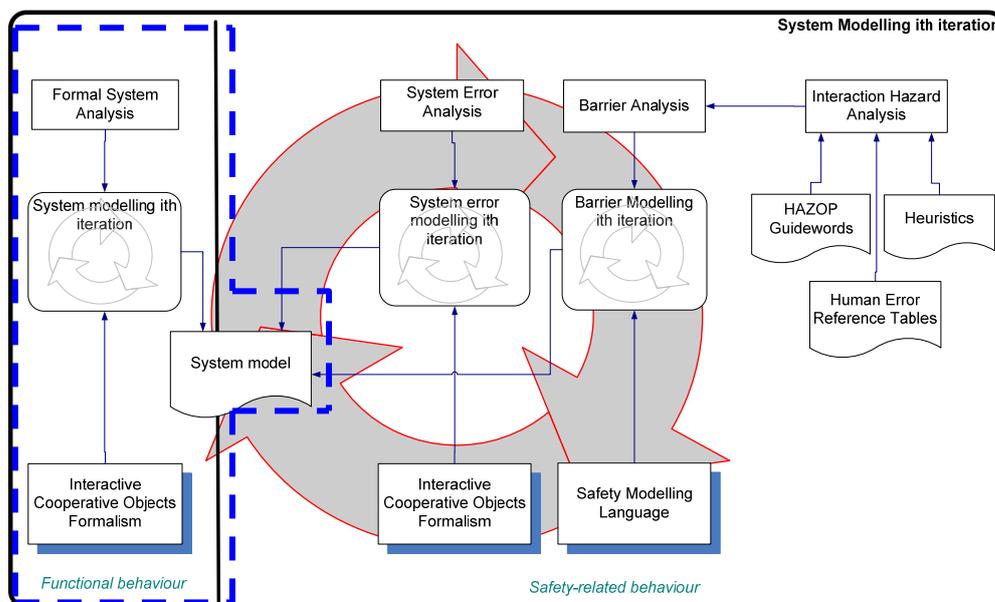


Figure 5-1. System modelling phase of the generic integrated modelling framework highlighting the functional behaviour

2.1 Behaviour

We present in this section an ICO model of a hypothetical ATM machine. It enforces restrictions on the order in which tasks can be carried out and therefore meets few HCI principles. The model has no allowance for human “errors”, provides no undo/redo/cancel function and gives the user little freedom of choice except for the choice of receipt and order of in which to take notes or the receipt.

An informal reading will be provided for the nets introduced in this chapter. In the following analysis, we use the following symbols:

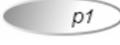
- States are represented by the distribution of tokens into places 
- Actions triggered in an autonomous way by the system are represented as  and called transitions
- Actions triggered by users are represented by half bordered transition 

Figure 5-2 presents a basic system model representing the behaviour of a hypothetical ATM. Figure 5-3 illustrates a rudimentary mock-up of the interface corresponding to the basic system model. The interface contains interactive buttons, such as Insert Card, With Receipt, Without Receipt etc. These interactive buttons correspond to user triggered events (half bordered transitions) on the system model. Of course with most real life ATMs, buttons such as Insert Card do not exist as the action of inserting one’s card into the machine triggers the system to go into the next state of having received the card.

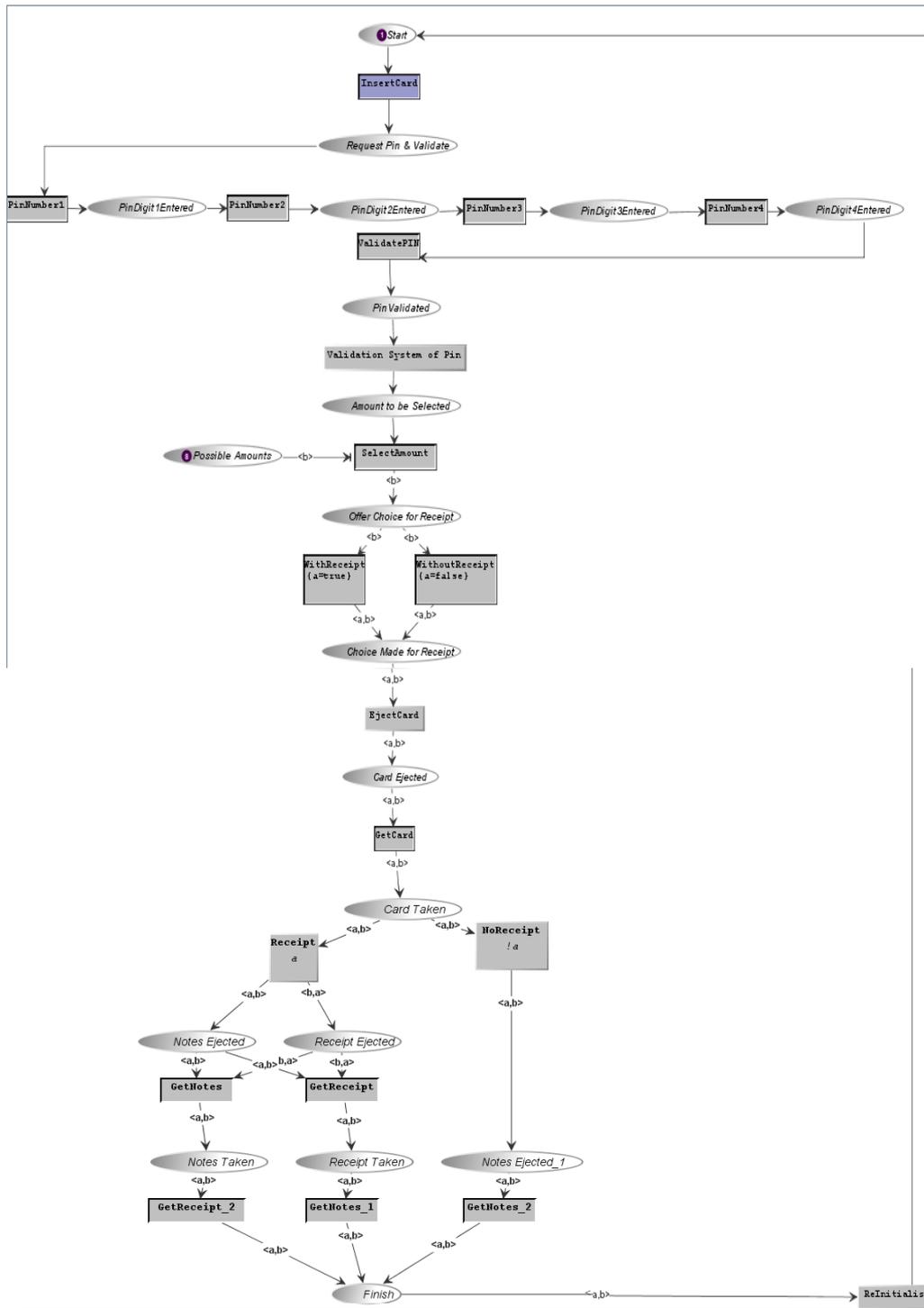


Figure 5-2. Basic ATM ICO System Model

The initial state of the Petri net has one token in place *Start* and 8 tokens in place *Possible Amounts*. The only fireable transition is *InsertCard*. When the *InsertCard* transition is fired, the *Request Pin and Validate* place receives a token, allowing the four pin number entry transitions to be fired consecutively. Until the pin has been validated, all transitions are user triggered events (or events requiring some action on the user interface). The *validation system of Pin* transition is a system triggered event, an autonomous event, not requiring human interaction. The behaviour of the system continues in a similar fashion, allowing only one transition to be fireable at a time. The *SelectAmount* transition makes a unification with one of the tokens of place *Possible Amounts* and sets a token in place *Offer choice of receipt*. *Possible Amounts* always contains 8 tokens each holding a value representing the amount of cash that can be withdrawn using the system. For example, token 1 = 20, token 2 = 40...token 8 = 160. This integer variable is represented as $\langle b \rangle$ in the system model and is carried through transitions until the end of the transaction. The *Offer Choice For Receipt* place makes both transitions

WithReceipt and *WithoutReceipt* fireable. These two transitions contain preconditions, true for *WithReceipt* and false for *WithoutReceipt*. This boolean variable is represented as $\langle a \rangle$ in the system model. The user's choice with respect to the receipt affects the behaviour of the system model from the place *Card Taken*. From here, the system will fire either transition *Receipt* or transition *NoReceipt* depending on variable $\langle a \rangle$ it receives ($\langle a \rangle = \text{true}$ for *Receipt* to fire, $\langle a \rangle = \text{false}$ for *NoReceipt* to fire). The rest of the system model allows the user to either take the notes first or the receipt first and then the system reinitialises.

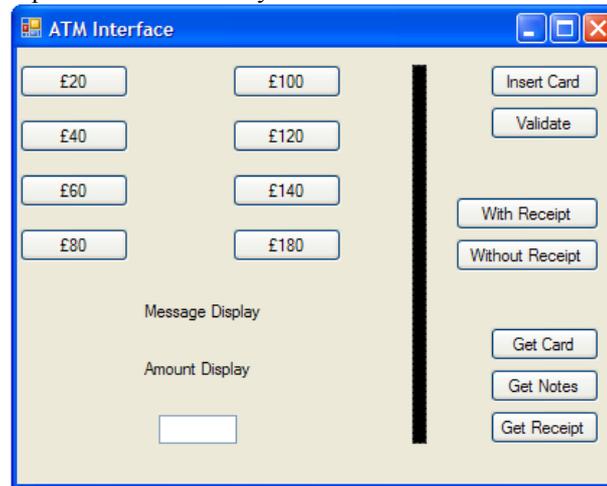


Figure 5-3. ATM Interface

What is interesting about the system model we have presented is that it offers little choice to the user since the model is mostly sequential. This design option goes against typical HCI guidelines. For example, one of Nielsen et al's (Nielsen and Molich 1990) heuristics is called "User control and freedom". Our system model offers little control to the user, except the choice of requiring a receipt, and offers little freedom of interaction. This means that the user also does not have, and in this case does not require undo/redo facilities. We can argue however, that providing too much freedom to the user makes the interaction susceptible to errors, such as post-completion errors for example. In the case of an ATM, leaving the card in the machine after withdrawing funds and taking the receipt can be considered a post-completion error. The error refers to the "clean up" task that is omitted at the end of an interaction when the user has achieved their goal (i.e. withdrawing cash). Though we could imagine that if the user never needed their card again, leaving the card in the cash machine would not be a post-completion error. Or perhaps, if the system operated such that the user leaves their card in the machine in preparation for future interaction with that system, then leaving the card in the machine would not be a post-completion error. However, most real life ATMs that we currently use are designed such that the user must take back their card before the cash and the receipt become available.

When considering such potential for errors, we can distinguish between those having physical system constraints, such as inserting the card, and those related to human constraints, remembering PIN numbers and other errors discussed in previous chapters. For example, the user putting the wrong card into the system for the current transaction is a user related problem and results in a PIN entry issue (since the PIN will be incorrect for the card that has been entered). While PIN entry alone is system-related erroneous behaviour, because it is system related security measures that impose PIN restrictions. Both types of potential erroneous behaviour must be considered and measures against them enforced. This illustrates the kinds of compromises that must be made between usability (i.e allowing an unlimited number of PIN entry attempts) and security (allowing one attempt only). Though the ATM case study relates to security, the thesis focuses particularly on safety. Here we illustrate typical design compromises and constraints.

2.2 Interaction

In this subsection, we employ the activation function table and the rendering function table, features of the ICO formalism, to describe the system model of the ATM. Before presenting these tables, Table 5-1 provides a list of user triggered behaviour and autonomous system behaviour. The main goal of the activation function and the rendering function is to make the bridge between the user interface and the behavioural model of the system (ObCS).

Table 5-1. User and System behaviour

User Triggered Behaviour (i.e. half bordered transitions)	Autonomous behaviour (non-bordered transitions)
InsertCard	Validation system of Pin (accepting PIN number)
PinNumber1	EjectCard
PinNumber2	Receipt (proceed process without providing receipt)
PinNumber3	NoReceipt (proceed process providing receipt)
PinNumber4	Reinitialise
ValidatePIN	
SelectAmount	
WithReceipt	
WithoutReceipt	
GetCard	
GetNotes	
GetReceipt	

2.2.1 Activation Function - Event-based

The activation function table summarises the event-based relation between interface widgets, such as the Insert Card button of the UI presented in Figure 5-3, user actions, in this case mostly clicking on the UI and user services, which refer to the half bordered transitions in the system model. The table shows how user actions on the UI change the system state. Table 5-2 presents the activation of the ATM. The first column represents the user interface element (interactor), the second column is the event corresponding to user action on the widget and the last column the user services and their related transition(s) in the ObCS.

Table 5-2. Activation Function for Figure 5-2

Widget	User Action	User Service
Insert Card Button	Insert the card	InsertCard
Pin Text Box	Press the key (key)	PinNumber1
Pin Text Box	Press the key (key)	PinNumber2
Pin Text Box	Press the key (key)	PinNumber3
Pin Text Box	Press the key (key)	PinNumber4
Validate Button	Click	ValidatePin
Amount Buttons (x)	Click (x)	SelectAmount (x)
With Receipt Button	Click	WithReceipt
Without Receipt Button	Click	WithoutReceipt
Get Card Button	Click	GetCard
Get Notes Button	Click	GetNotes
Get Receipt Button	Click	GetReceipt

The activation function table does not contain widgets “Amount Display Label” and “Message Display Label”. This is because these widgets are not directly affected by user actions. By this we mean that if a user tries to click or select a display area, it will have no impact on the system behaviour or current system state.

2.2.2 Rendering Function

The rendering function table presents state-based information. The table relates places to events (i.e tokens entering and leaving places) and rendering methods for the UI (i.e label.text=“Welcome”). Table 5-3 corresponds to the rendering function i.e. how internal state of the ObCS is presented to the user. The first column contains the name of the place in the ObCS, the second column describes the state change (token removed from a place or entered a place). The last column presents the rendering method triggered when the state change occurs.

Table 5-3. Rendering Function for Figure 5-2

Place	Event	Rendering Method
Start	Token entered	Label : text = “Welcome”
Request Pin & Validate	Token entered	Message Display Label : text = “Please enter your PIN and validate”
PinDigit1Entered	Token entered	Pin Text Box : text = “*”

PinDigit2Entered	Token entered	Pin Text Box : text = “***”
PinDigit3Entered	Token entered	Pin Text Box : text = “****”
PinDigit4Entered	Token entered	Pin Text Box : text = “*****”
Pin Validated	Token entered	Message Display Label : text = “Authorised” Pin Text Box : text = “”
Amount to be Selected	Token entered	Message Display Label : text = “Please select amount you wish to withdraw”
Offer Choice for Receipt	Token entered 	Message Display Label : text = “Would you like a receipt?” Amount Display Label : text = b
Choice Made for Receipt	Token entered <a,b>	If a then Message Display Label : text = “Please wait for card, cash and receipt” else “Please wait for card and notes”
Card Ejected	Token entered <a,b>	Message Display Label : text = “Please take your card”
Card Taken	Token entered <a,b>	If a then Message Display Label : text = “Please wait for receipt and notes” else “Please wait for notes”
Notes Ejected	Token entered <a,b>	If a then Message Display Label : text = “Please take notes and wait for receipt” else “Please take notes”
Receipt Ejected	Token entered <a,b>	Message Display Label : text = “Please take receipt and wait for notes”
Notes Taken	Token entered <a,b>	Message Display Label : text = “Thank you for using simple ATM”
Receipt Taken	Token entered <a,b>	Message Display Label : text = “Thank you for using simple ATM”
Notes Ejected_1	Token entered <a,b>	Message Display Label : text = “Please take notes”
Finish	Token entered <a,b>	Message Display Label : text = “Thank you for using simple ATM”

3 Error Tolerance

The model of a hypothetical ATM presented in the previous section was restrictive with respect to user control and freedom and did not allow for any potential erroneous human behaviour. For example, the system model did not allow the user to make a mistake on PIN entry. It assumed that each of the four PIN digits would be entered correctly before proceeding to the validation of the PIN. We can argue that the model presented was too loosely specified. Perhaps not for a cash machine, though if this approach is applied to a safety-critical interactive system, then additional information must be taken into account. In this section we consider some paths of erroneous interaction that could trigger system failures if not taken into account in the design. We present approaches for changing the system model to manage potential errors by extending the ATM model to tolerate such errors.

As we have seen in Chapter 2 section 3.5 and Chapter 4, there are many possible human related ‘errors’. In this section, we limit those that we present and use for showing the capabilities of the ICO notation and system model adaptation. We take three types of errors, data entry ‘errors’, sequencing ‘errors’ and timing ‘errors’. Additionally, we demonstrate how to formally account for entering a PIN for the wrong card and avoiding post-completion errors. Errors such as those discussed in the task modelling and error management chapter and in literature review can also be fully accounted for in the system model. However, we must be careful when integrating different kinds of information in the model-based design framework so as to not incorporate everything into the system model. As we discussed in Chapter 3, many design models exist suitable for representing for various types of data. Certain models will be more suitable for certain types of error data. We believe that user-related errors, not related directly to the UI, rather more generic errors, such as cognitive errors should be applied to the task model for system model validation. Human related errors that involve interaction could also be represented in the task model once a firm idea of the application under development is available in which case the errors can also be represented in the system model. In any case, data represented in the task model will strongly influences the system model.

3.1 Relating HERTs Task Analysis to System Modelling & Error Management

Before presenting the modelling of erroneous behaviour, we have reproduced the results of the skill-based error analysis of the PIN entry subtask of the ATM task analysis presented in Chapter 4 here in Table 5-4. We do not explicitly handle each of these errors in our extended ATM system model.

Table 5-4 provides a column entitled “Modelling Impact” which informally details if and how each error is taken into account in our extended ATM system model. In addition to these skill-based PIN entry errors, we also demonstrate how to address timing errors and post-completion errors.

Table 5-4. Modelling Impact of Skill-based PIN Entry Errors

Skill-based error type	Erroneous PIN entry task	Modelling Impact (Figure 5-5)	
		Modelled?	How?
Strong-habit intrusion	Insert PIN for card used more frequently	Yes	2 x bank cards 2 x PIN numbers 3 authorised attempts
Branching errors	Insert PIN for card used more frequently	Yes	2 x bank cards 2 x PIN numbers 3 authorised attempts
Overshooting a stop rule	Ignoring the insertion of the wrong PIN number	Yes	3 authorised attempts
Program counter failure	Enter PIN, get distracted, forget to validate	No	Add timer to validation of PIN
Detached intentions	User does not enter PIN	No	Add timer to PIN entry
Description error	Press wrong keys, therefore enter wrong PIN	Yes	3 authorised attempts
Input or misperception error	User unaware that PIN is required, therefore enter too late	No	Add timer to PIN entry
Associative error	Enter incorrect PIN that user is thinking about	Yes	3 authorised attempts
Omission	Do not validate	No	Add timer to validation
Repetition	Enter same digit or enter PIN number twice	Yes	5 th PIN digit has no impact If other digit is incorrect, then 3 authorised attempts
Reversal	Enter two digits in reverse order	Yes	3 authorised attempts
Premature action	Enter PIN before prompted to do so	No	Hide keyboard/softkeys until system is ready to accept PIN
Logical phenotypes	Correct PIN entry, validate before appropriate. Press key unassociated to insert PIN process	No	Hide keyboard/softkeys until system is ready for PIN validation
Order errors	Validate PIN before PIN is entered	No	Hide keyboard/softkeys until system is ready for PIN validation
Over motivation errors	Try to enter PIN quickly and make mistake	Yes	3 authorised attempts

The following three subsections, entitled handling of data entry errors, handling of sequence errors and handling of timing errors propose extensions and modifications to the ATM system model in order to tolerate the above discussed erroneous events. We have designed a new system model, which explicitly handles data entry errors and timing errors. The model also includes the possibility of the user entering the PIN for the wrong card since the system model recognises two bank cards. This is an example of an error identified using the HERTs analysis. In addition to this, we present a basic model to illustrate how sequencing issues can be modelled.

All of these types of errors could also be modelled implicitly, by simple modification to the system model representing the fact that a data entry, or timing type error could occur while the system is in a particular state without explicitly modelling the behaviour of the error and how to tackle it.

Figure 5-4 illustrates on the system modelling phase of the generic integrated modelling framework where the approach and techniques presented in this section fit in the framework. Note that the complete approach of the system modelling phase is not presented in this chapter but in the subsequent chapter.

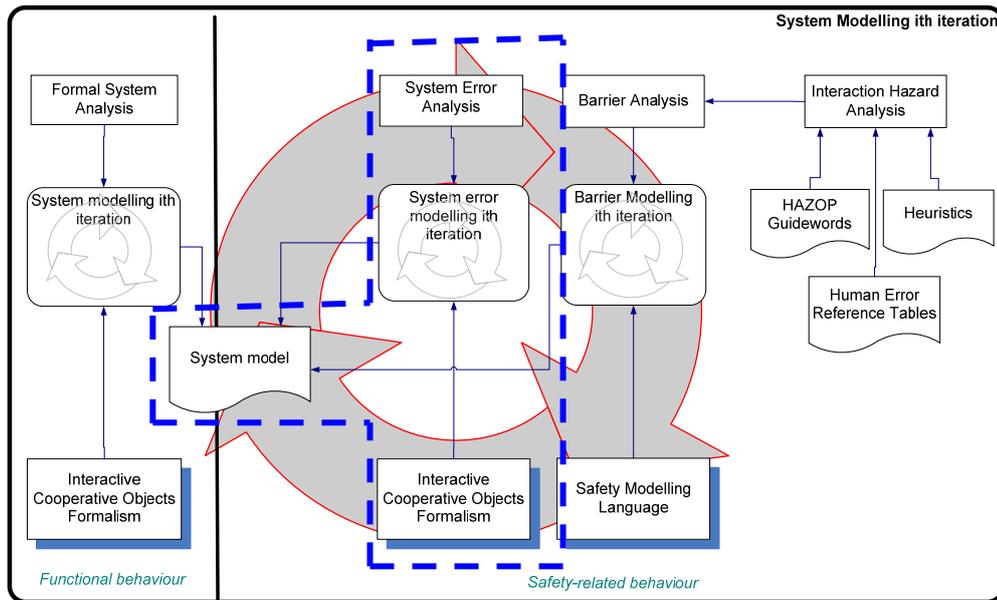


Figure 5-4. System modelling phase of the generic integrated modelling framework highlighting part of the safety-related behaviour

Figure 5-5 presents the complete ATM system model accounting for the above mentioned error types (except sequencing errors which are tackled separately for simplification). Since the model is not fully legible, we have segregated and labelled the model in order to explain relevant parts. Each part will be reproduced as a separate figure for explanatory purposes within the relevant subsection.

Before detailing the behaviour of the model, we informally describe the features of the redesigned ATM. The ATM keeps the card if the user enters three wrong PIN numbers. A PIN is considered wrong if and only if the PIN contains at least four digits and the first four digits do not correspond to the valid PIN. The PIN is correct if the PIN contains at least four digit and the first four digits correspond to the valid PIN (it is interesting to note that you can thus enter 5 digits and still have entered a PIN number considered as correct). The user can always cancel the transaction at any step, except after selecting the receipt option. The model accepts two cards, each with its own PIN. When the card is ejected from the machine, the system will retain the card if the user-triggered event of taking the card is not performed within 5 seconds.

3.2 Handling of data entry errors

The initial specification presented in the previous section is not lenient for any data entry errors (relating to the PIN or the amount of cash to be withdrawn). A more refined specification can make precise the detailed treatment of these types of errors. We provide in Figure 5-6 an extraction of the extended system model (label 1 in Figure 5-5) which explicitly deals with PIN entry errors. The system has been modified to allow three attempts before retaining the user's card (as with most real life cash machine).

Place $p3$ type is a 3-uplet $\langle i, a, cpin \rangle$. i represents the number of digits entered, a the number of attempts while $cpin$ represents the current value of the PIN. These values are updated each time a transition is fired. Starting from $p3$, there are four fireable transitions. These transitions are:

- $PinNo_T1$
- OK_T2
- $Cancel_T1$
- $Delete$ (the latter two are not visible in the extraction of the system model presented in Figure 5-6).

$PinNo_T1$, $PinNo_T2$ and $PinNo_T3$ are in mutual exclusion, this represents a key numbered. OK_T1 , OK_T2 are also in mutual exclusion, and represent the validation of the PIN. $Cancel_T1$ provides a cancellation of the transaction.

While i , the number of digits entered is less than 4, transition OK_T2 remains fireable. When i is equal to or more than 4, transitions OK_T1 becomes fireable (see preconditions on each of these two transitions).

If OK_T2 is fired, it means the user has tried to confirm the PIN with less than 4 digits. In this case, only two transitions are fireable, $Delete$ and $Cancel_T1$. $Cancel_T1$ enables the $GetCard_1$ transition to be fireable (not shown in the extraction of the system model presented in Figure 5-6), allowing the user to retrieve their card, while the $Delete$ transition is synchronised and fired when the Delete button is pressed. Transition $Delete$ sets i to 0 and $cpin$ to 0.

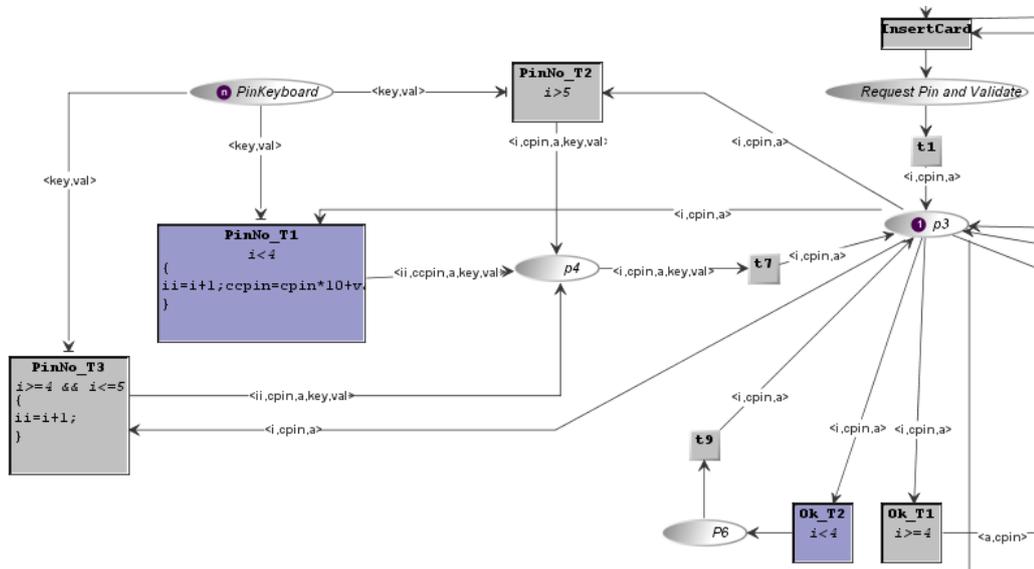


Figure 5-6. PIN Entry Part of the Extended System Model (label 1 in Figure 5-5)

If OK_T1 is fired, the user tries to confirm the PIN having entered at least 4 digits. If the PIN is incorrect with respect to the card in the machine (our model has two cards), transition $T10$, an autonomous transition fires, while if the PIN is correct, transition $T5$ fires. $T10$ is connected to the place $ValidPINs$, and places $CardUsed$. This transition checks to see whether the PIN that has been entered is a valid PIN with respect to the card that has been used. This part of the model is labelled 2a and 2b in Figure 5-5 and has been extracted and reproduced in Figure 5-7.

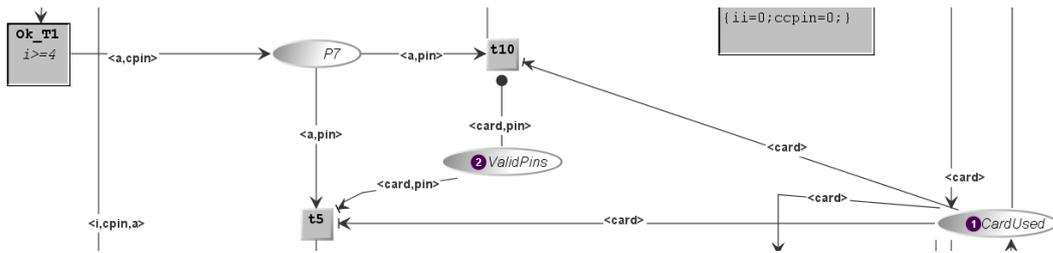


Figure 5-7. PIN Verification Part of the Extended System model (labels 2a and 2b in Figure 5-5)

If the PIN is invalid, the autonomous transition, *ErrorPin* fires. This process can occur up to three times (see precondition $a < 3$ on the transition). On the third incorrect attempt, transition *TooManyTries* fires placing a token in place *RetainedCards*. From here the system can be reinitialised though this time, only one card will be available (the card that was not retained). This is shown as label 3 in Figure 5-5).

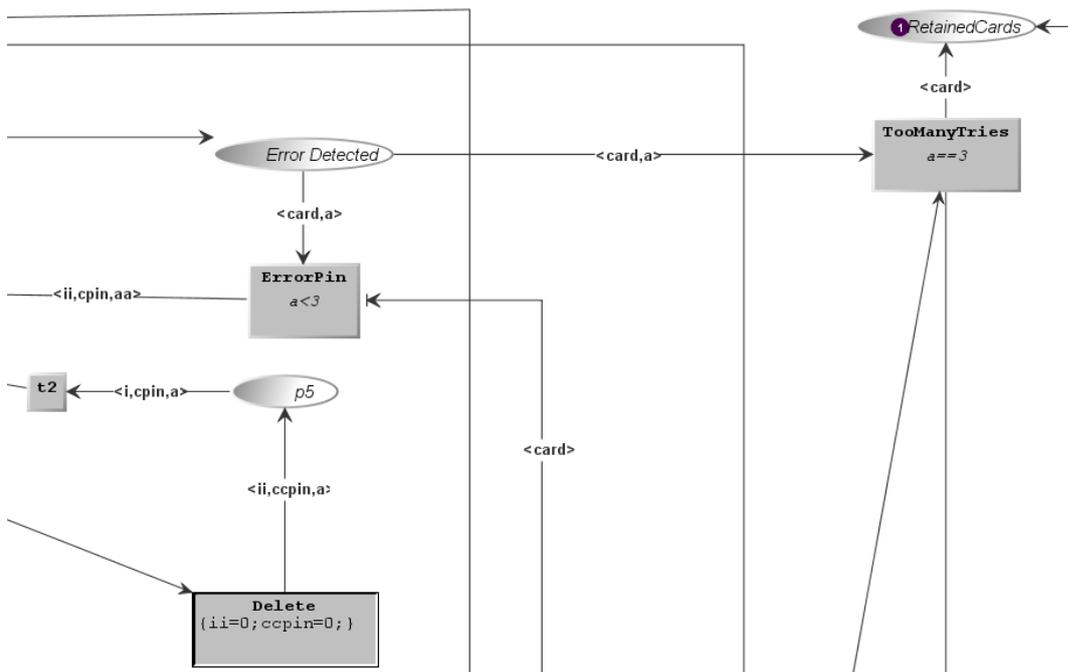


Figure 5-8. ErrorPin and TooManyTries Part of the Extended System model (label 3 in Figure 5-5)

When *T5* is fireable, and the PIN is correct, place *Amount To Be Selected* receives a token. From here, the process of selecting an amount (label 4 in Figure 5-5), deciding whether or not to take a receipt, the cash first or the card first is identical to the model described in the previous section (see Figure 5-2). Here, the only difference is the possibility to cancel the process at any time (label 5 in Figure 5-5). The three cancel transitions, *Cancel_T1*, *Cancel_T2*, *Cancel_T3* are presented in Figure 5-9.

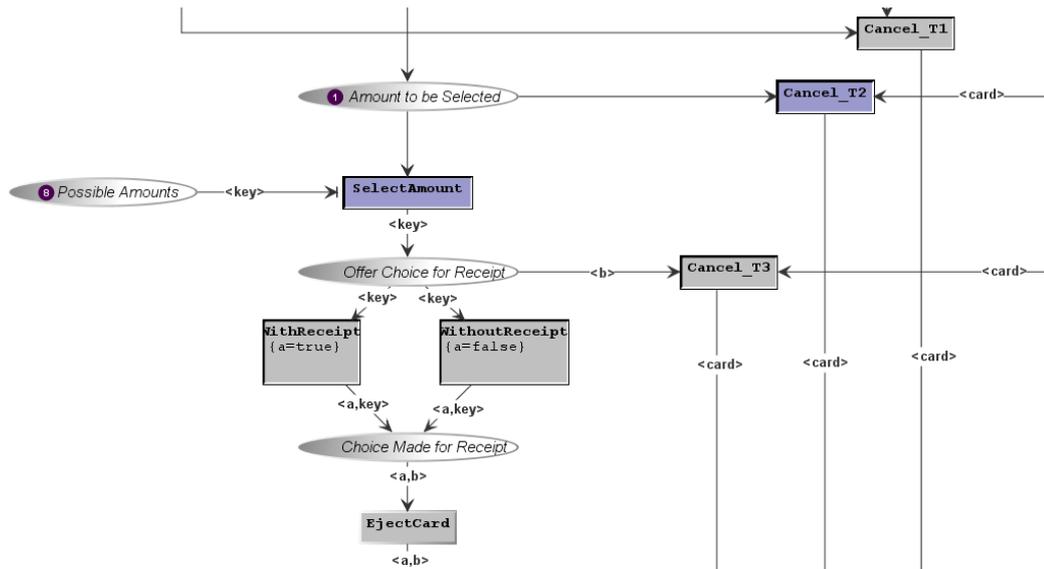


Figure 5-9. Cancel Feature Part of the Extended System model (label 5 in Figure 5-5)

Towards the end of the process, it is interesting to note that we have ensured that the card is returned to the user before the cash is ejected in order to avoid a post-completion error (Blandford and Connell 2000) (the last action to be performed is not the last action of the user’s goal which is to get cash).

The extensions to the system model presented for data entry error handling allow three attempts of the PIN number. However, the number of tries could have been modelled as unlimited or limited to just 1 attempt for example. The two abstract models presented in Figure 5-10 show how these can be modelled in a simple way. These models include PIN entry errors, but in an implicit way. Though we know an error occurs, we do not know what exactly, not how to tackle it. Whereas our previous models make explicit this behaviour and provide a means for dealing with PIN entry errors. We believe that abstractions of the explicit models for tasks such as PIN entry can be reused in higher-level specification system models.

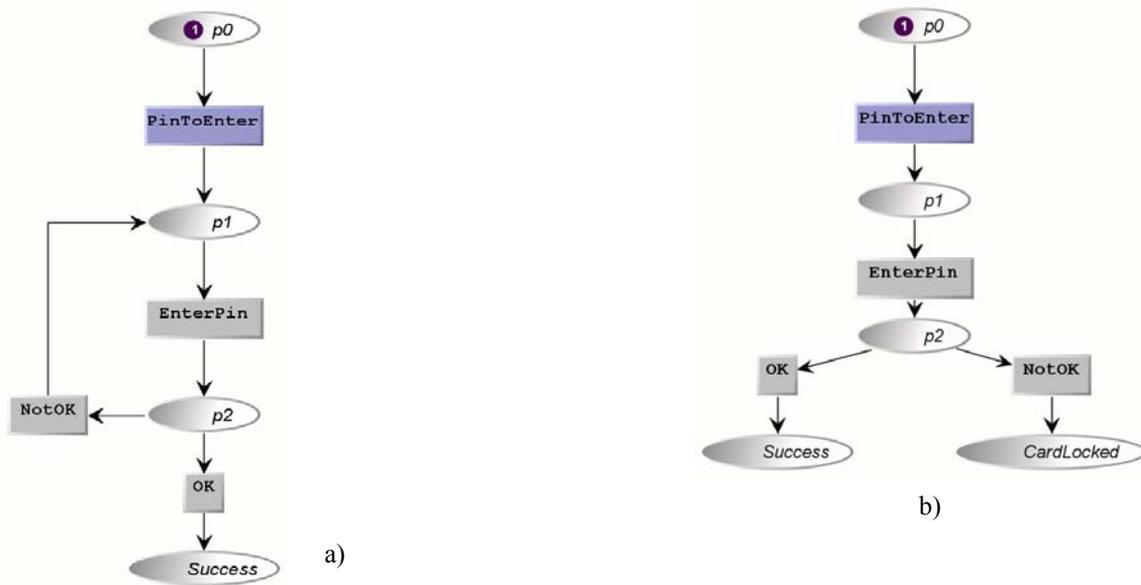


Figure 5-10. a) Unlimited PIN Entry Model, b) One Attempt PIN Entry Model

The place *ValidCards*, at the bottom of the Petri Net (label 6 in Figure 5-5) contains the list of valid cards (the two cards that are supported by the ATM). When this place is empty the model is “dead”.

3.3 Handling of sequence errors

Sequence errors can also be dealt with explicitly in the specification of a system model. When we refer to sequencing errors, we mean that users may try to interact with the system in an order other than the one permitted.

The models we have presented so far (Figure 5-2 and Figure 5-5) are restrictive in terms of user freedom. This can be interpreted in two ways, firstly that the model is so sequential the user will not be able to make a mistake, or, that it is so sequential, it restricts user interaction resulting in user frustration because there is a mismatch between the user’s mental model and the designer’s conceptual model. The user is then likely to attempt to interact in the way they believe is appropriate, and perhaps keep attempting until either they give up or understand system messages. As we know from our daily interaction experience, it is often hard to understand the system status and available options from a UI. This is one of the points of focus of HCI, to improve usability. One such approach, as suggested by Norman (Norman 1990) is to reduce the gulf of execution and the gulf of evaluation.

In contrast to our fairly restrictive models, we could design the system so that full freedom is provided to the user, to allow the interaction to occur in any order, giving a higher possibility that the user will achieve their goals by either trial and error or that the system behaves in the way they expect it to.

An example of a sequential interaction error is as follows. It may be the case that the user tries to enter their PIN before inserting their card into the machine. Although most ATMs currently available on the market do not operate in this manner, the design could be changed to handle the potential of sequencing errors. To impose certain sequencing requirements could involve both the hardware and software. For example if a system requires the card to be inserted before PIN entry or cash amount selection, the keypad of the machine could be made unavailable until the system has received the card. From a software perspective, widgets could be made invisible or inactive until a pre-required task has been achieved. Modelling sequencing behaviour involves using the interleaving approach to modelling.

One way of enhancing a system model that is too restrictive in terms of sequencing would be to impose interleaving behaviour. We can improve the system specification by adding concurrency, sequential properties and timing issues.

We present in Figure 5-11 a simple ATM model with maximum interleaving, i.e. no restriction on the order of interaction. When transition *Start* is fired, places *Card Entry*, *Request Pin and Validate*, *Pin Validation*, *Amount Selection*, *Receipt Choice 1*, *Receipt Choice 2*, *Card Return* and *Cash Ejection* receive a token. This means all subsequent transitions become fireable. Interleaving in Petri nets introduces parallelism. Due to the properties of Petri nets, when interleaving is imposed, more than one event can occur at the same time. Several transitions can be fireable at the any one time. For an ATM, this kind of behaviour is not critical. It would not be too serious if a user could select an amount of cash while also selecting whether a receipt is required. It would simply mean that many options and interactions would be available at the same time. Though this kind of behaviour is not a good idea for safety-critical interactive systems where users are usually highly trained and must follow strict procedures.

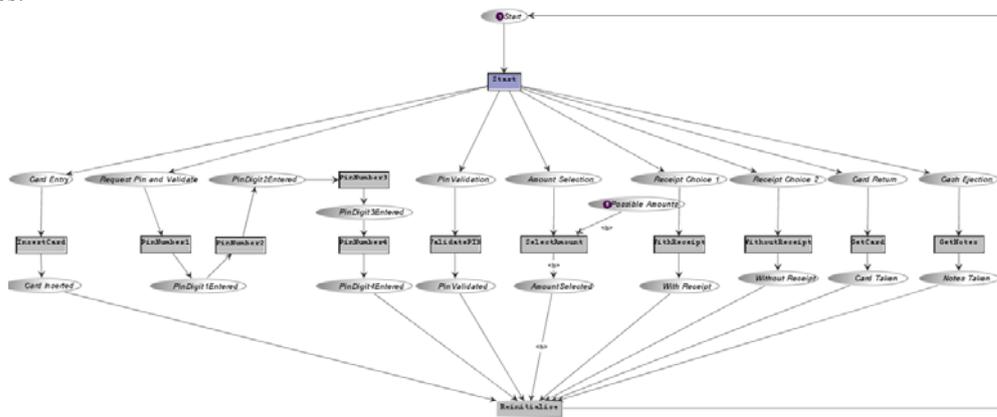


Figure 5-11. ATM System Model with Full Interleaving

We also demonstrate in Figure 5-12 a basic ATM system model with one interleaving option, which is arguably better than allowing full freedom. For example, in most cases it would be inappropriate to allow a card to be

taken back out of the machine if the card was not first inserted. This model ensures that the user has been identified before allowing them to request a receipt, obtain the card or obtain the cash. Transition *Identification* is only friable when places *Card Inserted*, *Pin Validated* and *Amount Selected* contain tokens.

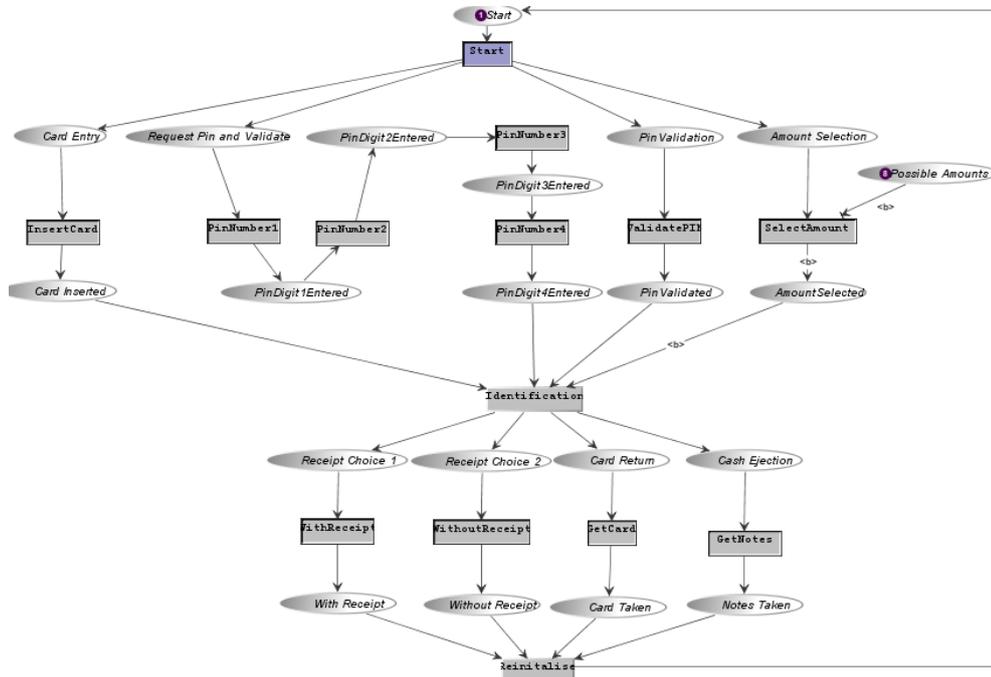


Figure 5-12. . ATM System Model with One Interleaving Option

To summarise this subsection on explicitly handling sequencing errors, the behaviour of the system in terms of order of events is closely related to the task model. If the user’s tasks are interleaved, the user is likely to make mistakes if the system is too restrictive. The system must therefore express options and address such issues. However if the user’s tasks are sequential, there is no need for the system to provide many options.

3.4 Explicit handling of timing errors

The final type of error we consider in this chapter are timing errors. This involves specifying what corrective actions the system has to take when a timing assertion is violated. In our case, we have modelled as an example that if the user fails to take their card from the machine within 5 seconds of it being ejected, the card will be retained. In terms of modelling with ICOs a simple modification to a system model is required. The transition dealing with timing issues in Figure 5-5 is label number 7. This section of the diagram has been reproduced in higher quality in Figure 5-13. As soon as the autonomous transition *EjectCard* is fired, the system begins counting 5000ms. If within 5000ms the *GetCard* transition is fired (i.e. the user takes their card from the machine, or the “Get card” button of the UI is clicked) the process continues as normal, a token is deposited in place *Card Taken*. If nothing happens within 5000ms, the *Timer* transition will fire, taking the token from place *Card Ejected* and placing a token in places *Card Retained*. This will allow the system to reinitialise. The simulation of the system model after a card has been retained will only allow the 2nd available card to be used.

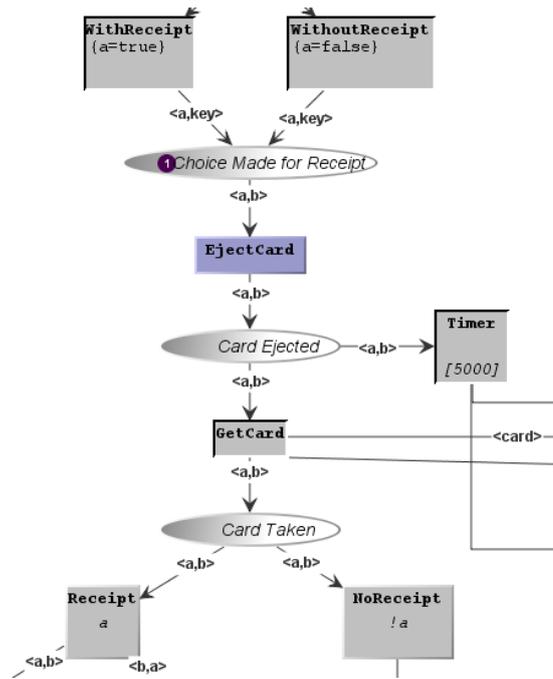


Figure 5-13. Timing Feature of the Extended System model (label 7 in Figure 5-5)

From a usability perspective, it is known that imposing temporal constraints on users is not ideal. However, such measures must be taken to achieve safety and security.

We have presented in this section, modelling techniques for addressing data entry, sequencing and timing errors as well as entering the PIN for the wrong card and post-completion errors. Further types of errors (such as those presented in Table 5-4) including rule-based and knowledge-based could have been considered and modelled in this section. Indeed, such errors, for example if the user tries to select an amount of cash that is not available in their account requires information from other sources, such as a database of account information. This could also be modelled using the ICO notation.

4 Model verification

As discussed in the literature review, Petri nets can be used to reason about model properties. Properties such as reinitialisation, liveness, dead-locks, bounded/unbounded systems. The analysis technique we are particularly interested in is Marking Graph Analysis because it provides a systematic analysis of all possible paths of interaction with the system model by identifying all possible reachable states. This data is useful for us when comparing task analysis scenarios and system interaction scenarios. We illustrate in this section how to apply the Marking Graph Analysis technique to the first ATM system model reproduced here in Figure 5-14. We have decided to implement the approach on this simpler model for explanatory purposes though.

4.1 Verification of properties

Before presenting the marking graph, we can confirm the following system properties:

- Reinitialisable: yes, because the model can always return to its initial state
- Bounded: yes, the number of tokens in the network does not exceed a finite number
- Deadlocks: no, therefore the model is live
- Dead Transitions: there are no transitions that will never be fireable

There are 20 reachable states for the simple ATM system model provided in Figure 5-14. The following list provides the markings of each state. We only detail places that contain tokens. I.e. initial state M0: {start, possible amounts*8} means that place *start* contains 1 token and place *possible amounts* contains 8 tokens. All other places within the Petri net model do not contain tokens. Places containing (true) or (false) refer to the status of preconditions.

M0: {start, possible amounts*8}

M1: {request pin & validate, possible amounts*8}
M2: {pin digit 1 entered, possible amounts*8}
M3: {pin digit 2 entered, possible amounts*8}
M4: {pin digit 3 entered, possible amounts*8}
M5: {pin digit 4 entered, possible amounts*8}
M6: {pin validated, possible amounts*8}
M7: {amount to be selected, possible amounts*8}
M8: {offer choice of receipt, possible amounts*8}
M9: {choice made for receipt(true), possible amounts*8}
M10: {card ejected(true), possible amounts*8}
M11: {card taken(true), possible amounts*8}
M12: {notes ejected(true), receipt ejected(true), possible amounts*8}
M13: {notes taken(true), possible amounts*8}
M14: {receipt taken(true), possible amounts*8}
M15: {finish, possible amounts*8}
M16: {choice made for receipt(false), possible amounts*8}
M17: {card ejected(false), possible amounts*8}
M18: {card taken(false), possible amounts*8}
M19: {notes ejected_1(false), possible amounts*8}

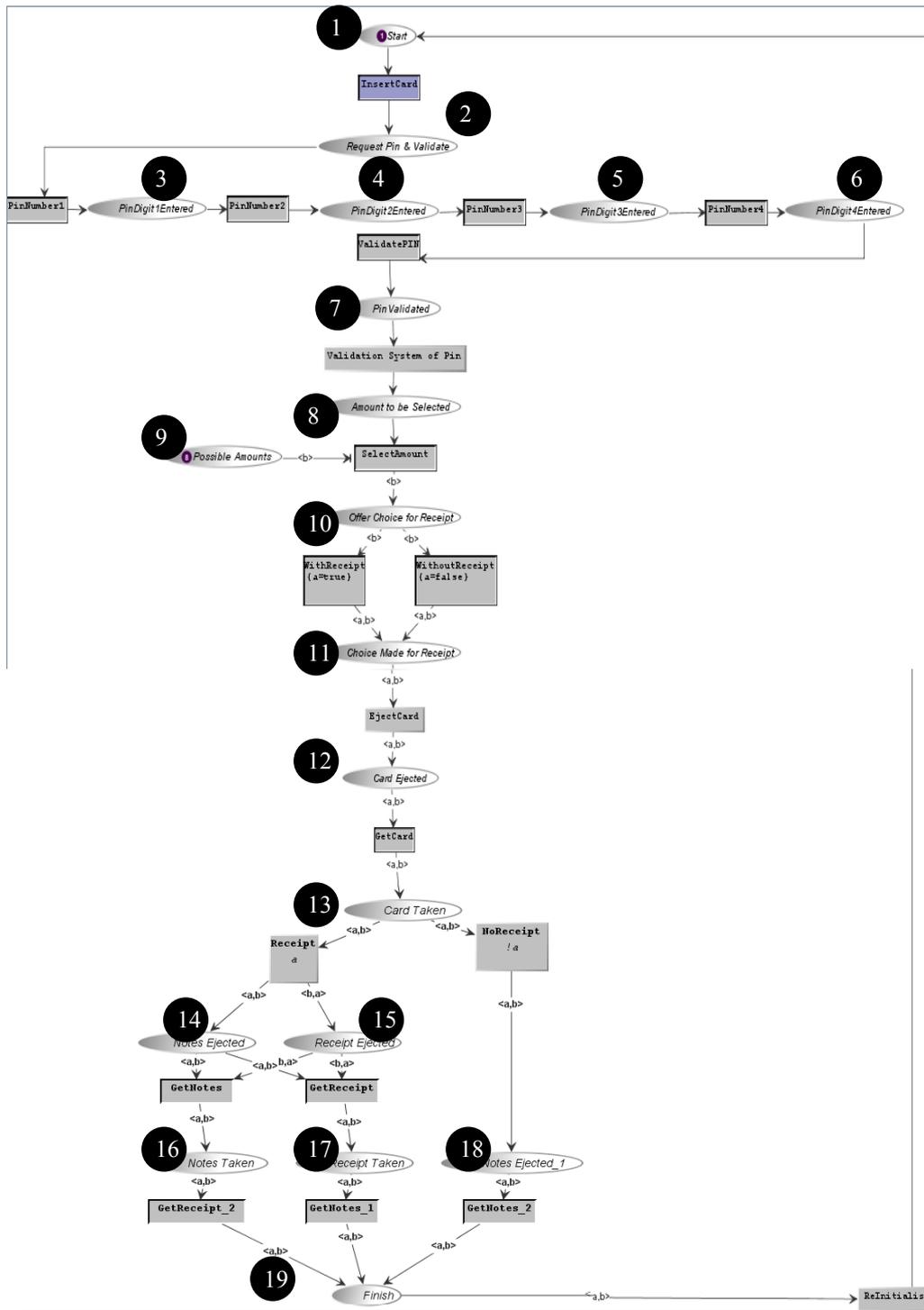


Figure 5-14. Numbering the Places of the Simple ATM System Model

Following the identification of all reachable states and their markings (according to tokens in places), a marking graph can be produced graphically illustrating how states are related according to transitions. In the marking graph presented in Figure 5-15, the 20 possible states are represented as eclipses. This time, as opposed to only detailing places containing tokens, we have given each place in the network a number, 1-19 (see Figure 5-14), and have represented the marking of each place for each state. For example, initial state M0 is represented as 1,0,0,0,0,0,0,1*8,0,0,0,0,0,0,0,0,0,0 in the graphical marking graph.

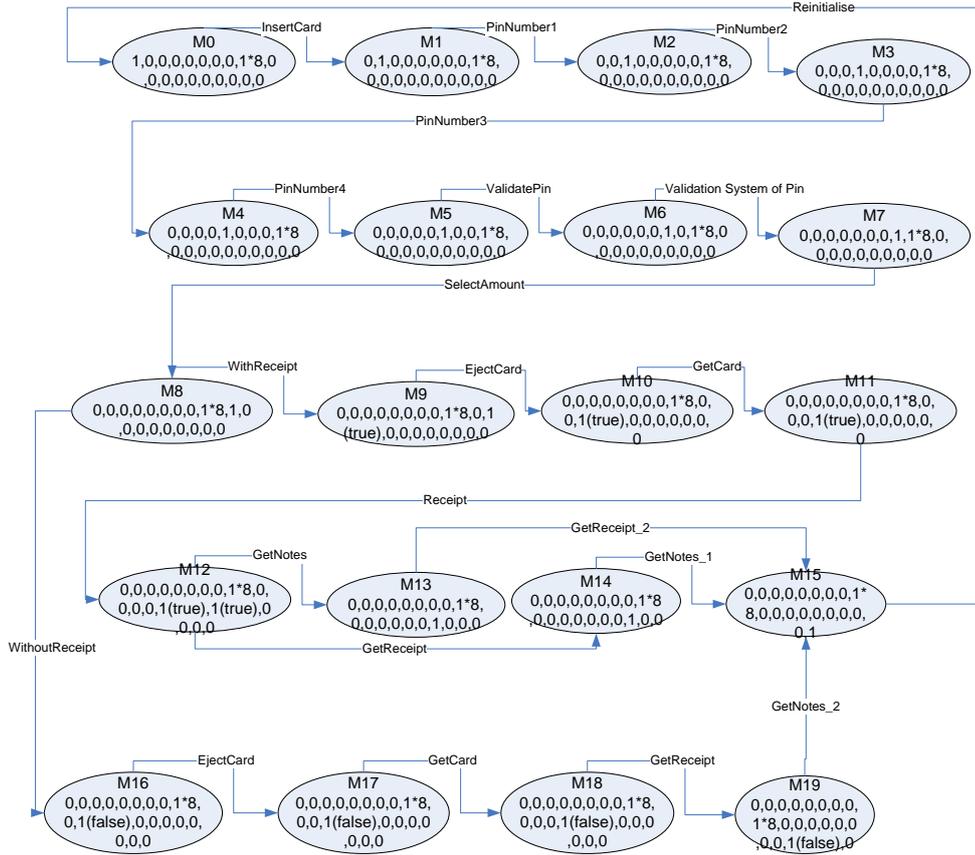


Figure 5-15. Marking Graph for Simple ATM presented in Figure 5-2

5 Supporting Integration

As with the previous task modelling and error management chapter, this section of the chapter is dedicated to the discussion on integration of the phase with other phases of the framework. Figure 5-16 illustrates the phase annotated with references to sections, figures and chapters where each particular process, document etc is presented. The framework allows the design process to commence from any phase depending on the current situation. Once the system modelling phase is complete, the system model is fed into the task modelling phase for cross model checking, ensuring compatibility of data representation. Here, it will be possible to know if the system model supports the task model and vice versa, including erroneous behaviour modelled in both models. Since the framework is iterative, this make take several attempts. As explained in the previous chapter, there is phase of quantitative evaluation and also testing. The testing phase relates more closing with the system modelling phase because it provides a means of model validation/verification by applying mathematical analyses to the models. We use analyses that result in potential scenario identification and extraction which is also useful for task model checking. The “system error analysis” part of the system modelling phase receives data from the safety modelling phase to include additional known erroneous behaviour. Furthermore, the system modelling itself receives (if necessary) data from the safety modelling phase since the process of design may well start from an existing incident or accident. The system model may begin entirely from an existing problem. A number of parts of Figure 5-16 refer to chapter 7, the barrier modelling parts which will be discussed in more detail in its respective chapter.

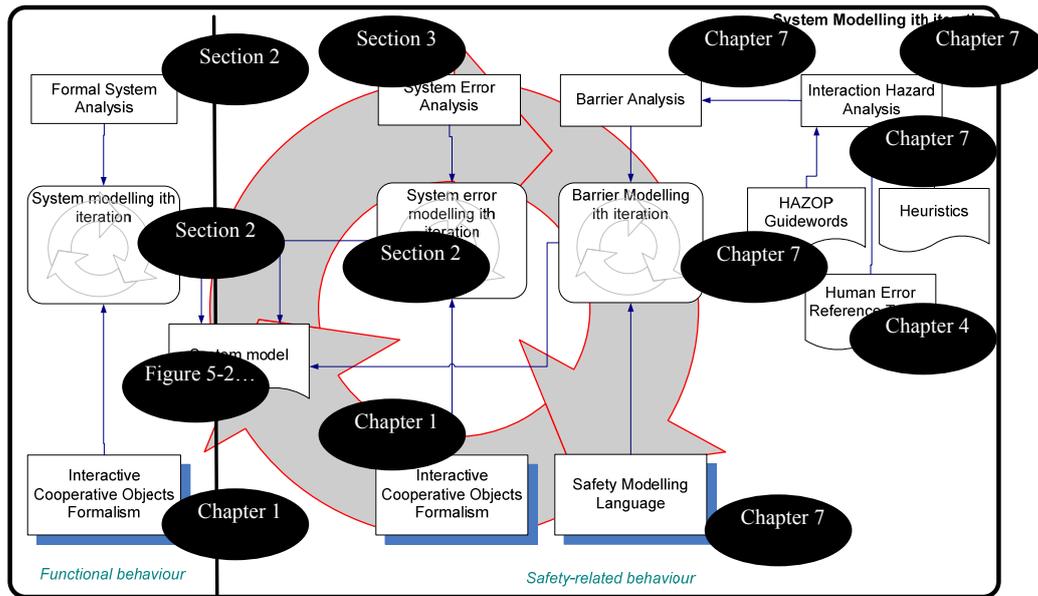


Figure 5-16. System modelling phase of the generic integrated modelling framework

6 Conclusion

We have seen in the literature review that much work has focused on the formal specification of user interfaces and applications, few consider the practical uses of formal modelling of errors for the improved design of interaction systems.

This chapter has shown that the ICO notation is fully capable of supporting human ‘error’ integration and erroneous behaviour with the formal specification of an interactive system. We have related the results of the task modelling and error management approach, presented in the previous chapter, which proposed using the Human Error Reference Table for the systematic analysis of human errors, to the modelling of erroneous events in a system model.

We have modelled three categories of errors, data entry errors, sequencing errors and timing errors as well as several particular skill-based cognitive errors such as entering the PIN of the wrong card as well as post-completion errors. The chapter has also discussed the problems associated to satisfying usability principles and safety/security issues. It has been shown that safety requirements may induce restrictions on human error tolerance, for example by imposing timing constraints on users, or allowing a fixed number of PIN entry attempts.

The approach presented in this chapter tackles the functional behaviour part of the system modelling phase of the generic integrated modelling framework, and the safety-related behaviour part of the system modelling phase. The subsequent chapter will follow-on from this discussing our approach for barrier analysis and modelling to complete the explanation of the system modelling phase.

Chapter 6

Safety Modelling

“Any intelligent fool can make things bigger and more complex... It takes a touch of genius - and a lot of courage to move in the opposite direction” (Albert Einstein, 1879 - 1955)

Chapter Summary

Although safety is an integral part of the development process, with data and information obtained fed into other modelling phases, safety modelling has been separated in order to treat it separately and highlight its distinct quality as Leveson also suggests (Leveson 1995)p.182. The modelling of safety using Safety Cases and Events and Causal Factors Analysis is not our main contribution, however these techniques applied to incident or accident reports feed into other phases of our generic integrated modelling framework and play an important role, particularly in validation phases. We show in this chapter how data obtained from incident and accident investigation techniques can be used to inform the formal specification of a system model in order to make hazards explicit. This mechanism is a way of designing for error avoidance, error detection and error recovery.

1 Introduction

In (Leveson 1986), Leveson, has argued that most accidents involving software are due to errors in the software requirements specification though Johnson and Holloway (Johnson and Holloway 2006) disagree. The authors use the term ‘hindsight bias’, which they say occurs when software engineers automatically assume that there has been a failure in requirements engineering simply because an accident has occurred. Furthermore, the term is exemplified stating “bias can be interpreted as influences that prevent objective consideration of an issue or situation”. These influences can lead to or be reinforced by the use of logical fallacies to support the findings of accident investigations. In particular, they often seem to be used to justify the identification of requirements failure in the aftermath of software related incidents and accidents (Johnson and Holloway 2006).

The increasing complexity of computer-controlled safety-critical interactive systems is placing increasing demands on the investigation of adverse events. At the same time, there is a growing realisation that accident investigators must consider a wider range of contributory and contextual factors that help to shape human behaviour in the causes of safety-related incidents.

A vast array of development techniques can be used to support the design of complex interactive systems that have yet to be developed. Far fewer provide explicit support for redesign in the aftermath of incidents and accidents. Existing investigatory tools, such as Why Because Analysis (Ladkin and Loer, 1998), provide relatively little support for such redesign.

In order to deal with this complex combination of contributory factors and systems, we promote the use of abstraction (via models) as a way of representing these components and their interrelations whether it is design, construction or investigation.

These adverse affects of accidents are complex to analyse because of the combination of contributory factors (human and technical) that together caused the system (humans and machines) to fail. Trying to discover what went wrong and why, can be extremely multifaceted especially when dealing with complex, computer-based systems. We argue that involving opinions grounded on different backgrounds can shed new light on the root causes of an accident. Lekberg's study of incident investigators within the Swedish nuclear power industry has shown that analysts tend to identify root causes that are closely related to their own background and training (Lekberg 1997). Johnson has also shown that by recruiting investigators from different backgrounds, it is possible to identify recommendations that might otherwise have been overlooked (Johnson 2003)p215. By expanding the range of experience recruited to an investigation, it seems more likely that we will learn additional lessons from previous failures.

This chapter aims to show how techniques from multidisciplinary domains (incident and accident investigation and systems design/modelling) can be used synergistically to support one another in order to both better understand an accident and its causes and to inform the design of safety-critical interactive system. A long-term aim of our research is to identify human ‘errors’ early in the development process to facilitate the design of error-tolerant systems. To this end, we propose to extend the system model to include hazardous states. This approach will help us in our ultimate goal of designing safer safety-critical interactive systems.

Recent accidents in domains ranging from aviation to healthcare have stemmed from complex interactions between human ‘errors’, system ‘failures’ and managerial issues. Failure in one area of a system, as we have often seen, will compromise other forms of protection (Johnson 2003) p366. The complexity of these interactions makes it difficult to use insights from previous failures to inform the design of many safety-critical systems. Further problems stem from a lack of integration between the methods that are used to investigate accidents and those that are used to inform the development of complex systems.

In this chapter, we show how an existing accident investigation technique can be combined with our ‘state of the art’ systems development techniques using ICOs, for large concurrent systems. To be more concrete, we present an approach for reflecting on flawed safety cases to shed light on false assumptions and design implications which can be further considered to improve the safety case. We also prove properties of a formal system model of an interactive safety-critical system such that, a given accident is prevented from occurring in the re-designed system model. This does not, of course, imply that the accident cannot recur. There may be problems in the development of an adequate system model since human behaviour, actions, emotions etc are difficult to capture. However, this integrated approach may serve to increase our confidence in applying ‘lessons learned’ to the redesign of complex, safety critical systems.

We present how to take into account the hazards highlighted in the safety-cases. The system model is extended to explicitly represent hazardous system states and actions identified from the safety cases. The aim is not to model every possible problematic situation that can occur in the system model. It is highly reasonable that not all of the information provided in a safety-case can be applied to a system model, for example information regarding personnel and training. However, we aim to show how the behaviour of the system can be influenced by hazardous events and also lead to hazardous states.

The aim of this chapter is to show that by analysing an accident from multidisciplinary perspectives, i.e. incident and accident analysis and formal specification of interactive systems, we can improve the analysis process and conclude with reusable results that can be used to improve future versions of the systems.

2 Incident and Accident Investigation

Formal specification and verification techniques can be used to support the design of interactive systems. The assumption is usually that a period of requirements elicitation helps to drive the design, then implementation and finally evaluation of an interactive system. Usually these different activities merge and are conducted in an iterative fashion. In contrast, this chapter and proposed approach focuses on the aftermath of an adverse event. In such circumstances, there is a pressing need to ensure proposed designs do not propagate previous weaknesses into future systems.

As mentioned, our analysis begins with the products of an incident or accident investigation. This is justified because there are no recognized design techniques that start with such an assumption. Existing investigatory tools, such as Ladkin and Loer’s Why Because Analysis, provide relatively little support for subsequent redesign (Ladkin and Loer 1998). They focus more on the identification of root causes in an objective and systematic manner. Other approaches, such as Leveson’s STAMP technique, help to understand the organizational and systemic context in which an accident occurs but, again, offer few insights into the specific steps that can be taken to avoid any future repetitions of adverse events (Leveson 2004).

Of course, our decision to begin with the products of an accident investigation is both a strength and a weakness of our technique. It is a strength because it lends novelty given the absence of previous approaches in this area. It is also a weakness because it raises questions about the usefulness of the approach. How do we apply the technique in the absence of an accident investigation? What happens if risk analysis identifies a potential problem with an existing design but no accident has happened yet and there is no formal investigation yet?

In order to address these issues, we have chosen to use a safety case notation to represent the insights from an accident investigation. Techniques such as the Goal Structuring Notation, presented in the state of the art, help to

identify the key arguments that can be used to demonstrate the relative safety of a complex system. Accidents often reveal the flaws in these arguments. Hence, we can represent the insights obtained in the aftermath of an adverse event in terms of the consequent changes that must be made to a safety case. Of course, this is not the only way in which we might trigger revisions to such arguments. Near-miss incident reports or revised hazard and risk assessments can also be used to further inform the safety cases that justify a complex system (Johnson 2003).

Our approach which uses Petri nets to represent accident report information does not have the same goal as the work discussed in the literature review by (Johnson et al. 1995). We do not intend for the Petri net models to aid accident analysts or safety managers for example, to better understand accident reports. We believe that experts of varying backgrounds, such as system designers, Petri net modellers and accident investigators should continue to use techniques and methods that they are familiar with. We do however; propose to use Petri nets and data from accident reports as a means of informing model based systems design. The hazardous events and conditions leading to an accident are identified from the accident report by accident analysis specialists using typical accident investigation techniques.

2.1 A Multidisciplinary Approach to Incident & Accident Investigation to Inform

We present an approach exploiting three complementary techniques for the analysis of an incident or accident to inform the future design of safety-critical interactive systems and to address the issues of system redesign after the occurrence of an incident or accident.

The starting point of our work is the realisation of the need to employ different techniques in order to examine human, technical and system issues that make up the accident we are investigating.

With respect to the generic integrated modelling framework, this part of the approach fits into the safety modelling phase and testing phase. Figure 6-1 presents these two phases in abstraction from the rest of the framework. We have included the data flow arrow which connects the two phases (an information based data flow). It can be seen, that information from the incident or accident report is used to perform an Events and Causal Factors Analysis (ECFA). As we will see in this chapter, the ECFA is used for model validation. We also include the process arrow from the system model for the marking graph analysis. Data flow arrows connecting these phases to others are not shown for simplification.

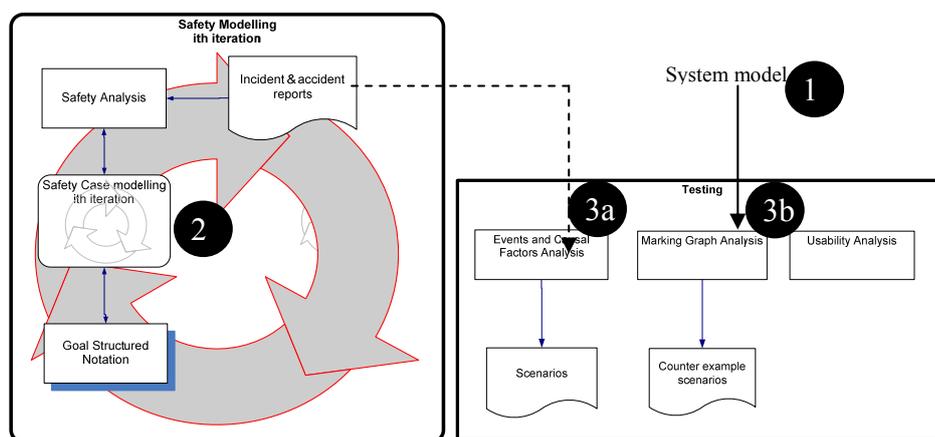


Figure 6-1. Safety modelling phase and testing phase of the generic integrated modelling framework

A range of techniques have been developed to address these issues. For example, as we presented in Chapter 4, standard task modelling techniques for individual and collaborative tasks have been extended to analyse the causes and consequences of operator ‘errors’. Our proposed approach aimed to tackle task modelling during early design phases. However, task models can also be inspected to identify not only the role that different activities played in the causes of an accident; they can also be used more generally to search for alternate problems in the operation systems. For example, an accident may reveal individual workload problems that are symptomatic of wider issues throughout a safety-critical application. A further justification for using this approach during accidents and incidents investigation is that it has recently been extended and is now able to support the identification of different types of common interaction ‘failures’. Though the previous chapter

illustrated the approach from the perspective of no accident occurring, the technique is applied in the same way as and will not be repeated here from a post-incident or accident perspective. This technique is used to understand the tasks and activities required to achieve goals allowing us to analyse low-level failures involving decision making, physical interactions with the system, wrong judgements etc. The technique is applied to the first case study involving a fatal mining accident (see Chapter 8). Indeed, classical use of task modelling techniques deals mainly with an error-free perspective on user tasks. The combination of techniques proposed in this thesis also reflects the different research traditions of the two international teams involved in this work.

Thus in this chapter, we provide further techniques to address safety-related issues in the model-based design of safety-critical interactive systems design.

The techniques used are safety cases. They are used here to reflect on flawed assumptions, which is controversial with respect to the way safety cases are usually employed (See Chapter 2 section 3.3). Secondly, we use Events and Causal Factors Analysis, a technique used in accident investigation to identify the path of events and contributing factors leading to an accident, and finally, Marking Graphs (see Chapter 1 and Chapter 5), used in this case to systematically explore all possible scenarios leading to the accident based on a formal system model. Towards the end of the chapter we discuss how the results can, in turn, be used to modify the system model such that the same accident is made impossible in the informed system model. This is done by simulating the identified scenarios on the adapted system model.

This chapter therefore shows how an analysis of Safety Case arguments can be used to support the application system analysis during the investigation an incident or accident. For the purposes of explanation, we will again be using an ATM as an example.

It is important to emphasise the multidisciplinary nature of the work described in this chapter, and thesis in general. Safety cases are often required by regulatory bodies to assure that safety standards are met. Usually, analytic practices such as Fault-Tree Analysis, or formal verification and so on are used, however, we are interested in the higher level argumentation, and not the detailed techniques that were employed for the system in question. Safety argumentation is insightful as it represents how technology and human factors are engineered in order to mitigate hazards. Given a failure of such argumentation in the aftermath of an accident, looking at a flawed safety case can shed light on false assumptions and design implications can be further considered to improve the safety case. This is an important area for further work, given the commercial and regulatory importance of these documents.

The intention, in this chapter, is to show that it is possible to trace the common causes of the system and human failures back to the assumptions and arguments that are embodied within a safety cases.

Referring back to the literature review, our approach on safety modelling is different to that proposed by Kontogiannis et al., (Kontogiannis et al. 2000) in that they use Petri nets to model an accident report. Our approach consists of first modelling the system independently of the accident using information available in the accident report and external sources if the accident report is insufficient. We then extend our model in various ways to include issues such as hazards, human error, barriers.

The list below provides a textual breakdown of the activities performed for this phase of safety modelling with each technique described. Figure 6-2 provides a graphical overview. Please note although the steps are numerical, some must be performed in parallel.

Formal specification of system (discussed in Chapter 5), modelled using a dialect of high-level Petri nets performed by system designers. This is labelled 1 in Figure 6-1, the generic integrated modelling framework abstraction and Figure 6-2 presenting a breakdown of the approach phases relevant to this chapter. The accident report is used to formally model the interactive system. Where there are gaps in data required for modelling the system, technical reports on equipment used are researched. This Petri net based approach to system modelling has the capabilities to represent actions, states and data in the same model. In addition, by using the Interactive Cooperative Objects (ICO) formalism (Navarre 2001) we are able to model highly interactive systems and thus include both actions performed by humans and events produced while humans interact with the system. Being able to gather all these elements in one model is mandatory when analysing an accident involving interactive system and human operators. The approach proposed here does not require the explicit use of ICOs; any formalism capable of modelling all these elements would fit within the method.

Even for a system that is not safety-critical, it is still necessary to ensure the system's efficiency and reliability but these issues are more salient for safety-critical systems. Petri nets are widely used in systems engineering.

This does not, however, eliminate the concern that a complementary approach based on Petri nets and Concurrent Task Trees may impose strong expectations on the skill sets that would be required for any investigation. It is also hard to make the right abstractions of the real accidents and the system behaviour involved in the accident, which is amenable to formal specification and automatic analysis and still reflects all the necessary aspects to explain in a satisfactory way the causes of an accident. Equally, a range of recent initiatives have begun to recognize the importance of explicit and detailed training if accident investigation agencies are to cope with the increasing complexity of many safety-critical application processes. For example, the NTSB recently opened its Academy for investigators in Virginia. An introduction to Petri nets and the application of basic and high level Petri nets including the ICO formalism are presented in Chapter 1 and Chapter 5.

Safety Cases in order to explore the high-level relationship between contributing factors, this is performed by accident investigators. This is labelled 2 in Figure 6-1, the generic integrated modelling framework abstraction and in Figure 6-2. We produce safety cases based on our assumptions regarding the system in question, and on the information that is given by the accident report, we are limited at developing only initial safety case arguments. However, they are, to a great extent, insightful for the purposes of this research. Throughout, safety cases are used to introduce an analytic technique employed. For instance, a safety case arguing the safety of a certain barrier introduces the analysis of the specific component, thus linking it to other analyses presented here.

Scenario producing techniques, Events and Causal Factors Analysis and Marking Graph Analysis. These are labelled 3a and 3b respectively in Figure 6-1, the generic integrated modelling framework abstraction and in Figure 6-2. These serve two purposes. Firstly, it is intended to ensure that the system model accurately models the sequence of events that led to the accident according to information provided by Events and Causal Factors Analysis. Secondly, our approach can be used to reveal further scenarios that could eventually lead to similar adverse outcomes

This is the aim of the research; to use the given information, presented in different formats and levels of formality, to inform the specification of a redesigned formal system model using ICOs. We anticipate that the approach is likely to minimise the adverse affects of erroneous events while interacting with such systems.

Figure 6-2 presents a graphical overview of our multidisciplinary perspectives on accident analysis at an abstract level. The flow diagram indicates the key phases of the work presented in this chapter. As can be seen, the envisaged approach is relatively complex. This is justified given the nature of the systems that are being studied and the requirement to integrate techniques from two very different domains. The procedure is divided into two parts, the incident and accident investigation and the system design part (top and bottom of the diagram).

This process suggested relies on the identification of scenarios. By scenarios, we mean possible sequences of events that can lead to the same accident and also the sequences of events available in the system model. Two techniques support these tasks. The first technique, Events and Causal Factors Analysis (ECFA) provides a scenario of the events and causal factors that contributed to the accident. This is shown in the upper part of the diagram. In parallel to this technique, we apply the Marking Graph technique, to the 'formal ICO system model including erroneous events'. The Marking Graph technique is typically used together with Petri net models to generate an exhaustive list of every possible scenario, or sequence of events, that could occur in a particular abstraction of a complex system. This exhaustive list will include 'normal' behaviour as well as scenarios that lead to accidents. For this reason, there exists an 'extraction of relevant scenarios' phase in the process diagram which is applied to ECFA and the Marking Graph results. By relevant, we mean scenarios that lead to the given accident and not scenarios resulting in correct operation of the system.

We will assume that our ATM machine is situated within a form of airlock; a room that can only be entered with an authorised bank card, and can only be exited with an authorised bank card. Our scenario will assume that a person enters the airlock using their authorised card, has their card retained (perhaps because of three failed PIN entry attempts). At the same time, a fire occurs in the building and the person cannot exit the airlock because the building has been designed to only allow exit on use of an authorised bank card.

3 Application of the approach

A principle phase of the approach is the formal specification of the system. This has been presented in the previous chapter thus we will continue with the subsequent phase. Please note that these phases do not necessarily have to be carried out in a particular order. The incident and accident investigation techniques can be applied at the same time as the system modelling techniques until information from one process is required for the other.

3.1 Safety Cases

The diverse analytic techniques we have used in this chapter needed a way to be connected as to how they all relate to the accident that occurred. In order to do so, we used the Goal Structuring Notation (Kelly 1998) to develop safety cases for each of the issues under investigation. Safety cases comprise of safety requirements and objectives, the safety argument, and the supporting evidence. Goal Structuring Notation (GSN) is a graphical argumentation notation, which represents the components of a safety argument, and their relationships.

Safety cases build up an argument that any risks associated with an application is as low as reasonably practicable. These arguments can be based on assumptions about the relationship between an application and other systems. They can also be based upon assumptions about the environment that a system will operate in. These conditions can be difficult to control and hence can force revisions to a safety case in order to demonstrate that the system continues to be safe within revised operating conditions. Changes in operating practices or the functional demands on a system can introduce the same need for revised argumentation.

In cases where a revised argument cannot be made then it can be necessary to introduce design changes. A continuing problem here is that it is difficult to ensure the modularity of a safety-case. Hence any changes within the design of a system sub-component will often force renewed inspection of the argumentation associated with the safety cases of other related components. If the changes cannot be accommodated within the argument then redesign may be necessary.

Figure 6-3 depicts the highest level of the safety case for the ATM within the safety airlock using GSN. The safety requirement G1 in this example would be to assure that the ‘Evacuation procedure of the airlock’ is acceptably safe. In order to argue that G1 is valid, four strategies have been deployed (S1.1, S1.2, S1.3, S1.4). The argument is continued with new safety sub-goals, until they can not be broken down any more (we are only demonstrating the safety cases on the hypothetical ATM system as an example, the provided safety cases are therefore not fully exhaustive). At this stage, evidence proving that the safety requirement is met is required (see E1.1.1.1 in Figure 6-4 for example). For arguments that need further elaboration, GSN uses the diamond symbol to index to related elements of a safety case.

Safety Cases are interesting and useful because they present the whole view and strategy towards achieving safety. When applied in an incident or accident the way we suggest in our approach, we can discuss not only technical or human failures, but understand more about the deeper problems that led to these failures taking place, organizational and so on. Safety cases using the GSN for the ATM in an airlock example are presented in Figure 6-3 to Figure 6-7. Although based on our hypothetical views about the system, we can assume that in order to assure safety of the evacuation procedure in our example, claims such as those made in the safety cases provided here would have been made.

Though in our approach, the original safety cases would normally be obtained after an accident, in order to identify flawed arguments, here we have developed safety cases based on assumptions of our ATM incident.

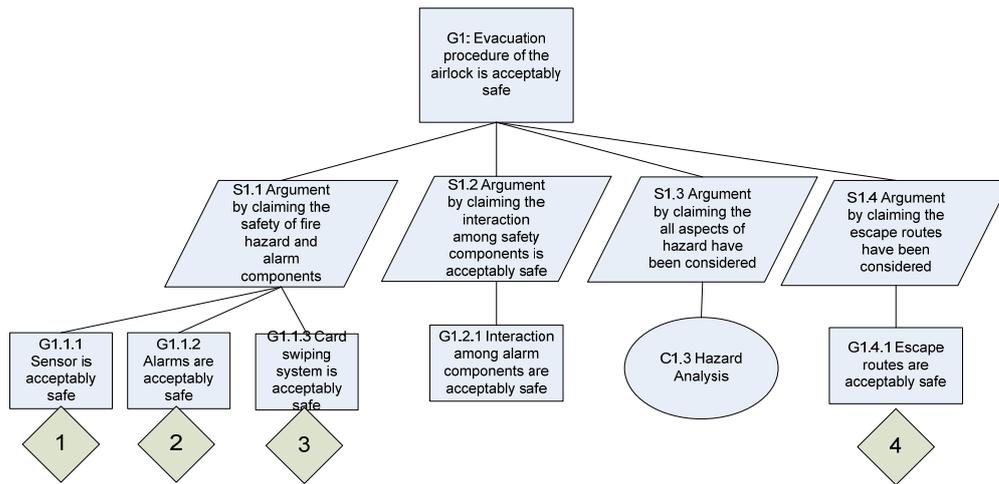


Figure 6-3. High Level Safety Case of ATM in Airlock Using GSN

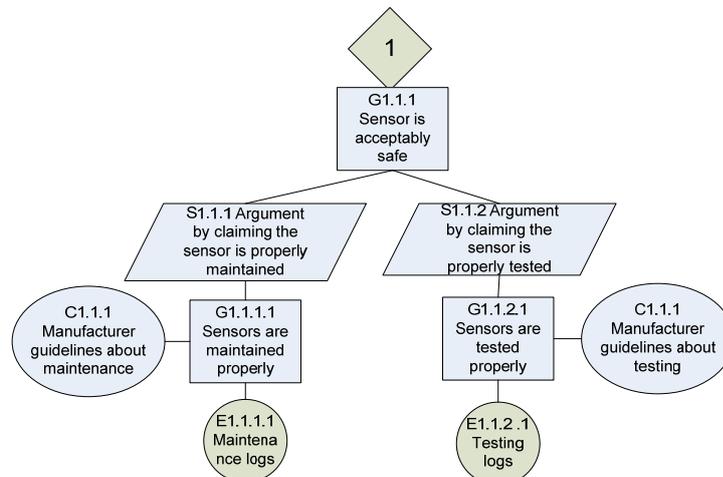


Figure 6-4. Goal 1 Safety Case Using GSN

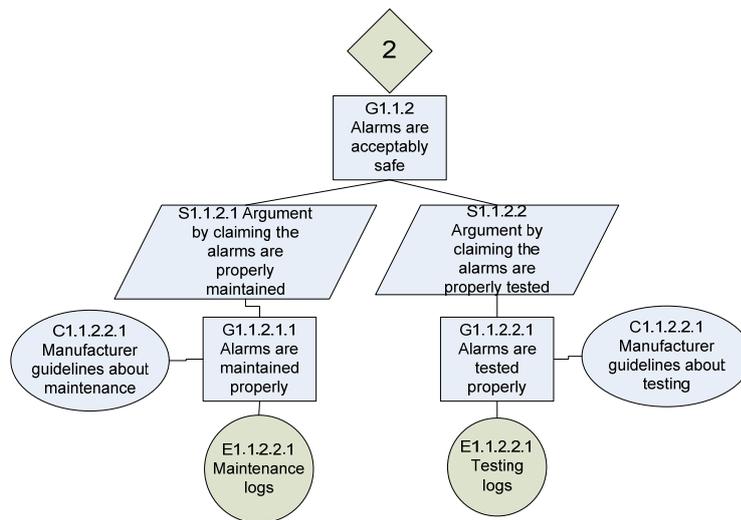


Figure 6-5. Goal 2 Safety Case Using GSN

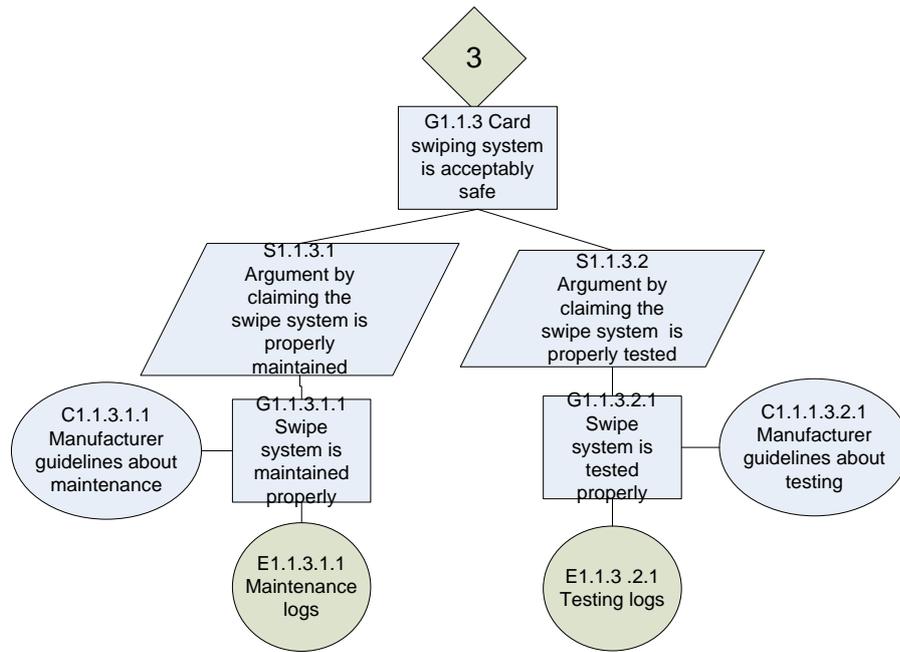


Figure 6-6. Goal 3 Safety Case Using GSN

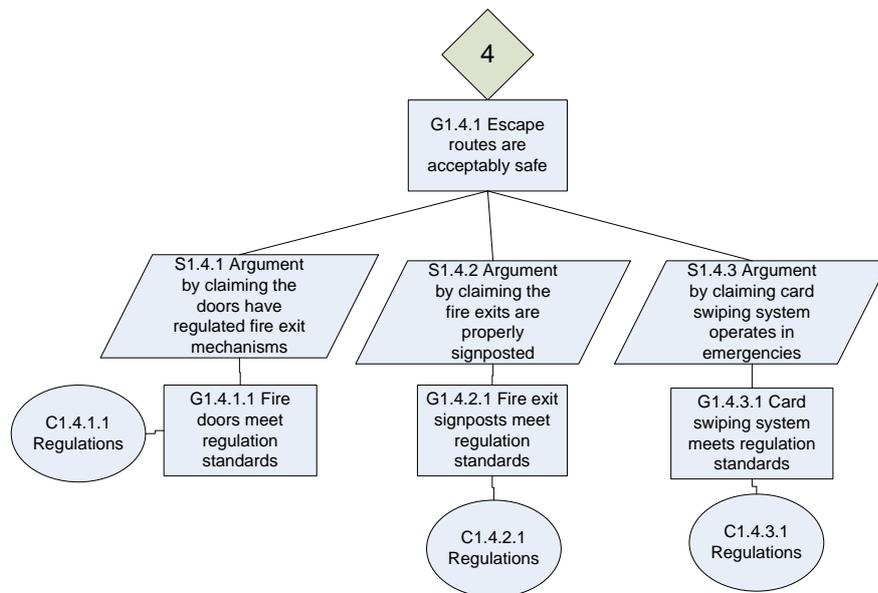


Figure 6-7. Goal 4 Safety Case Using GSN

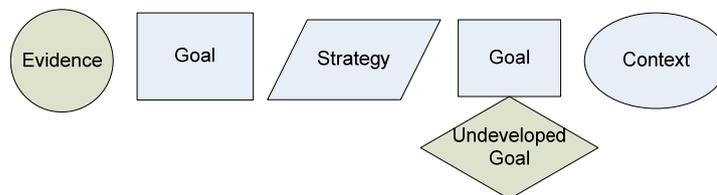


Figure 6-8. Safety Case Diagram Key

Accident reports can be an unreliable source of information; they may be incomplete and can be subject to bias. This would create potential problems for any approach that uses the results of these investigations to inform future redevelopment. It is, therefore, important that potential omissions, inconsistencies and bias are identified prior to the analysis described in this approach, for instance, using the types of systematic analysis advocated in Johnson (Johnson 2003). In contrast, our use of GSN focuses on establishing an argument for redesign rather than on the identification of weaknesses within a mishap report.

The following section will present the application of the incident and accident investigation techniques, Event and Causal Factors Analysis to the ATM in an airlock example. Following which, we will discuss how to relate both safety modelling techniques to the formal specification of the system in order to improve future designs of the same or similar systems. The approach allows us to identify flawed safety case arguments, run incident scenarios on our informed system model to ensure the same scenario is no longer possible, and also to potentially identify new hazardous scenarios.

3.2 Events and Causal Factors Analysis

It is necessary for the purposes of our approach to have an overall perspective of how different factors and conditions and different system states evolve throughout time towards leading to an incident or accident. Events and Causal Factors Analysis (ECFA), does exactly the above; in a chronologically sequential representation of events, ECFA can provide the analyst an overall picture of how different factors relate to one another and how they contributed to an accident. In brief, and as discussed in the state of the art, events are represented as rectangles and conditions as ovals. Events are connected with arrows, while conditions with dashed arrows. If an event or condition is an assumption, then it is represented with a dashed rectangle or oval accordingly. Based on chronological sequence, events are connected moving from left to right, keeping the main events on a primary horizontal line, and contributing or secondary events and conditions, above or below.

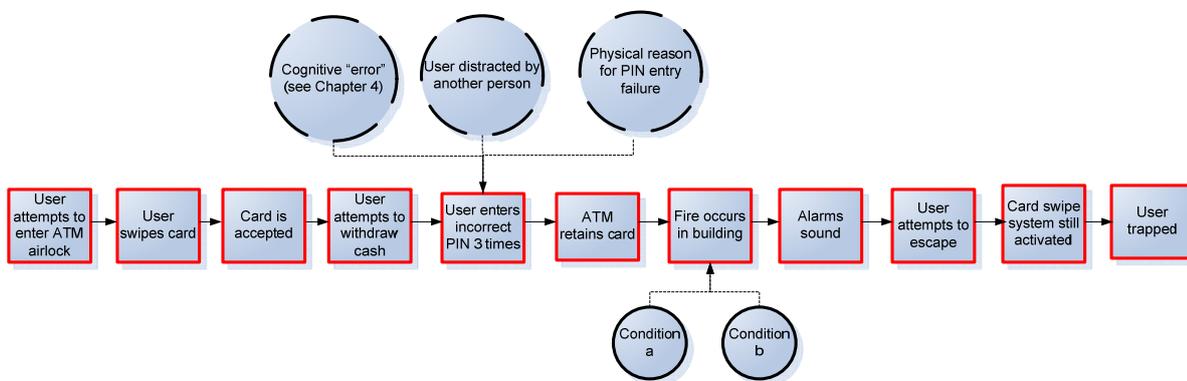


Figure 6-9. Simple ECFA for ATM Airlock “incident”

Figure 6-9 presents a rudimentary ECFA diagram for the ATM ‘incident’ described above. The main events beginning with the ATM software upgrade and ending with, users obtain/steal large sums of money, are presented in the centre of the diagram. We have included three external assumed conditions that may have contributed to the user entering the wrong PIN number. These are cognitive ‘error’ such as those discussed in the literature review, distraction, or having inside knowledge about the operations of the faulty ATM. In order for the user to have such knowledge, an event, also included in the ECFA states that the software programmer, for example, provided information to the user.

3.3 Relating Safety Analysis to System Model Validation

In this section we show how the safety cases and ECFA can be used to inform the specification of the system and to provide safety-related means for model validation.

System modelling can help to provide assurance that a given incident or accident is unlikely to occur in any system redesign. Safety-critical interactive systems in particular, benefit from such techniques because they can provide precise, complete and concise models. Many abstract description techniques also support accurate reasoning about system behaviours before they are developed. This is essential if we are to validate potential design changes in the aftermath of major accidents. Petri nets are widely used in systems engineering. This does not, however, eliminate the concern that a complementary approach based on Petri nets may impose unrealistic expectations on the skill sets that would be required for any investigation. Equally, a range of recent initiatives have begun to recognize the importance of explicit and detailed training if accident investigation agencies are to cope with the increasing complexity of many safety-critical application processes. For example, the USA National Transportation Safety Board recently opened its Academy for investigators in Virginia.

We discuss here, how the results of safety case analysis and ECFA can in turn, be used to modify the system model such that the same incident is made impossible in the informed system model. This is done by simulating the identified scenarios on the adapted system model.

We identify ‘relevant and runnable scenarios’ (sometimes by means of extraction if the system model is too large and we only wish to focus on one particular part of the model) represented in different ways (one as an ECFA diagram that can be read and applied to a system model, the other as a marking graph of the system model using Petri nets). These are fed into the redesign phase, as shown in Figure 6-2, in order to prevent a similar incident or accident from being triggered by a user or by the system (in terms of system model representation and model simulation).

Once the system model has been adapted according to the incident or accident scenarios identified, it is possible to manually run the scenarios on the new incident-tolerant system model in order to prove that it does not allow the same sequence of events to occur and thus not re-trigger the incident or accident, i.e. that the identified scenarios are no longer possible.

3.3.1 Using Flawed Safety Cases to Inform the System Model

By looking at the safety cases presented in the previous section, we can identify one particular hazardous state (in terms of system modelling), that is, when the user no longer has a valid card and needs to exit the building. From this perspective, we can argue that the safety cases are flawed. We can argue that a full safety analysis of the ATM airlock system was not implemented, and that this particular case was not considered.

This identified hazardous state can be used to inform the system model. Figure 6-10 presents the ATM system model developed in the previous chapter including this additional hazard. In order to model the hazard, the system model has been modified to include the airlock system. This is a simple modification in which access to the start of the ATM procedure is not possible until the airlock system receives a valid card and the user has entered the airlock. The same concept is applied once the user has finished their operation. They cannot leave the airlock without swiping a valid card.

Here we present the behaviour of the system according information obtained from the safety cases. The system model explicitly represents the fact that the user cannot exit the airlock when there is a fire and when they do not have an authorised bank card (because they incorrectly entered their PIN three times and the ATM retained their card). Figure 6-10 presents the new informed system model including the hazardous state “User Trapped”.

Since the model is complex and almost illegible as a screenshot, we have highlighted three parts of the model, which are the modifications made in order to support the explicit modelling of the hazardous state, in order to discuss in further detail. Each of these three parts will be reproduced as higher quality figures. The remainder of the model is the original ATM system model including error tolerance presented in the previous chapter.

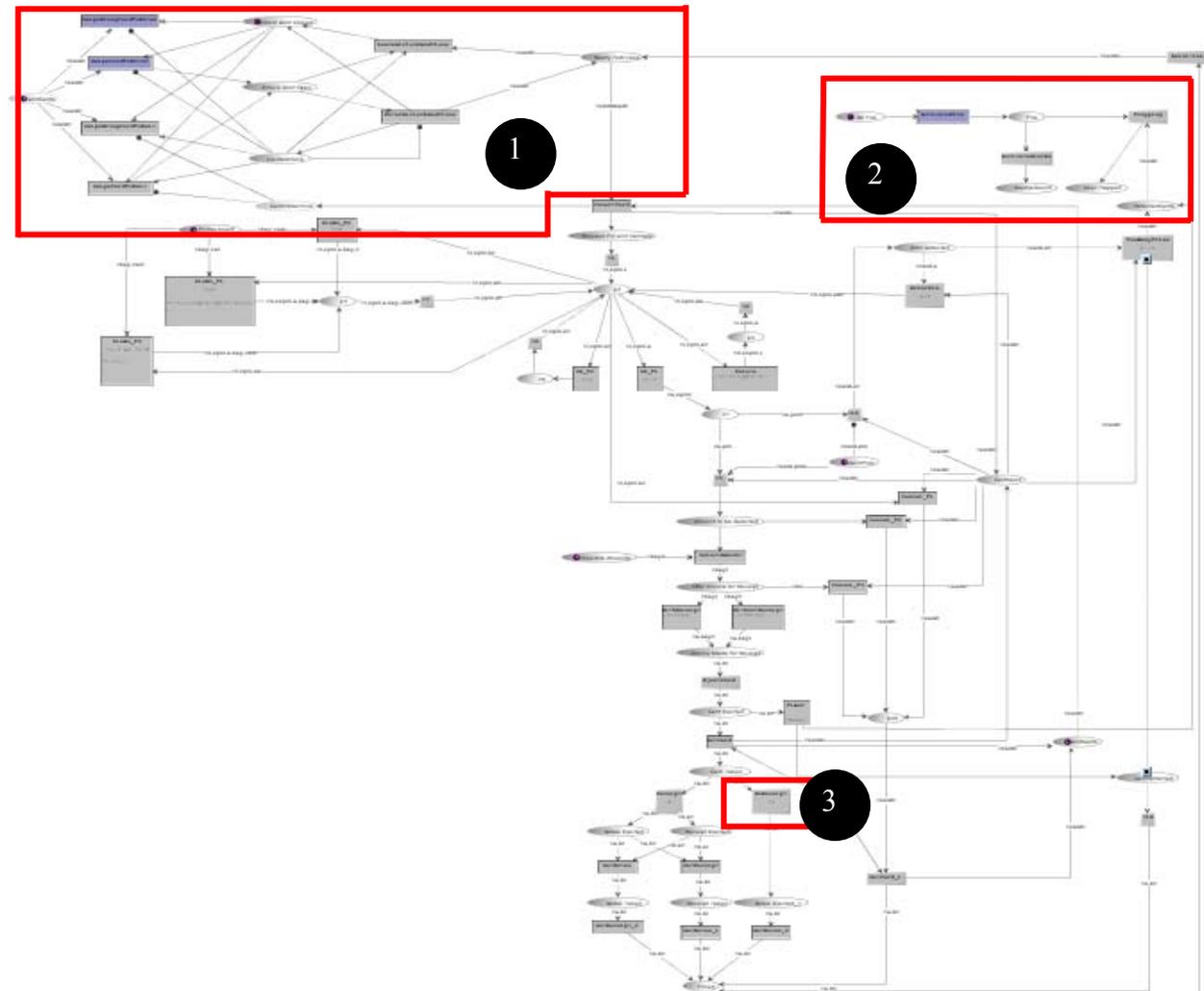


Figure 6-10. System Model Including Hazardous State “User Trapped”

3.3.2 Modification 1: Entering/Exiting the Airlock

The first modification to the system model is the airlock feature. This is label 1 in Figure 6-10. This part of the model has been extracted and reproduced in Figure 6-11.

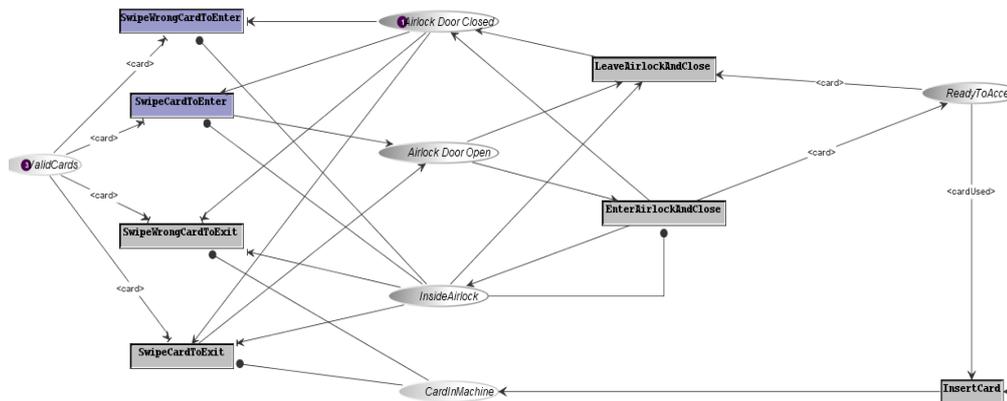


Figure 6-11. Airlock Feature of Informed System Model (label 1 Figure 6-10)

The airlock feature behaves as follows. Initially there is one token in place *Airlock Door Closed*, and 3 tokens in *Valid Cards*. Synchronised transitions *SwipeWrongCardToEnter* and *SwipeCardToEnter* are fireable. *SwipeWrongCardToEnter* represents the scenario when the user attempts to enter the airlock to access the ATM but with a card that is not authorised by the airlock. When this transition is fired, the system model remains the same, there is no impact.

The identification process of this erroneous user behaviour has not been explicitly presented in this thesis. Though entering authorised data, such as a card for an airlock or a PIN for an ATM is a common task. We therefore decided to make explicit the fact that the user may swipe the wrong card when attempting to enter an airlock. This is the same kind of error we identified in the previous chapter when analysing the subtask of PIN entry in a cash machine (Entering the wrong PIN). Many further types of erroneous behaviour could have been modelled though we have decided not to analyse them all and illustrate them all here since they are beyond the analysis we are presenting.

When the transition *SwipeCardToEnter* is fired, the token is removed from place *Airlock Door Closed* and one set in place *Airlock Door Open*. In this state, the only available transition is *EnterAirlockAndClose*, signifying that the user must enter the building and the close the door or allow the door to close automatically. When the *EnterAirlockAndClose* transition has been fired, a token is set to place *ReadyToAccess* and transitions *SwipeWrongCardToExit* and *SwipeCardToExit* become fireable. Place *InsideAirlock* continues to hold a token.

It should be noted that place *ValidCards* is a “virtual place”. This is a recent extension made to the ICO notation and Petshop tool discussed in the literature review. A virtual place is in effect a replica of its original place but without all of the connecting arcs. The purpose of virtual places is to improve legibility. The interested reader can see our recent paper presented at HCI Aero 2006 (Barboni et al. 2006b).

When place *ReadyToAccess* contains a token, the user can begin to use the cash machine as presented in the standard model in the previous chapter. Once the transition *InsertCard* has been fired, place *CardInMachine* receives a token. While place *CardInMachine* has a token, transitions *SwipeWrongCardToExit* and *SwipeCardToExit* are no longer available. This function operates by means of an inhibitor arc from the place to the two exit transitions. This represents the fact that the user cannot swipe their card to exit the airlock while their card is in the machine (here we assume that the user has one valid card). Again, place *CardInMachine* is a virtual place (the original place is label 3 in Figure 6-10).

The transition *SwipeCardToExit* sets a token in place *AirlockDoorOpen* and *InsideAirlock* from which transition *LeaveAirlockAndClose* can be fired. When this transition is fire, the model is effectively reinitialised with place *AirlockDoorClosed* containing a token, and transitions *SwipeCardToEnter* and *SwipeWrongCardToEnter* fireable.

This modelling feature has an assumption that should be made clear. We have assumed that the user will exit the airlock when they swipe the card to exit (i.e transition *SwipeCardToExit*). We have not modelled the possibility

for the user to stay in the building once they have swiped a valid card to exit the airlock. Though we are incorporating “erroneous” human behaviour, one must bare in mind that this is a system model and not a user model, and not all aspects of the user can be appropriately taken into account in a system model. The system model focuses mainly on the behaviour of the system. The complete behaviour of a potential end user should be represented in a dedicated model, such as a user model and/or task model. This would then cover the possibility of swiping a card to exit and not actually exiting the building, which could then be used to improve system design.

3.3.3 Modification 2: Fire, Alarm and Trapping Feature

Figure 6-12 presents label 2 of Figure 6-10, the fire and alarm feature of the informed system model. It is necessary to include the fire in order to run scenarios on the system model. Though the fire is not actually part of the system behaviour, it is a hazard that could potentially affect the system behaviour. For simulation purposes, the transition *ActivateFire* is synchronised. This is so that it can be fired by the analyst of the system model at any time to analyse its impact on the rest of the model. In this abstraction, the behaviour of the model is as follows. Place *No Fire* contains a token enabling *ActivateFire* to be fired at any time (there is no relation to any other state in the system model). When *ActivateFire* is fired, the token is removed from place *No Fire* and place *Fire* receives a token. Place *Fire* containing a token triggers the autonomous transition *ActivateAlarms* to fire (though this feature has no specific impact on the state we are interested in, which is when a user is trapped). Autonomous transition *Trapping* can only be fired when place *Fire* contains a token and place *RetainedCards* contains a token. As mentioned in the previous chapter when discussing the behaviour of the “normal” system model, place *RetainedCards* contains a token if either the user incorrectly enters their PIN three times, or if the user takes too long retrieving their card after it has been rejected. Thus, if the user has no card, and there is a fire, place *User Trapped* contains a token. This is the hazardous state we are interested in. While in this state, the transition, *SwipeCardToExit* cannot be fired, thus the door cannot be opened.

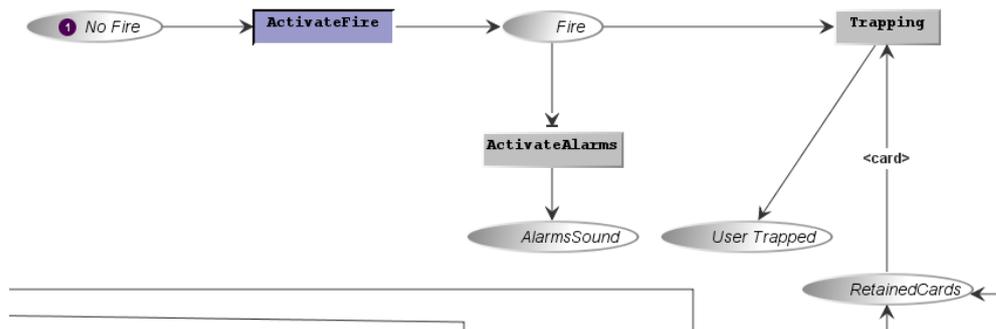


Figure 6-12. Fire and Alarm Feature of Informed System Model (label 2 of Figure 6-10)

3.3.4 Modification 3: Card In Machine Place

The final part of Figure 6-10 that we reproduce in higher quality is label 3, the original place *CardInMachine* to briefly discuss its behaviour and the significance of a virtual place. This place is connected to the synchronous transition *InsertCard* presented previous in Figure 6-11. The place receives a token when the card has been inserted into the machine. While it contains a token, transition *SwipeCardToExit* cannot be fired because of the connecting inhibitor arc (see Figure 6-11). The same place is connected to synchronous transition *GetCard* and autonomous transition *GetCard_1*. The first *GetCard* is fired when the user takes their card from the machine following a successful transaction, while *GetCard_1* is used to represent the user taking their card after a cancellation of transaction. Both transitions are fireable when place *CardInMachine* contains a token (i.e. when the transition *InsertCard* has been fired at the start of the procedure). Thus if either of these two *GetCard* transitions is fired, the place *CardInMachine* will no longer contain a token and transition *SwipeCardToExit* will become fireable because the user will then have the card in their hand.

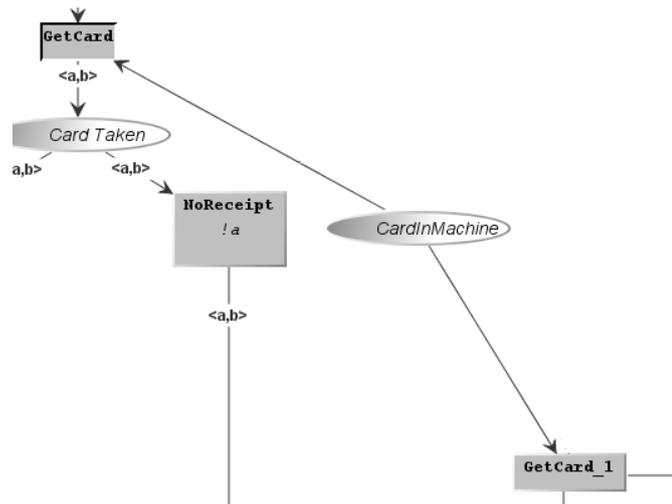


Figure 6-13. Card in Machine Feature of Informed System Model (label 3 of Figure 6-10)

We have presented extensions made to the system model in order to represent the hazardous state identified in the flawed safety cases. The following section illustrates how to systematically relate primary events from the ECFA to places, transitions and states in the informed system model to ensure we have accurately modelled the incident scenario.

3.3.5 Relating the ECFA to the System Model Including Hazardous States

Table 6-1 details the relationships between primary events modelled in the ECFA chart, and events in the Petri net systems model. As previously mentioned, the ECFA chart is read from left to right thus the table follows the same approach with ‘event 1’ in the table being the left most primary event etc. Since we only consider the primary events, the table does not include any conditions.

Table 6-1. Relating ECFA with System Model including hazard

ECFA Name	Number &	Event Type	Petri net Equivalent	Petri net Token Representation
1- User attempts to enter ATM airlock		User	Represented as user swiping card and being inside the airlock (or not).	3 x ValidCards, 1 x AirlockDoorOpen n x PinKeyboard, 1 x NoFire, 2 x ValidPins
2- User swipes card		User	Represented in system model	3 x ValidCards, 1 x AirlockDoorOpen n x PinKeyboard, 1 x NoFire 2 x ValidPins
3- Card is accepted		System	The airlock door opens	3 x ValidCards, 1 x AirlockDoorOpen n x PinKeyboard, 1 x NoFire 2 x ValidPins
4- User attempts to withdraw cash		User	Process of entering card, pin, selecting amount, receipt etc is modelled	Not listed here due to complex combinations of possible states
5- User enters incorrect PIN 3 times		User	Incorrect PIN entry modelled	Not listed here due to complex combinations of possible states
6- ATM retains card		System	Card retaining modelled	2 x ValidCards, 1 x AirlockDoorClosed 1 x InsideAirlock, 1 x CardInMachine n x PinKeyboard, 1 x NoFire 2 x ValidPins, 1 x CardRetained 1 x RetainedCards, 1 x CardUsed
7- Fire occurs in building		System	Fire represented	1 x Fire (in addition to current state of system)
8- Alarms sound		System	Alarms represented	1 x Fire 1 x AlarmsSound (in addition to current state of system)
9-User attempts to escape		User	Not represented in system model	
10- Card swipe system still activated		System	Modelled, card must be swiped to exit building even when there is a fire. Card cannot be swiped to exit building	2 x ValidCards, 1 x AirlockDoorClosed 1 x InsideAirlock, 1 x CardInMachine n x PinKeyboard, 2 x ValidPins 1 x CardRetained, 1 x RetainedCards

		when card has been retained.	1 x CardUsed, 1 x Fire 1 x AlarmsSound
11- User trapped	User	Modelled in the system as a place	2 x ValidCards, 1 x AirlockDoorClosed 1 x InsideAirlock, 1 x CardInMachine n x PinKeyboard, 2 x ValidPins 1 x CardRetained, 1 x RetainedCards 1 x CardUsed, 1 x Fire 1 x AlarmsSound, 1 x UserTrapped

Although the ECFA chart that we have produced depicts only the path of events that led to the accident, reverse engineering the process can produce numerous possible scenarios that could lead to the same accident. This is because there can be more than one way to reach a certain condition or system state which then resulted in a specific cause.

3.3.6 Marking Graph to Identify Further Scenarios Leading to the Same Incident

In some cases, there may be several scenarios of events that are useful for further analysis. In large complex models, it may be more useful to extract and analyse only the latter part of the chain. In the ATM example, we can assume, for the purposes of explanation that we are only interested in combinations of events leading to the last event “User Trapped” since this is the only hazardous state we have explicitly represented.

The use of these marking graph techniques and the ECFA helps to ensure that the system model captures the behaviour that was observed during an incident or accident. However, we may also have identified additional scenarios that could lead to *User Trapped*.

We are primarily interested in the sequence of events (transitions fired) that lead to a single token being present in the place representing this event, which we will call the ‘incident state’ (the distribution of tokens in the Petri net including 1 in the *User Trapped* place).

To date, we know from the ECFA that the incident state occurred when the user incorrectly entered their PIN on three occasions and had their card retained. Though the place *User Trapped* may also receive a token with another or several different sequences of events from that identified so far.

Because of the size of our informed system model, tool support is necessary to produce a marking tree and perform a marking graph analysis. As mentioned in the literature review, tools such as :

- ARP (<http://www.ppgia.pucpr.br/~maziero/diversos/petri/arp.html>).
- JARP (<http://jarp.sourceforge.net/us/index.html>)

have proved successful for us. The complete marking tree and marking graph will not be presented here as the process of performing such analyses has been presented in the previous chapter. Here we will discuss the procedure and use of tool support which allow us to identify and extract relevant scenarios leading to the incident state. Figure 6-14 provides a block diagram, based on the approach diagram presented previously in Figure 6-2 for the procedure proposed in this section. This figure makes explicit reference to the User Trapped hazard.

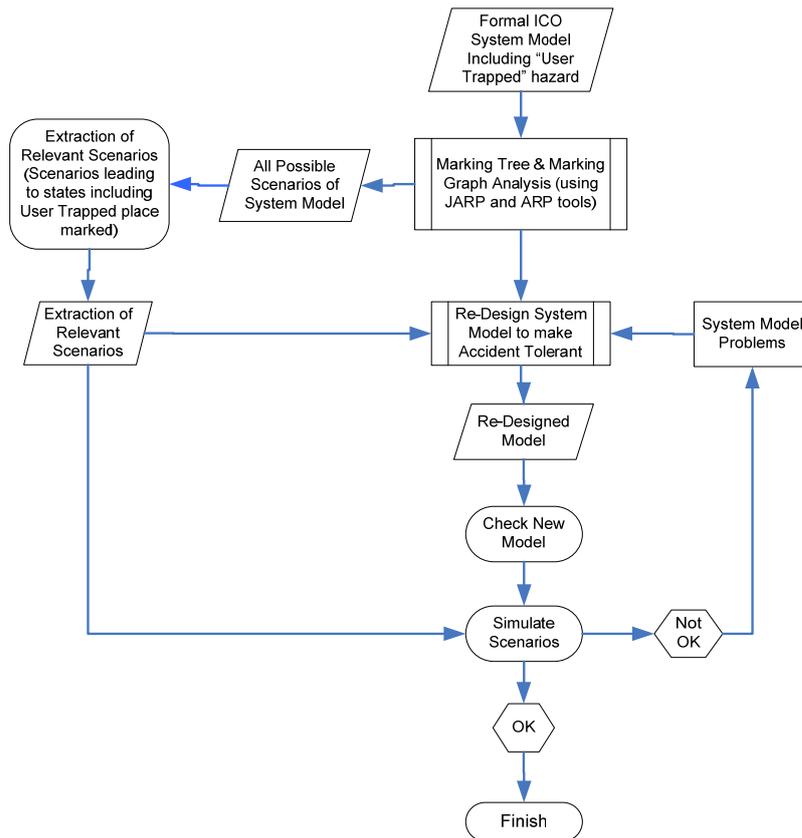


Figure 6-14. Improving the System Model to Avoid Hazardous State

The marking tree, consisting of an explicit list of all possible states must first be produced. This is possible using the ARP analysis tool. Following which, a textual marking graph can be produced. However, JARP and ARP do not accept preconditions, timing aspects, tokens with values, synchronised transitions; these are particular features of the ICO notation (and more generally speaking specific to high-level Petri nets models). The textual marking graph produced by ARP can be manually drawn into a graphical marking graph since the tool currently does not support graphical marking graphs.

Thus all states that include place *User Trapped* containing 1 token are extracted from the marking graph. We identify which transitions are fired in order to reach the incident state from non-incident states. These are the key transitions that will be used to redesign the system model. The identification phase allows us to identify further scenarios of events that could lead to a token being set in place *User Trapped* and to verify that our model accurately represents the actual incident scenario represented in the ECFA.

Although we do not present the full marking graph or scenario extraction here, we can assume that one additional scenario is possible for leading to the same incident state. Rather than the user incorrectly entering their PIN three times and having their card withdrawn, a second scenario maybe that the correctly perform the withdrawal transaction, but fail to take their card from the machine within 5 seconds from it being ejected (this is modelled in the standard ATM system model presented in the previous chapter). This additional scenario would be identifiable using the marking graph approach.

The original incident scenario, and any additional scenarios can be used to improve the informed system model including hazardous states to make this particular hazard non-reachable in the new design.

In our example, the redesign could take two forms. Either, once there is a fire, the swiping card system is automatically deactivated and the airlock door can be opened freely, or, allow the swiping card system to remain activated even when there is a fire, but not when the user's card has been retained. The first option is probably the most suitable in terms of safety, though to make the modelling aspect more interesting, we have chosen to model the second scenario. The following section presents the modification made to the system model to ensure that the user can exit the airlock even if their card has been retained.

3.3.7 Improving the System Model to Avoid the Hazardous State

We have presented the extensions made to the informed system model in order to deal with the hazardous state identified using the flawed safety cases and the ECFA. So far, the system model has been informed to make explicit the incident scenario but is still not safe.

According to the flawed safety cases that allowed us to identify and model a hazardous state (i.e. when there is a fire and the user has no valid card to exit the building) and the additional scenario identified from the marking graph analysis, the system model must now be adapted so that the same hazardous state cannot be triggered by a user or by the system in the future.

The system model must therefore be modified to avoid this hazardous state. We have improved the system model by ensuring that the airlock door is opened if there is a fire and if the user's card has been retained (we continue to assume that if the user's card has not been retained and there is a fire, the user must swipe the card to exit)

In our improved system model (an extract presented in Figure 6-15), designed to avoid the hazardous state, an additional arc has been connected from place *Airlock Door Closed* from transition *Trapping*. The place *Airlock Door Closed* in Figure 6-15 is a virtual place to avoid drawing arcs across the whole system model. When there is a token in place *Fire* and in place *CardRetained*, transition *Trapping* is fired, removing a token from place *Airlock Door Closed* and setting a token, in the new place *Open Door for Emergency*.

We could make further safety improvements by generally allowing users to exit the airlock without even swiping an authorised card.

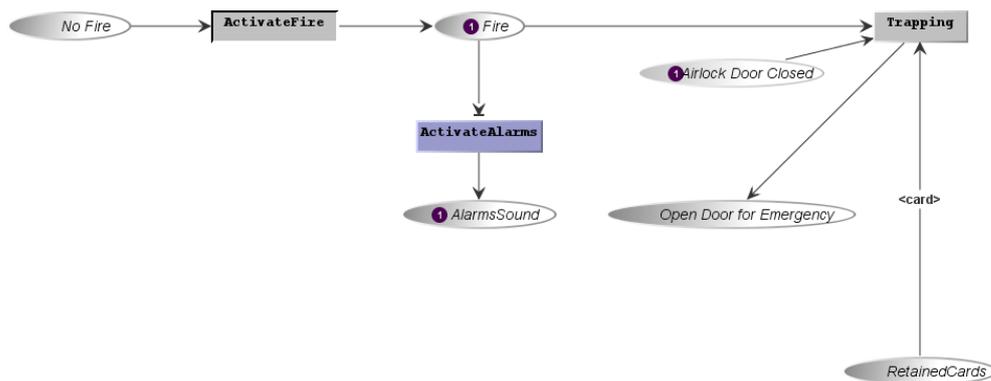


Figure 6-15. Modification Made to System Model to avoid Hazardous State

The incident scenario represented in the ECFA can again be tested on the improved system model to ensure that the same incident can no longer occur. This can be achieved by running scenarios on the Petri net using the Petshop tool.

4 Supporting Integration

This phase of the framework has been reproduced in Figure 6-16 annotating the sections and chapters that the approach has been demonstrated. The design of a system may begin from this phase. With safety cases or with an incident or accident that sparks redesign. The safety modelling phase plays an important role in the generic integrated modelling framework supporting many of the other phases. The integration of such information of course depends on the system under development, if the system is non-safety related or non-safety critical, it may not be relevant or cost beneficial to include such additional information in the design. However, this part of the framework feeds into the system modelling phase, task modelling phase and testing phase in the following ways. An incident or accident report can inform the system modelling in general, the error analysis of a system model, and the barrier analysis for informing a system model. Furthermore, from the safety modelling, data represented in the safety cases can be used to inform the human error analysis part of the task modelling phase, such that arguments presented in safety cases can be analysed and justified against the tasks that an operator may have carried out directly before an incident or accident. The incident and accident report, as shown in Figure 6-16 also provides data to perform an Events and Causal Factors Analysis which is then used to deduce accident scenarios, supporting the testing phase for model validation purposes. The research presented in this

chapter has been published in the following papers based on a safety-critical interactive system case study, (Basnyat et al. 2005c), (Basnyat et al. 2005b), and (Basnyat et al. 2005a).

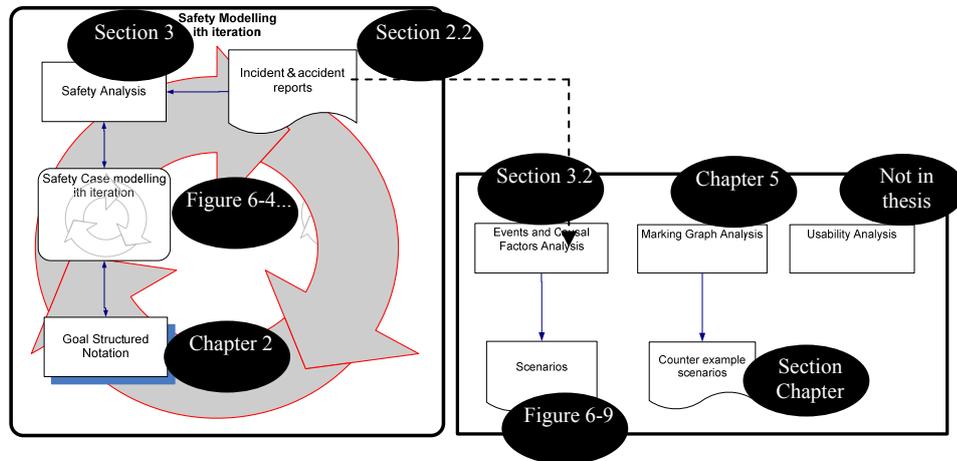


Figure 6-16. Safety Modelling and Testing Phases of the Generic Integrated Modelling Framework

5 Conclusion

The increasing complexity of many computer-controlled application processes is placing increasing demands on the investigation of adverse events. At the same time, there is a growing realisation that accident investigators must consider a wider range of contributory and contextual factors that help to shape human behaviour in the causes of safety-related incidents.

In order to support the construction of reliable interactive systems it is critical to provide ways of learning from experience when designing a new system. This is one of the aims of the approach presented in this chapter. We cannot, however, provide automatic means of redesigning the system to prevent any recurrence of an accident or incident. Our approach, in contrast, provides tools, methods, techniques and notations to the designers to help them check whether the new model makes the re-occurrence of the accident/incident impossible.

We have described in this chapter, a method for a multidisciplinary perspective to accident investigation and informing the system model. The approach involves two disciplines, incident and accident investigation, and system specification. From an incident and accident investigation perspective, Safety Cases and Events and Causal Factors Analysis charting has been used. The latter technique has been widely adopted, for instance by organisations as diverse as NASA, Eurocontrol and the US Department of Energy. The results of this initial analysis are then used to inform more detailed systems modelling using Petri Nets. It is important to stress that other investigatory tools might have been used just as there is a broad range of alternate techniques for systems modelling.

Our approach is driven by the information contained in an incident or accident report. These documents provide a convenient starting point for the construction of Safety Cases and ECFA diagrams, though in most cases Safety Cases would be produced before system implementation and before an accident and would therefore be obtainable in an incident or accident investigation.

There is, however, a danger that our approach will bias redesign work towards the particular causes identified in an official report rather than the wider contextual factors that can easily be overlooked in the aftermath of an accident or incident (Koester 2001). It is apparent that our integration of accident investigation and detailed systems design techniques can only form one part of a wider approach to redesign in the aftermath of adverse events.

It is clear that some events, conditions and actions cannot easily be represented in the system model. These would include human cognitive processes for example. In contrast, these events are more conveniently considered in the natural language annotations of the investigation techniques rather than in the places and transitions of a Petri Net. For this reason, other techniques, such as task modelling, with additional human error

analysis presented in Chapter 4 complements the approach presented here and can help to reveal different perspectives on the accident.

This chapter has provided steps towards the use of accident investigations to inform system modelling and verification. We deliberately chose to integrate approaches that are pitched at different levels of abstraction. The accident analysis focused on a novel application of the GSN technique for the development of Safety Cases. This decision was deliberate because GSN provides a means of representing the implicit assumptions that are often difficult to capture in formal models of interactive systems. For instance, previous diagrams have represented assumptions about component interaction conflicts and component maintenance.

The decision to use GSN in the preliminary stages of this work allows high-level analysis that complements detailed system modelling. Though there is no automatic translation from the insights provided by the application of GSN to the properties that we might want to verify using the ICO models. An event based reconstruction technique such as Events and Causal Factors analysis, promoted by the US Dept of Energy provides a relatively automatic translation from this analysis into the places and transitions of the systems models, which is why the two techniques complement one another. This is the subject of current research where, for example, task analysis has been used as a stepping-stone towards the identification of theorems to assert against particular system models (Leveson 2004). We can take high-level safety goals from the GSN, such as the need to ensure effective maintenance. These can be broken into a large number of different sub-tasks. We can then analyze the systems support that is available for each of these subtasks. However, it is unlikely that we will ever be able to automate the process of deriving putative theorems from the often highly contextual and situated observations of accident investigations.

The second stage of our approach relies upon system modelling using the underlying Petri Net formalism embedded within the ICO approach. This offers numerous benefits. Firstly, the process of translating high-level observations from the safety case analysis helps to ensure that the meta-level arguments about system failures can be instantiated in terms of specific subsystems. In other words, the construction of the systems model helps to validate technical assumptions in the accident report. If we cannot develop a coherent system model then there may well be important details that have been omitted from the account of an accident. Secondly, the development of the system model provides an abstract representation that is amenable both to animation and various forms of proof. This enables us to go beyond the specific scenarios of an individual accident. We can begin to experiment with new designs to ensure that they would not suffer from the same problems as previous designs. In other words we can conduct various forms of reachability analysis to determine whether a new or revised system might reach the same unsafe state. This style of analysis offers other benefits because we can begin to look for other transitions that were not observed in the particular accident being analyzed but that might lead to the same undesired outcomes in future accidents.

By applying such development techniques, the validation of the safety-critical system appears earlier in the design process than in more classic techniques. A formal validation occurs through the use of verification techniques and analysis tools. This validation is possible due to the algebraic foundation of the Petri nets. This simulating task is fully supported by PetShop environment through actions on the user interface simulating the real actions on the real system.

Chapter 7

Barriers

*Intellectuals solve problems, geniuses prevent them.”
(Albert Einstein, 1879 - 1955)*

Chapter Summary

Barriers may be considered as part of the safety modelling phase. Since we are mostly interested in informing the system model, we separate barrier analysis and modelling for the purposes of explaining the approach and contribution. In the generic integrated modelling framework, barriers are represented within the system modelling phase. In this chapter we present our approaches on identifying, analysing, classifying and modelling barriers identified post-incident/accident and pre-incident/accident. The barriers we are particularly interested in are technical barriers, human barriers, socio-technical barriers and software barriers.

1 Introduction

Today, safety has become paramount in the design and operation of many technological systems. Often such systems present hazards that cannot be easily eliminated and therefore these systems become safety critical. To mitigate the risk caused by the potential consequences of these hazards, risk reduction must occur for the system to be safe enough to be accepted by society. The means of risk reduction are usually dedicated safety systems that stop the evolution of scenarios leading to unacceptable consequences.

Risk reduction is a key factor in the design of safety critical systems and in assessment of their operational safety. It is achieved either by preventing hazards or by protecting against hazards. Prevention typically involves design modifications of the total system, including for example operating procedures. Protection involves the design of additional systems, which embody barriers that fend against adverse events, damage or harm (Schupp et al. 2004).

To avoid double use of the word system, we will refer to safety systems as simply barriers. When we refer to ‘the system’, this reference is made to the system that is being designed and operated as a whole, for instance the plant, aircraft or computer system. Several definitions for barriers exist. For example,

- “A barrier is an obstacle, an obstruction, or a hindrance that may either (i) prevent an action from being carried out or an event from taking place, or (ii) prevent or lessen the impact of the consequences, limiting the reach of the consequences or weakening them in some way” (Hollnagel 1999b).
- “The combination of technical, human and organizational measures that prevent or protect against an adverse effect” (Schupp et al. 2004).
- “Barriers represent the diverse physical and organisational measures that are taken to prevent a target from being affected by a potential hazard” (Johnson 2003).

A discussion of barriers has been presented in the literature review. Barriers are usually regarded as systems that prevent or stop an undesired consequence. The ability to stop is important here, and defines the scope of what the barrier is. For instance a fire extinguisher is not a barrier itself, as it has to be operated by a human who must have received some training, and it must be in an easily accessible place. These elements are part of the barrier too. While systems and barriers should be independent to a certain aspect, they will often share components.

Commonly there is not a one-to-one relation between hazards and barriers. One hazard may be protected against by several barriers (see Section 4.3) and one barrier may feature in the mitigation arguments of several hazards.

During the life span of the system, usually during late engineering and early construction stages, barriers are designed and integrated with the system specifically to reduce risk.

Barrier analysis has been proposed as a useful means of identifying the ways in which particular defences either did or did not protect a target system from particular hazards (Johnson 2003). An important benefit of this approach is that it can be applied both to systems and to operators. For instance, barriers can include the testing

and redundancy techniques that are intended to protect complex computation systems in command and control applications. Equally, barriers can involve the application of Crew Resource Management training to help minimise the scope for ‘team based error’. These barriers can include the knowledge and competency of system operators however this may be stretching the definition of a barrier too far, and is often rejected.

In this chapter we will deal with two main kinds of barriers, a special but very common category of barriers, those that are socio-technical and secondly, software barriers.

A **socio-technical barrier** is essentially a combination of hardware and software, but also depends on human action for it to function correctly. The barrier thus assigns safety critical tasks to human operators who therefore become crucial in maintaining system safety. As these barriers are socio-technical, the tasks involve interaction with system software and hardware.

The task of the operator appears often hard to integrate in system design, and may occur too late (Daouk and Leveson 2001) whilst the technical part of designing a socio-technical system is often systematically addressed and supported by notations and tools. These integration difficulties may lead to operators not being aware that a task is safety critical, which can obviously cause accidents. It is the specification, analysis, verification and documentation of the safety critical human tasks that we are interested in when we identify, analyse and model socio-technical barriers.

A **software barrier** (depending on when it is implemented) a feature, patch or modification to software code in order to prevent hazardous human-computer interaction occurring. Though our approach and goals of this thesis are aimed towards interactive applications, we are not concerned with the software directly. Our software barriers will be implemented on system models that describe the behaviour of an interactive application.

In this chapter we outline an approach that facilitates the identification, analysis and modelling of these types of barriers. Most importantly, we simplify system analysis by explicitly defining barriers, analysing how these function, and only subsequently integrating them in the system, instead of directly trying to analyse the system as a whole.

Figure 7-1 illustrates, as with the previous approach chapters, how this part of the approach fits into the generic integrated modelling framework. We consider the barrier modelling part of the system modelling phase with input from the safety modelling phase discussed in the previous chapter and expanded in this chapter.

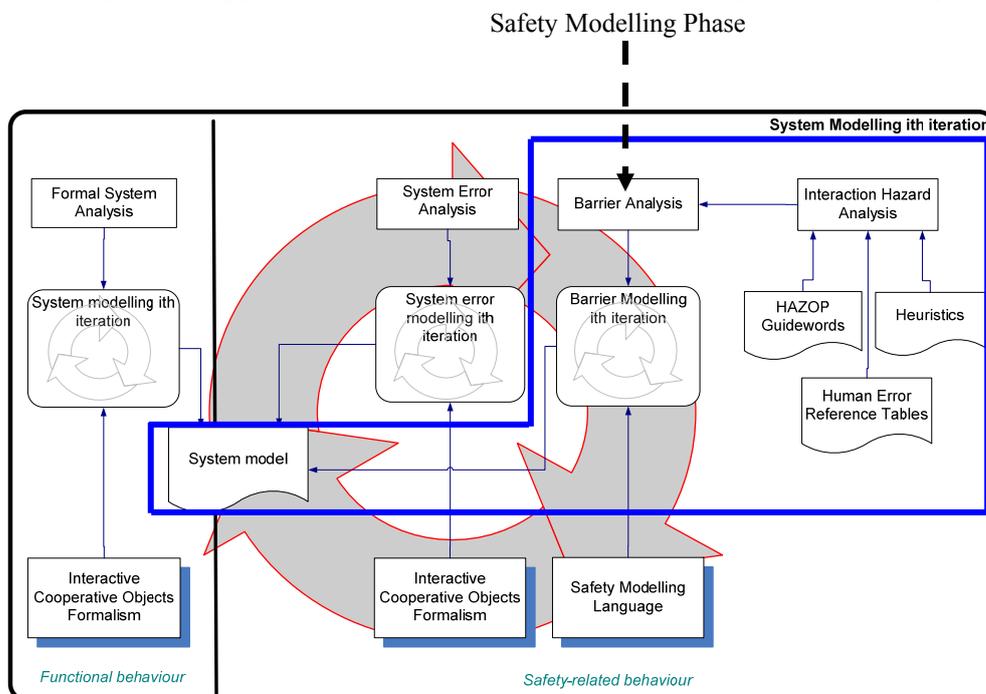


Figure 7-1. Barrier Analysis and Modelling With Respect to the Generic Integrated Modelling Framework

1.1 Methods for Identifying Barriers (pre and post incident/accident)

We propose two techniques for identifying barriers and modelling them. The first is from a post-incident or accident perspective. We suggest that incident and accident reports, as well as safety cases (particularly flawed safety cases such as those discussed in the previous chapter) may contain information relating to barriers and failed barriers. The barriers may be hardware, software or socio-technical. They can be extracted, analysed for reasons of failing and then modelled for integration with the system model. Barrier analysis is often associated with accident analysis in aiming to determine which barriers failed, why they failed and how to rectify the problems. “Barrier analysis starts from the assumption that a hazard comes into contact with a target because barriers or controls were unused or inadequate” (Johnson 2003) p.355).

The second approach is from a pre-accident perspective, in which we suggest that an interactive application can be analysed in order to identify hazardous human-computer interaction and then suggest and model software barriers to improve the system model.

Our approaches combine previous work in the field of barrier analysis and modelling with model based design of interactive systems. The barrier analysis can inform the system model by providing recommended technical and human-related barriers such as restrictions on operating techniques.

The following sections present the approaches, from the two perspectives on the ATM Cash Machine example as in previous chapters. Towards the end of this chapter, we will show how the barriers can be used to inform and be integrated with the formal system model.

2 Post-Incident Barrier Identification

Barrier Analysis has been widely used to identify the way in which defences either protected or failed to protect a target system from potential hazards. Many barriers fail from common causes, including misconceptions that can be traced back to early stages in the development of a safety-critical system. For instance, unwarranted assumptions can be made about the impact of training on operator behaviour in emergency situations. Similarly, Barrier Analysis can also be used before a system has been designed to inform the system model and make it more tolerant to errors by incorporating human and technical barriers into the design.

Barrier and task analysis has been widely used to represent and reason about human behaviour in the aftermath of an adverse event. Most of these complementary techniques focus on identifying the ways in which defences can be undermined. They explain what happened to expose a system to a potential hazard. This is important given the increasing complexity of safety-critical interactive applications. However, it is only part of the story behind any particular accident or near-miss incident. In addition, investigators must account for the reasons why an adverse event occurred. This is a difficult task. For example, several independent barriers might fail from a combination of different reasons. These might include pathological combinations of system failure and human ‘error’. Equally, however, there can often be common causes that explain why multiple barriers fail to protect an application process. These common causes often include under-investment in safety-systems or an over confidence about operator performance under emergency conditions in a command and control application.

From an incident and accident investigation perspective, our approach aims to assist investigators determine the reasons behind the failure of a barrier system to mitigate a hazard. We use barrier analysis because we are investigating the failure of safety-critical systems, where ‘barrier’ elements are clear and structured, such as alarms, sensors and so on. There is an abundance of accident analysis techniques used to identify root causes, however we are interested in understanding how the specific system components did not perform as they were intended to, thus having a more focused perspective on the accident.

However, Smith et al., (Basnyat et al. 2005c; Smith et al. 2004) argue that the representation of such barriers are commonly absent from safety case documentation and the associated arguments for compliance to particular standard.

We present in this section an approach for using safety cases to identify simple barriers that can be used to inform the system model to improve safety for the redesign of the system or for future systems.

2.1 Using Safety Cases to Identify Barriers

The intention is to show that if an accident involved the failure of multiple barriers, it is also possible to trace the common causes of those failures back to the assumptions and arguments that are embodied within a Safety Case.

Referring back to the safety cases presented in the previous chapter, a number of barriers were mentioned in the hypothetical safety cases that were developed for our ATM example in an airlock. These included:

- Sensors
- Alarms
- Emergency exit signposts

It should also be stressed that barriers can also include the knowledge and competency of system operators, which suggests that human activities such as training and maintenance are very important for the success of certain barrier systems (in more complex safety critical interactive systems than an ATM in an airlock).

Table 7-1 provides an example of a simple Hazard-Barrier-Target (HBT) analysis following the identification of barriers from the Safety Cases. System users are considered as targets that will be affected by a potential hazard (a fire). The barriers that protect these targets include the sensors, alarms etc.

Table 7-1. Hazards, barriers and targets

Hazard	Target	Barriers
Fire	ATM users/clients	Sensors, alarms, emergency signposts

Following this, we proceed with a barrier analysis in order to identify the way in which potential hazards threaten the functional components of a safety-critical application. The investigation, typically proceeds through the elaboration of a number of high-level tables such as those illustrated in this section.

Barrier analysis proceeds by analyzing potential reasons why each of the barriers may have failed. This can be done at a relatively high-level abstraction during the early stages of an accident investigation so that analysts do not prematurely close down a particular avenue of investigation into an adverse event.

Table 7-2 represents the results of such a preliminary application of the approach to our ATM example.

In our example, we have not elaborated on the operation of the sensors and alarms in our incident scenario. For the purposes of illustration, we assume here that the sensors and alarms failed.

Table 7-2. Detailed barrier analysis and correspondence to Safety Cases

Barrier	Reason for failure	Safety Component (See Chapter 1)	Case (See Chapter 1)
Sensors	There were no smoke sensors in the airlock area, though smoke detectors were located in adjacent rooms.	G1.1.1	
Alarms	The alarm sounded, but later than if there would have been smoke detectors in the ATM airlock area.	G1.1.2	
Emergency Exit Signposts	Did not fail	G1.4.1	

Investigators can proceed by examining the evidence that these arguments rest on. In this case, suspicion falls on the guidelines suggesting that the card swiping system used to enter and exit the airlock should remain in operation even in emergency situations. In our incident, this design option caused the user to remain trapped in the airlock because the designers and safety standard guidelines did not consider the possibility that a user may no longer have a card once inside the airlock.

The barriers identified from the safety cases can be modelled and added to the system model. Though we did not explicitly mention so in the previous chapter, our system model already included barriers including the hazardous state modelled the fire, alarms and sensors. Thus in some respects, these barriers have already been modelled (simplistically) and will not be reproduced or discussed further in this chapter. We could expand on the

way in which we have modelled the sensors and barriers in the system model to include timing issues, with respect to the time in which sensors sense smoke from when the fire is started etc, though illustrating this will not be any more fruitful for discussing our approach and techniques.

2.2 From the Incident/Accident Report

A second technique, from an incident and accident investigation perspective, is to extract barriers directly from the incident report and not from safety cases. The description of the incident may reveal further types of barriers that existed in the system that perhaps contributed to the incident or failed directly. In our simple hypothetical example of an ATM airlock incident, we do not have an incident report and are unable to demonstrate how this can be applied thus for the purposes of explanation, we will assume that since the user was trapped in the airlock, partly because the exit door required the user to swipe the card (also because there was no smoke detected in the airlock area), we will build on this to identify a new barrier.

We continue to assume that the safety case argued that the card swiping system met standard guidelines for safety, still requiring a card to be swiped in an emergency situation for illustrative purposes of our approach.

Here, we could argue that the safety of the system could be improved by implementing a technical barrier such that when there is a fire and the user has no card, the card swiping system will deactivate. This barrier builds upon an existing system (even though it is probably not the safest suggestion, removing the swiping system completely in emergencies would be safer!).

The previous chapter, which extended the system model to include the hazardous state and was then modified to improve the safety of the system by changing the behaviour of the card swiping system in emergencies implemented the barrier we suggest here. The modification to the system was not described as a barrier in the previous chapter because we were simply discussing an approach for including hazardous states in the system model and improving the design according to identified hazards. Here we are explicitly interested in barriers, and can now refer to our modification as a barrier.

Examples of socio-technical barriers that could be identified and extracted from an incident or accident report could be a plant operation procedure in which the operator failed to act in accordance to manufacturer guidelines. The procedure, which must be carried out by human interacting with a system, which must be carried out in order to avoid a particular situation, is a socio-technical barrier. Another example is a fire extinguisher which requires human intervention before it becomes a barrier to protect a person from fire and smoke. Though we are more interested in barriers that involve direct interaction with a system.

Until now, we have provided a description of how barriers can be identified, from an incident and accident investigation perspective using safety cases and the incident or accident report itself. The following section will illustrate, in more detail, the procedure and notations used to support the modelling of such barriers.

3 Approach to Post-Incident/Accident Barrier Analysis and Modelling

Here we discuss in more detail the tools and techniques that have proved successful for the analysis and modelling of barriers post-accident.

The approach employs a formal description technique to provide non-ambiguous, complete and concise models, thus giving an early verification of some potential problems to the designer before the new application/system is actually implemented. However, as we have previously mentioned, formal specification of interactive systems often does not address the issues of erroneous behaviour (human and technical) that may have serious consequences for the system. We believe our approach for identifying, analysing and modelling barriers can help improve future designs.

Figure 7-2 presents a graphical overview of our 3-step approach to barrier analysis and modelling, with a view of improving system safety in the redesigned system.

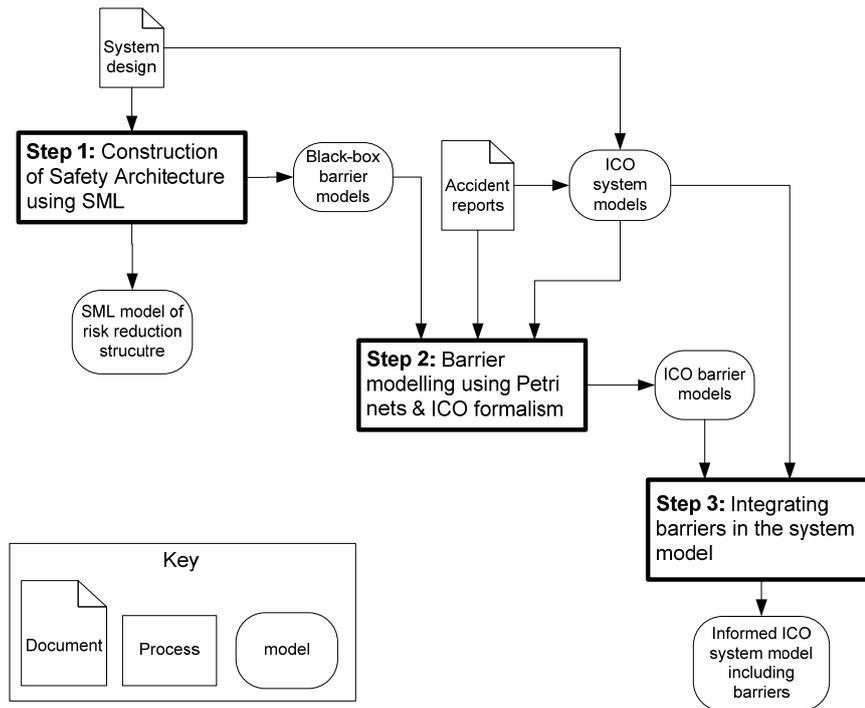


Figure 7-2. Approach Diagram

Step one uses the Safety Modelling Language (SML) (see section 3.1) to identify a structure which could achieve risk reduction (Schupp et al. 2004). This structure makes up the safety architecture of the system. Here the specific hazards are analysed, and barriers are devised that can prevent targets (e.g. workers, environment) from being affected by these hazards. In this first step, barriers are treated as black boxes, it is specified why they are in the system, not how they function. Designers are thus supported in reasoning about risk reduction conceptually. The first step has as an input, the system design. This information may comprise of incident and accident reports, design criteria documents such as Questions Options and Criteria (QOC), prototypes etc. The outputs of the first step are the SML model of risk reduction structure and the black box barrier models.

In **step two** each individual barrier is analyzed, designed and modelled. Often various techniques are required to achieve this. Our approach employs the ICO formalism to model and analyse the mechanisms of the barrier, their specifications and to verify their functions. The result of this step is a full design of the barrier which will achieve the safety function as specified in step one. This may use various parts of the system, hardware, software and human to achieve the required safety function. The approach diagram in Figure 7-2 shows that the “ICO system models” are informed by the system design documents and the accident report (as discussed and illustrated in the previous chapter on safety modelling). The output of this second step is the ICO barrier models.

In **step three** the functions specified by each individual barrier are connected to the system model as a whole by integrating the barrier into the system. The result is an informed ICO model including barriers.

With respect to socio-technical barriers and the ICO notation, we argue that an operator may have a number of functions some of which are specified within a barrier. Using this mapping, of operator tasks with tasks required for barrier functioning, it becomes clear which of the operator’s tasks are specified by which barrier, and therefore which specific tasks are safety-critical.

Depending on the system under consideration, it is useful to graphically represent the system at a level of abstraction that includes the various components of the system, including human operators if necessary and current barriers that may be in place. For a software application, (see Section 4.1) this may not be necessary since interaction with the application may be sufficient for analysis and modelling, especially with support from the ICO notation and Petshop tool which allow simulation of the system model while providing rendering on the interface. However, if several software applications are operating with strong connections between them, a graphical representation may be useful.

We suggest using a Piping and Instrumentation Layout Diagram to semi-formally model an abstraction of large

physical safety-critical interactive systems. Additionally, the system model (described using a dialect of high-level Petri-nets) allows us to reason about the system and to check conformance with the other models (task model, safety case and barriers).

3.1 Safety Modelling Language

The first step of our approach uses Safety Modelling Language. In short SML (Schupp et al. 2004) uses the Hazard-Barrier-Target (H-B-T) model to model the safety architecture of a system. The H-B-T model assumes that targets are vulnerable to the effects of hazards, and that targets can be protected against these effects by barriers. In some respects it is similar to other barrier models, such as the accident evolution and barrier function model (Svenson 1991), and the ‘Swiss-cheese’ model (Reason 1990).

In Figure 7-3, a basic example of a SML diagram is shown based on our ATM airlock incident. It shows that the fire is hazardous to ATM users. However the user is protected (to a certain extent) by smoke detector sensors and by fire alarms. Assuming these were well placed and fully operational, they would give most users enough time to exit the airlock.

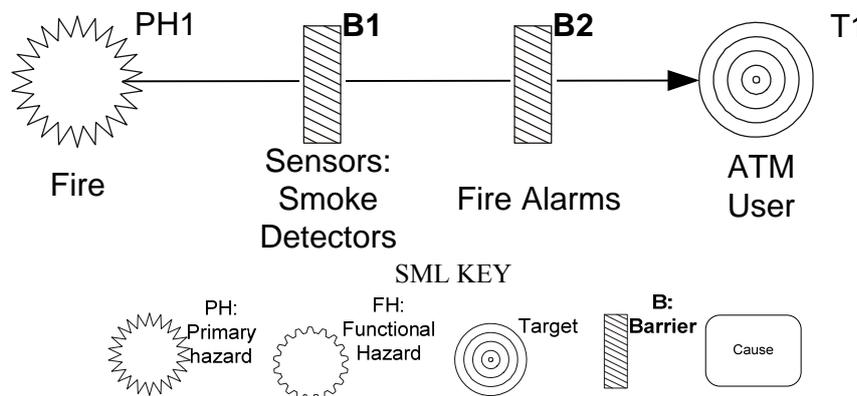
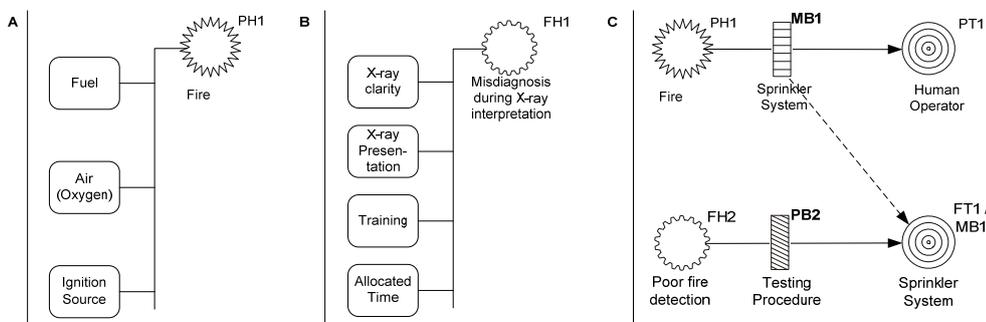


Figure 7-3. A Typical H-B-T Diagram for the ATM Airlock Example Including SML Key

SML models hazards in a more complex manner than the basic H-B-T model in Figure 7-3. A hazard is something that has the potential to cause an adverse effect to a target. A hazard is a ‘label’ that humans apply to complex phenomena perceived as hazardous. SML does not provide insight into the hazardous phenomenon itself but into the relations this phenomenon has with the rest of the design/system. It is modelled using two components: Causal elements that provide a link to the mechanism of the hazard, and effects, that provide the link to the targets. For instance, when the elements ‘flammable substance’, ‘oxygen’, and ‘ignition source’ are present in a design, these will cause a fire hazard, having heat radiation, smoke and high temperature as effects. This is shown in Figure 7-4a.

An example of a human factors related hazard is a misdiagnosis in interpreting an X-ray photograph in a medical domain. This can for instance be caused by the causal elements ‘training’, ‘available time’, and issues such as ‘X-ray clarity’.



Key: PT: Primary Target, FT: Functional Target, MB: Mitigative Barrier, PB: Primary Barrier

Figure 7-4: SML representation of (a) fire hazard, (b) a human error hazard, (c) recursion

To model the failure of barriers, SML defines *primary* (PH) and *functional* (FH) hazards. Primary hazards cause

direct harm to humans, neighbouring installations, and the environment. The barriers in between primary hazards and primary targets are called primary barriers. Functional hazards are phenomena due to either human factors or other causes that adversely affect other barriers, thus making these fail. Poor fire detection causes a sprinkler system to become inoperable, a testing procedure protects against this, as shown in Figure 7-4c. In this way, a risk reduction problem is defined recursively; when a barrier is used, it can fail due to a functional hazard

A consequence of this is that the list of primary hazards quickly provides insight into why the systems' safety is critical. Next, accident mechanisms, and the role humans play in these can be understood via recursions. This and many other aspects of the language such as the different kind of symbols and barriers are not further explained in this paper though Figure 3 shows some.

Whilst SML helps to define the barriers and their role in risk reduction, we need to understand which tasks are defined by these barriers, and how they must be integrated in the system. SML is not helpful here. As shown in the previous chapter, when modelling the modification to the card swiping system, we use the ICO formalism based on Petri nets to achieve that. The ICO barrier models built using the SML model represents both human and system behaviour, thus allowing task analysis. In addition to the advantages we have previously mentioned of using formalisms, is that we can verify that the barrier will function correctly.

4 Pre-Incident Software Barrier Analysis and Modelling

In the aviation industry, cockpits are becoming increasingly interactive with the introduction of the keyboard cursor control unit (KCCU) combined with graphical display units (DU) to be embedded in the Airbus A380 for commercial flights in 2006 and later in the Boeing 787 aircraft.

There exists extensive literature on types of human computer interaction failures that can occur while a user is operating an interactive system. Examples include mode confusions which are a kind of automation surprise (Rushby et al. 1999). Evidently, the consequence of such "errors" in a safety-critical context can potentially be devastating. The roots of such "errors" are often the result of poor design. Typically, there is a mismatch between the designer's conception of the system and the user's interpretation of the system. From an organizational perspective, such "errors" could also be a consequence of poor training, management etc.

Ideally, the possible existence of these "errors" should be considered throughout the design process, as early as during requirements gathering to mitigate them and minimize development costs. Techniques for analyzing software safety such as Preliminary Hazard Analysis (PHA) including Fault Tree Analysis (FTA) and System and Subsystem Hazard Analysis (SHA and SSHA) including deviation analysis such as Failure Mode Effects and Criticality Analysis (FMECA), mode confusions analysis and state machine hazard analysis are targeted for such early analysis (Leveson et al. 1997b).

The identification of potentially dangerous software aspects can be used to inform changes to the design, and training procedures etc. With respect to the design, such modifications could be considered as software barriers.

While barriers have been used quite intensively in the field of hardware and socio-technical systems (Hollnagel 1999b) their use in the field of software systems remains very limited as for instance in (Leveson 1991). The intrinsic nature of interactive software systems that involve users in their daily operation require the same level of reliability but, surprisingly, a barrier approach has not been applied to this area so far. This might be related to the fact that interactive applications have been kept away from command and control safety critical areas or to the fact that redesigning the system has been considered as a more adequate alternative than integrating barriers to the extant system.

In addition to the definitions of barriers we have seen so far, Leveson (Leveson 1991) makes a distinction between three types of barriers called lockout, lockin, and interlock, respectively (see Chapter 2 section 3.1.2.2). These will be used to classify our suggested barriers in order to select barriers directly related to the system model. In addition to this classification, we also refer to Hollnagel's barrier system classification in which he defines physical, functional, symbolic and incorporeal barriers (see Chapter 2 section 3.1.2.2).

This phase of our generic integrated modelling framework on barrier analysis and modelling focuses on software barriers and presents a complementary approach for the systematic identification of software-related user interaction hazards and barriers and modelling of barriers. The basis of sound design for a safety-related system is the identification, through systematic analysis, of the hazards which the system might encounter in operation as claimed in (Falla 1997). The proposed approach provides systematic ways to deal with the increase of

reliability of safety critical interactive applications.

The design of a usable, reliable and error-tolerant interactive safety-critical system is a goal that is hard to achieve but can be more closely attainable by taking into account information from previous usages of the system. One such usually available and particularly pertinent source is the outcome of an incident or accident investigation. Designs of any nature can be improved by taking into account previous experiences, both positive and negative as we have seen in the previous chapter and in the first half of this chapter.

However, it is not always the case that an accident report will be available, especially when new technology is involved. With a more pro-active perspective, we aim to analyse a safety critical interactive applications, for which we have no accident data. We aim to identify flaws in the application and suggest software barriers that could minimise potential erroneous user interaction with the system.

In previous chapters, as part of our generic integrated modelling framework, we have focused on task models and error analysis of sub-tasks to identify potential for the cash machine PIN entry example. On safety modelling of design flaws in the previous chapter and in the first half of this chapter technical and socio-technical barrier identification, analysis and modelling. We argue that although it is important for human “error” analysis and consideration for human factors to be included throughout and in early stages of the development process, further problems can occur during the interaction with the system.

4.1 Approach to software barrier analysis and modelling

In addition to the skill-based Human Error Reference Table (HERT) defined and exemplified in Chapter 4, we apply a typical heuristic evaluation to the UI of the ATM example and a Hazard and Operability Studies (HAZOP) analysis. The identified interaction hazards are used to propose socio-technical and technical barriers to mitigate these hazards. The proposed barriers, which may take the form of improved training, software implementation modifications, or UI design improvements for example, are also categorised according to Leveson’s (Leveson 1991) and Hollnagel’s (Hollnagel 1999b) barrier classifications. By doing so, the barriers can be filtered and represented in their relevant design model. For example, barriers relating to the behaviour of the software can be represented in the system model while barriers relating to improved training could modify the operator’s task and thus be modelled in the task model. Barriers relating to warning signs, personal protective equipment or fire extinguishers for example should be represented in a dedicated model.

In this part of the barrier analysis and modelling phase, we focus on barriers related to the behaviour of the system. The relevant software barriers are extracted from the identified list and are modelled using the ICOs formalism.

Figure 7-5 presents a block diagram describing the process of identifying, analysing, extracting and modelling software barriers. The shaded blocks represent external data, such as the HERTs, HAZOP and Heuristics that are used as input for the analysis.

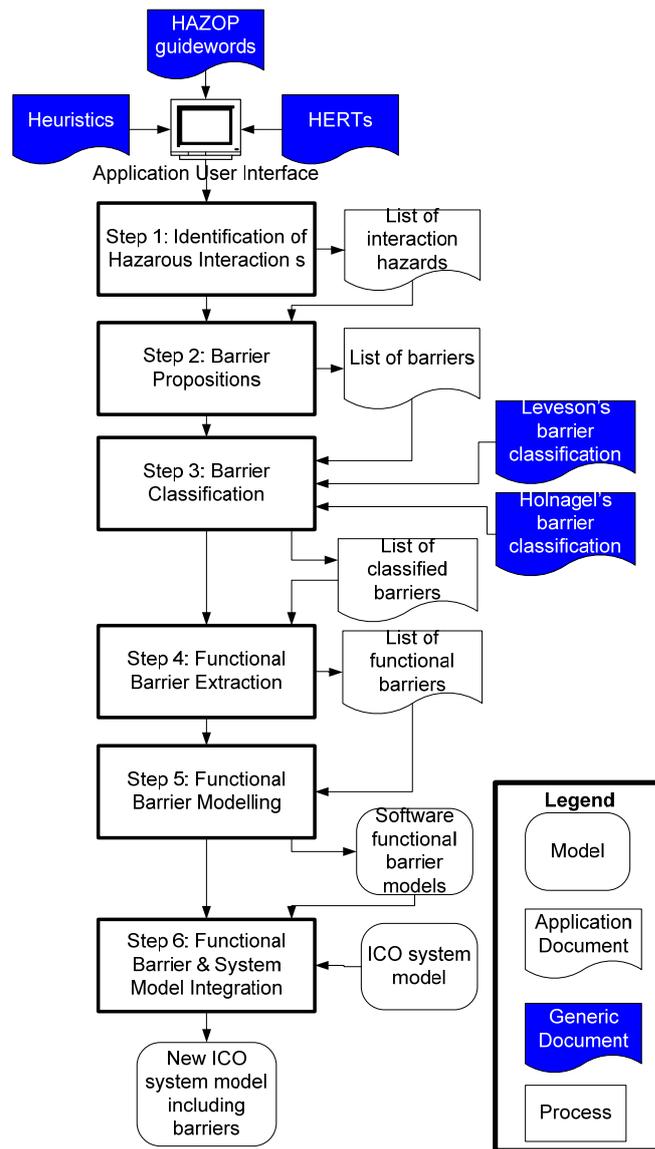


Figure 7-5. Software Barrier Identification, Analysis and Modelling Approach

The following section details the interaction hazard analyses and results followed by a section which presents the proposed barriers. Finally, we present the modelling of the barriers and their integration with the previously modelled ATM system model.

4.1.1 Interaction Hazard Analysis/Unintentional poor design

This section is dedicated to the description of the interaction hazard identification analyses including related work in the field of software safety analysis. In (Falla 1997), it is argued that current software safety standards provide little advice regarding threats which ought to be considered when undertaking software hazard analysis such as environmental and operating conditions, logic control, real time executive, system function calls, system resources, timing and software design notations. Although all these types of hazards are important, in our approach we focus our attention on hazards relating directly to UI interpretation problems and ways of improving the UI to assist the operator.

We present our approach on the User Interface part of the ATM example to remain consistent with the rest of our approach presentation chapters. The following figures illustrate some very basic UI screenshots to give the reader a general idea of which interactions we are referring to when applying the analyses.

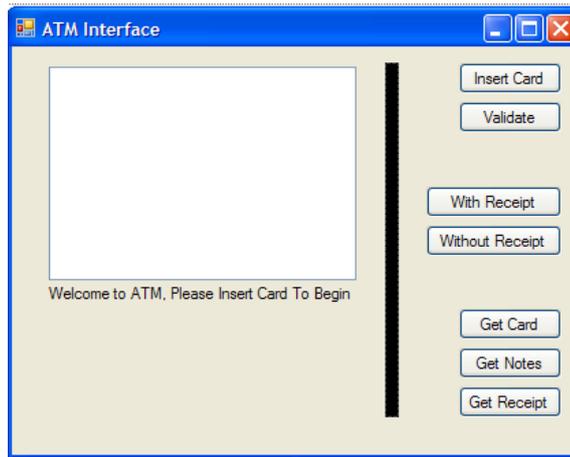


Figure 7-6. ATM Interface (Welcome Screen)

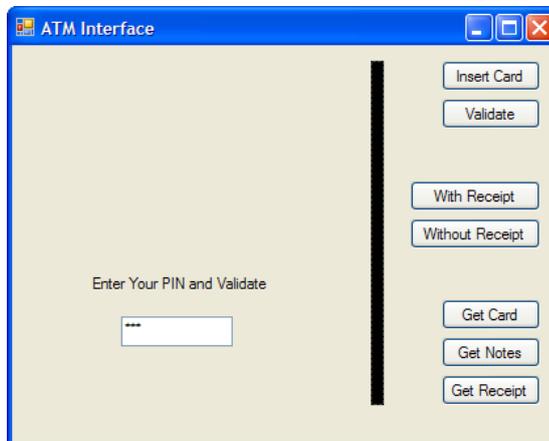


Figure 7-7. ATM Interface (Enter PIN Screen)

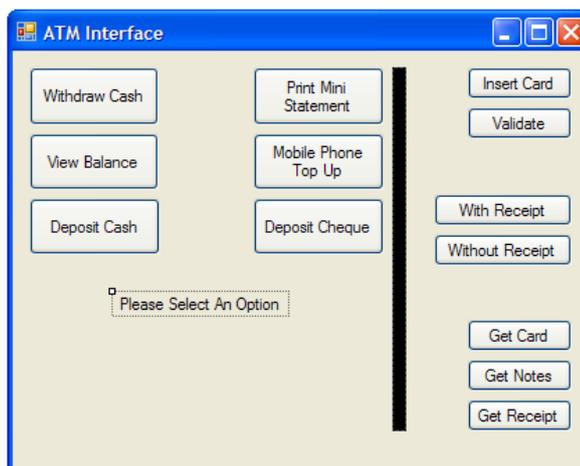


Figure 7-8. ATM Interface (Select Option Screen)

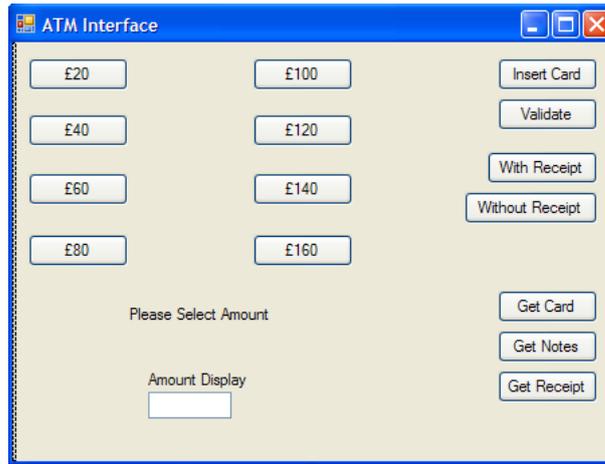


Figure 7-9. ATM Interface (Select Amount Screen)

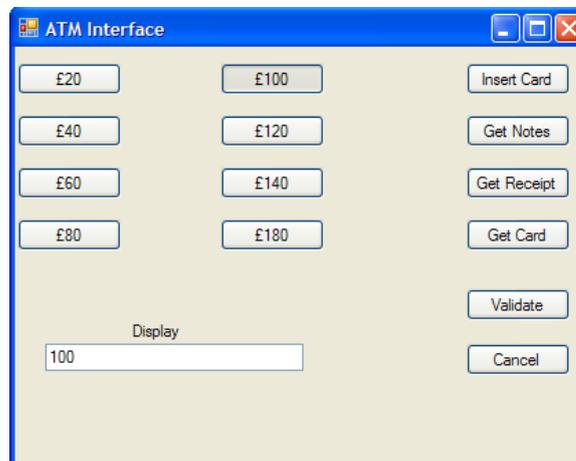


Figure 7-10. ATM Interface (Amount Selection Screen)

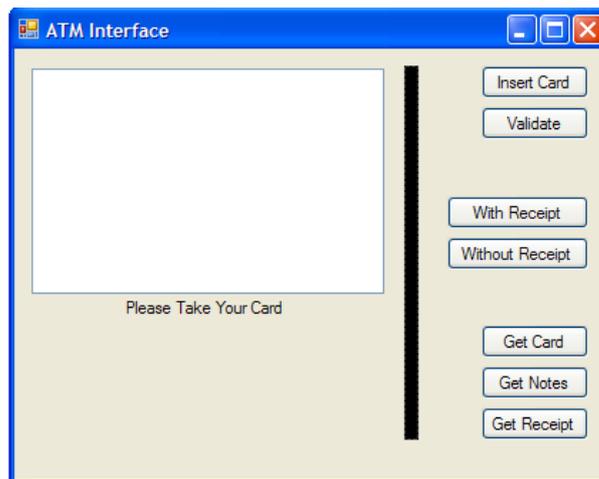


Figure 7-11. ATM Interface (Card Ejection Screen)

4.1.1.1 Skill-based HERT analysis

So far in this thesis, we have applied to HERTs to subtasks of a user task model. However, this kind of task-based analysis of deviation requires detailed information concerning user behaviour which in some circumstances can be difficult to obtain, for example, analysing the exact tasks of pilots of nuclear power plant operators. Here we propose a complementary approach using HERTs that is based on generic “error” types and does not require this kind of information.

We apply the HERTs directly to the User Interface. Using HERTs we are able, for each type of human error, to deduce possible erroneous interactions with the system and what the impact of the identified ‘error’ will be on the task in hand. The aim of these reference tables is not to guarantee a comprehensive and exhaustive identification of every possible eventuality but to provide investigators or safety analysts with systematic ways of exploring likely and reoccurring user deviations.

Table 7-3 documents an extract of the outcome of this process focussing on the skill-based error table only. The key benefit of this human ‘error’ analysis is that its style not only helps to identify alternate problems that might jeopardise the future safety of an application but which did not previously arise during the design, user testing or evaluation phases.

Table 7-3. Extract of the HERTs Analysis on ATM UI

Error Type	Example	ATM ICO System Model Impact
Strong-habit intrusion	Select “withdraw cash” instead of “print mini statement”	User taken to withdraw cash option
Strong-habit capture	Intend to validate PIN but don’t	No impact, system is idle waiting for PIN to be validated
Strong-habit exclusion	Select withdraw cash option, but then don’t withdraw cash	No impact, system is idle waiting user to select amount
Branching errors	Select withdraw cash, select amount, but then don’t answer receipt question	No impact, system is idle waiting user to select receipt option
Overshooting a stop rule	Select print a mini statement option knowing it was not the intended action	Not modelled
	Confirm PIN knowing you have typed the wrong PIN	Incorrect PIN, two more authorised attempts
Program Counter Failures	Select withdraw cash option, get distracted, then omit the amount selection	No impact, system is idle waiting user to select amount
Post-completion error	Perform withdraw cash procedure, but do not take card back from machine.	Card retains card after 5 seconds
...

We only show an extract of the results for a number of reasons. Firstly, an ATM interface is not an ideal application to perform such an analysis on. The interface of an ATM is fairly sequential, leaving little room for human error, thus the results are less interesting. Also, we have not fully modelled the ATM system model, which means some of the results have an unrealistic impact on the system model. We could have implemented timing constraints on the model so that the system impact would be more realistic, but this is not the aim of the chapter or of explaining the approach. Here we simply wish to show how the HERTs skill-based table can be applied to the ATM interface. The results provide examples from various screens of the UI, though a thorough analysis of the whole system should be performed.

4.1.1.2 Heuristic analysis

Heuristic Evaluation (HE) (Nielsen and Molich 1990) is an established Usability Inspection Method (UIM) in the field of HCI (Human-Computer Interaction). It has been argued that such a UIM is not the most effective for identifying potential usability problems (Cockton et al. 2003). We understand that for larger, more complex systems involving complex activities from users, it would be more appropriate to base the evaluation on a user goal and involve analysts to predict usability as we have done while identifying potential scenarios of interaction with a system (Basnyat et al. 2005b).

We present the complete analysis results here though they are not very interesting due to the nature of our non safety-critical and simple ATM application. An extract is presented in Table 7-4. The results are used, in addition to the HERTs analysis results, to identify potential software barriers. By this we do not mean barriers to accessibility as in (Killam and Holland 2003), instead we mean potential software barriers that could help prevent erroneous user interactions.

Table 7-4. Extract of Heuristic Evaluation Results on ATM UI

Nielsen's 10 Heuristics	ATM UI
Visibility of system status	Ok
Match between system and the real world	N/A Due to very basic nature of UI design
User control and freedom	Cancel feature available once PIN has been validated. Model also includes delete feature to delete PIN and re-enter
Consistency and standards	N/A Due to very basic nature of UI design
Error prevention	No confirmation of chosen options, no warning of 5 second card detainment rule
Recognition rather than recall	Only necessary to recall PIN, human "errors" can therefore occur here
Flexibility and efficiency of use	Very little flexibility once an option such as "withdraw cash" has been selected
Aesthetic and minimalist design	N/A Due to very basic nature of UI design
Help users recognize, diagnose, and recover from errors	None available
Help and documentation	None available

4.1.1.3 HAZOP Analysis

HAZOP is a systematic technique that attempts to consider events or process in a system exhaustively. Within a particular system domain (or scenario), items (or events) are identified and a list of guidewords is applied to the items. The guidewords prompt consideration of deviations to item behaviour (guideword examples include less, more, none, more than) to elicit the potential for arriving at possible hazardous states (Smith and Harrison 2002). These guidewords provide the structure of the analysis and can help to ensure complete coverage of the possible failure modes (Pumfrey 1999).

Although a HAZOP analysis should generally be applied to a Piping and Instrumentation Diagram or some form of graphical representation of the system, we apply the analysis to the UI. An extract of the results can be seen in Table 7-5.

Table 7-5. Extract of HAZOP analysis results on ATM UI

Guideword	Deviation	ATM UI Interpretation
MORE THAN or AS WELL AS	All intentions achieved, but with additional effects (qualitative increase)	N/A UI too simplistic
PART OF	Only some of the intention is achieved (qualitative decrease)	Select withdraw option, but do not select amount to withdraw
OTHER THAN	A result other than the intention is achieved	Select print mini statement option instead of withdraw cash option
REVERSE	The exact opposite of the intention is achieved	Say no to receipt when intending to say yes
Early, Late, Before & After		Take card too late, the machine will retain card after 5 seconds
...

4.1.2 Software Barrier Identification

In this section we discuss the relation between the identified interaction hazards and potential barriers that could be imposed to minimise the threats. We detail the categorisation of proposed barriers according to Hollnagel and Leveson's categorisations. The barriers identified following the heuristic analysis relate mainly to the interface design, layout and representation. The list below provides rudimentary barriers to resolve some of the issues identified.

Table 7-6. Barrier Suggestion and Classification

Hazardous Interaction & Barrier Suggestion	Analysis Technique Identifier	Leveson's classification	Hollnagel's classification	
To deal with selecting wrong function of the ATM	HERT			
Select print a mini statement option knowing it was not the intended action	HERT			
Very little flexibility once an option such as "withdraw cash" has been selected	Heuristic			
Select print mini statement option instead of withdraw cash option	HAZOP			
Say no to receipt when intending to say yes	HAZOP			
B1) "undo" feature to go back to previous menu				
Intend to validate PIN but don't	HERT	Lockin	Preventing	Logical soft
B2) Explicit on screen feedback/instructions requiring validation once 4 digits are sensed		N/A	Symbolic	Indicating, Countering, Regulating
Select withdraw cash option, but then don't withdraw cash	HERT			
Select withdraw cash, select amount, but then don't answer receipt question	HERT			
Select withdraw cash option, get distracted, then omit the amount selection	HERT			
No confirmation of chosen options, no warning of 5 second card detainment rule	Heuristic			
Select withdraw option, but do not select amount to withdraw	HAZOP			
B3) Feedback and warning to retain card. Retain card if no activity for x seconds.		N/A	Symbolic	Indicating, Countering, Regulating
		Lockout	Functional	Mechanical Hard & Logical Soft
Confirm PIN knowing you have typed the wrong PIN	HERT			
B4) Allow minimum number of attempts though provide clear warning (partly implemented)		N/A	Symbolic	Indicating, Countering, Regulating
		Lockout	Functional	Logical Soft
Perform withdraw cash procedure, but do not take card back from machine.	HERT			
B5) Ensure card is taken before cash is ejected (already implemented in our ATM system model)		Lockin	Functional	Mechanical Hard & Logical Soft
Only necessary to recall PIN, human "errors" can therefore occur here	Heuristic			
No barrier suggestion for human memory				
Take card too late	HAZOP	N/A	N/A	N/A
B6) Retain card after 5 seconds (already implemented in our ATM system model)		Lockout	Functional	Mechanical Hard & Logical Soft

In total, 6 barriers (after generalising the findings) were identified using the three complementary techniques.

The barriers are in italic font in Table 7-6. They can be considered a combination of protective and preventative barriers. These barriers are classified (according to Hollnagel’s classification of system barriers and Leveson’s classification of software barriers).

Since our example, the ATM cash machine involves physical hardware interaction, some barriers reflect the behaviour of the physical system, such as card retaining. However, the same barrier has an impact on the system model representing the behaviour of the system. Thus such a barrier should also be present in the system model which will trigger physical events on the machine.

Of the 6 barriers identified in this analysis, barriers B1, B3, B4, B5 and B6 can be modelled and integrated with the system model. Barriers B4, B5 and B6 have been implemented (B4 partly because we currently provide no feedback).

The barriers are suggestions from the designer’s or expert’s point of view. Team work is essential for this type of activity to obtain views of several people. Techniques such as QOC can be useful for tracing design decisions and validating design choices.

The next logical phase of our approach then, after the extraction of barriers that are directly related to the system model, is to model these barriers for integration with the system model. This has been illustrated in Chapter 5, when we changed the behaviour of the ATM system model such that, if there was a fire and the user had no card, then the card swiping system would be overridden by the door automatically opening.

Here we provide a final example of barrier modelling for system integration. We take barrier B1 as an example and apply it to the identified hazardous interaction “Say no to receipt when intending to say yes” (we do not apply it to all identified cases).

4.1.3 Software Barrier Modelling

This section presents how the extracted barriers can be modelled using the ICO notation. We have chosen to illustrate barrier B1 as it is both representative and still has a limited impact on the models. The barrier involves a loop back to the previous state. This allows the user to effectively “undo” an action, and when satisfied, confirm and continue.

Software barrier modelling and indeed socio-technical barrier modelling as we have seen in the first half of this chapter can take one of three forms. These are represented in the following three figures.

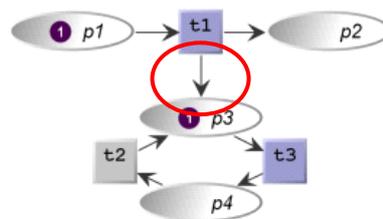


Figure 7-12. New arcs between existing components of the system model

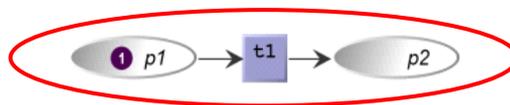


Figure 7-13. New components of the system model

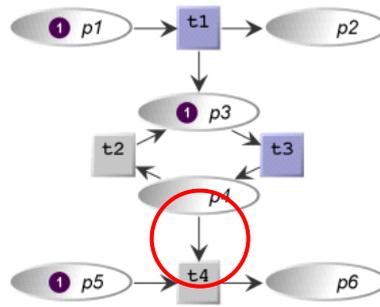


Figure 7-14. New arcs between existing components of the system model the new components

Barrier B1, the undo feature takes the form of Figure 7-14. It involves new components and arcs connecting those new components to the existing components. Figure 7-15 illustrates an extract of the ATM system model. We present here the receipt feature of the model before barrier (B1) implementation. Figure 7-16 illustrates the same extract, this time including barrier B1.

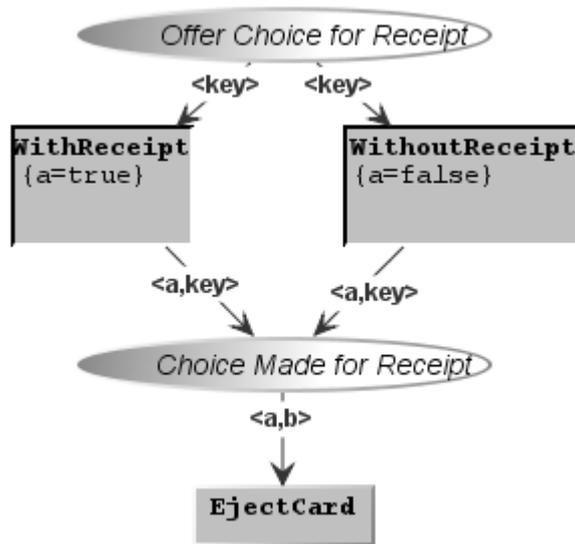


Figure 7-15. Receipt Feature of System Model Before Barrier Integration

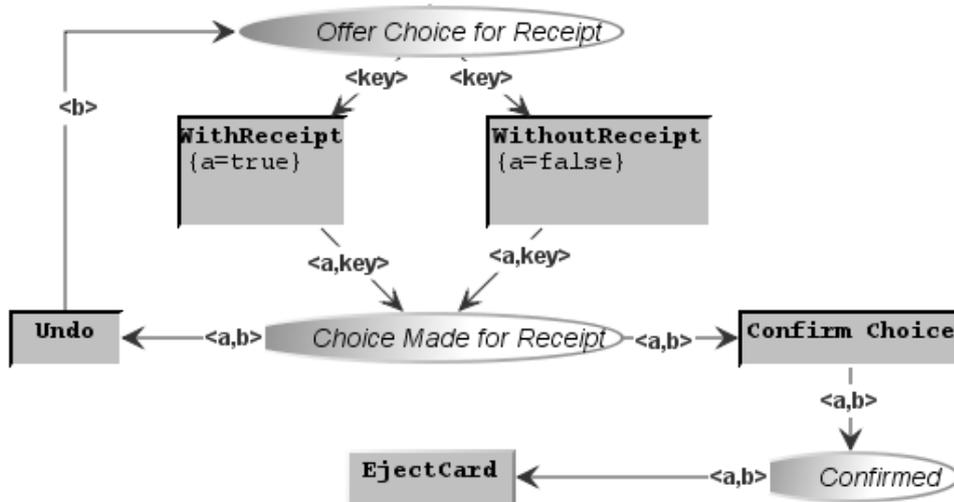


Figure 7-16. Receipt Feature of System Model With Barrier B1

The barrier feature works as follows. When there is a token present in place *Offer Choice for Receipt*, synchronous transitions *WithReceipt* and *WithoutReceipt* are fireable. The firing of either of these transitions sets a token in place *Choice Made for Receipt*. Here, both *Undo* and *Confirm Choice* are fireable. The *Undo* transition takes the system back to the state when a token is set in *Offer choice for Receipt*, while the *Confirm Choice* transition sets a token in the confirmed place allowing transition *Eject Card* to fire.

The type of undo/redo feature advocates usability, providing the user with more options and flexibility, though it makes the system model more complex in turn making it more error prone and less reliable.

4.2 Barrier and System Model Integration

It is difficult to show the barrier without showing and exploiting the system model, as the barrier often impacts the system model. This can be seen in the previous section where we illustrate the modelling of barrier B1 and are forced to show part of the existing system model. Thus the idea of barrier modelling and barrier model integration as two separate activities is ideal but difficult in reality. The barrier could impact only a subset of the system model or the majority of the behaviour. As we have seen in the previous section, the modification can simply correspond to adding arcs to existing components. Figure 7-17 illustrates the concept of barrier model and system model integration.

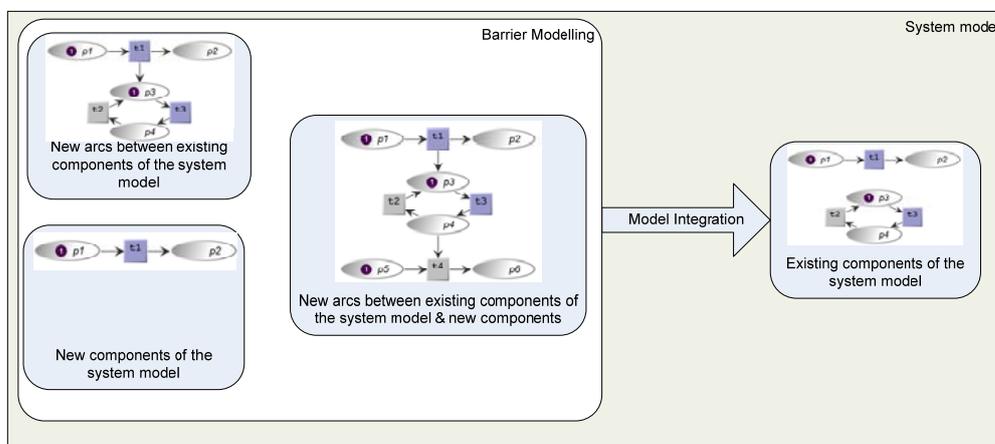


Figure 7-17. Barrier Model and System Model Integration Concept

5 Supporting Integration

In Chapter 5 we discussed the integration and presentation of the system modelling phase of the framework with the rest of the phases but did not discuss the barrier analysis and modelling parts. Here we make explicit where the barrier parts have been presented and discuss their integration with other processes and phases. It is unlikely that a system design will begin with barrier analysis as it depends firmly on system modelling, and potentially incident and accident investigation data. The barrier analysis is used to support the system modelling, to design the system with additional safety-related data in mind. The interaction hazard analysis may (depending on available knowledge about users and the system) receive data from the task model, in order to perform an analysis of the application based on end user tasks and goals. Furthermore, the barrier analysis may receive information from the safety modelling phase, from safety cases or incident and accident reports which may contain implicit or explicit references to successful or failed barriers. Our research on barrier analysis and modelling has been published in the following (Basnyat et al. 2005c), (Schupp et al. 2006), (Basnyat and Palanque 2006) and (Basnyat et al. Submitted).

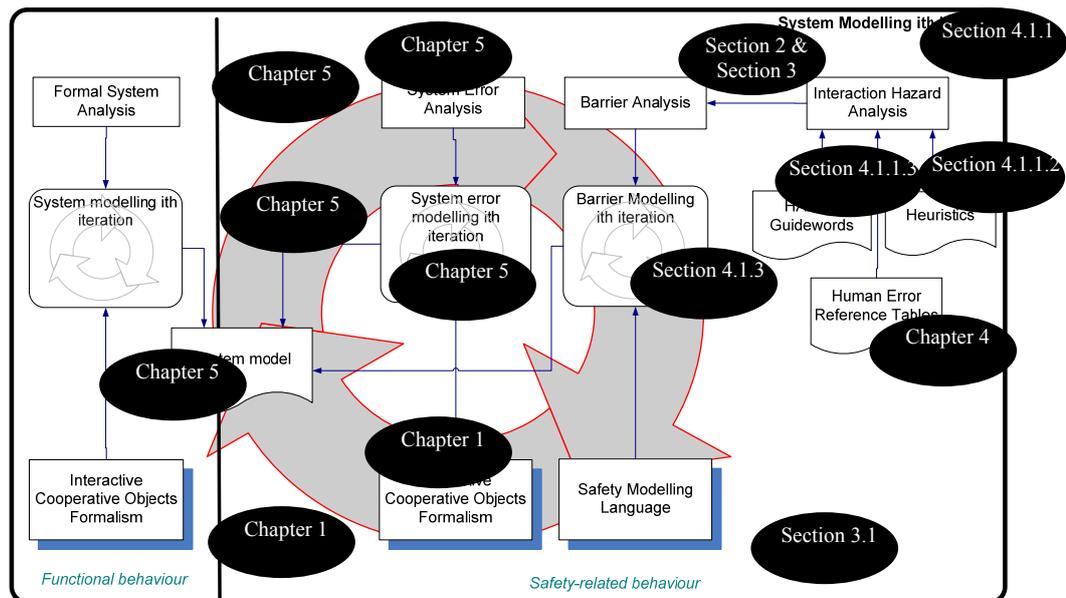


Figure 7-18. Barrier Analysis and Modelling With Respect to the Generic Integrated Modelling Framework

6 Conclusion

This chapter has presented a systematic approach for identifying socio-technical and software barriers after an incident or accident and before an incident or accident. The post-incident technique involves the use of safety cases and incident reports to identify barriers and failed barriers. These identified barrier may be of different types, such as technical, human or socio-technical barriers, some of which may directly impact the system model. It is these barriers we are particularly interested in due to our interest in system modelling. For this, we have presented a three-step approach for identifying, modelling and formally specifying safety critical human tasks, interactive system and their associated barriers. The Safety Modelling Language is used to model existing as well as identify new socio-technical barriers. We then used the Interactive Cooperative Objects (ICO) formalism to specify the behaviour of the barriers. Finally we integrate these barriers with a system model, also specified using the ICO formalism.

We have shown that it is possible to integrate individual and multiple barriers with the system model. The advantages of using the ICO formalism within the Petshop environment means we can simulate our informed models to determine the impact that the barrier will have on the system. However, it is important to note that integration of multiple barriers raise additional issues related first to the increase of the size of the models, second to the potential conflicts between the various barriers. However, we believe that formal description techniques can provide a unique way of detecting such conflicts even though their resolution remains an open (design) issue.

In addition to barriers identified post-incident, we have provided an approach for identifying software related barriers pre-incident. This is achieved by means of user interaction hazard identification, using three complementary techniques, HERTs, HAZOP and Heuristic analyses. Again, we have used the ICO formal description technique to model the extracted functional barriers and integrated them with an existing model of the ATM cash machine model. The modelling of such barriers allows us to simulate potential user interactions with the system to ensure that the barrier eliminates the previously identified hazards.

The approach also aids in identifying further barriers such as symbolic and immaterial barriers that should be represented and taken into account within the development process, however not within the system model.

We believe that by identifying and incorporating socio-technical and software barriers such as those discussed in this chapter within their relevant models, we can obtain a verification of some potential problems before the application is actually implemented. This will ultimately lead the design of safer safety critical interactive systems by embedding reliability, efficiency and error-tolerance in the system.

Part 2 Conclusion

This section concludes part two of the thesis. In part two we presented the various phases of the generic integrated modelling framework. Each phase, including the proposed techniques, tools and methods has been applied to an ATM cash machine example. In addition to demonstrating the applicability of the framework, illustrations and discussions on how these phases and approaches are integrated have been provided.

The framework tackles usability, reliability and safety by applying user centred design, formal specification, human 'error' and incident and accident investigation techniques respectively.

The framework provides means (notations, tools, techniques, methods etc) for the systematic inclusion of human factors issues, safety issues and reliability issues throughout the model-based development process of safety critical interactive systems. The framework could also be applied to non-safety critical systems by excluding unrelated phases such as the safety modeling phase. The fact that we use models and notations to support our approach introduces additional costs to the development process.

The framework, its proposed techniques and approaches can be considered costly, in terms of time, knowledge and expertise required to perform them. In this conclusion, we discuss arguments supporting our approach in terms of cost/benefit issues.

It must be noted that what we suggest and present in this conclusion has not been proven but is expected and anticipated from our approach. The representations of the bars in the graphs provided below do not refer to specific statistics. It is also worth stating that, it is unrealistic to argue that in a user-centred, safety-centred or reliability-centred development process will the system be completely usable, safe or reliable. One cannot guarantee that the system will be perfectly safe for example which is why industrial standards adhere to words such as "acceptable safe". It would be useful and we should have developed an experiment to validate or invalidate such claims but we believe this is beyond the work of this PhD. Indeed, this would require multiple case studies and a lot of resources unfortunately not compatible with a PhD project. However, cost/benefit aspects are important and should be mentioned in the thesis.

The same arguments surrounded the introduction of Object Oriented Programming (OOP) languages with respect to structured programming languages. It was impossible to suggest that OOPs would be more beneficial, and still cannot be completely proven, but at least two identical systems, with teams of designers, of the same expertise, knowledge and tools etc would need to be designed in order to validate claims that one method is better than the other. Such a validation would take a lot of time and would still not give a certain result. Thus for us also, and our proposed framework, we cannot prove that it would be better than a similar approach since we cannot apply the two techniques in identical situations.

The benefits we describe are greater for the kinds of systems we are interested in, which is safety-critical interactive systems. Though again a thorough investigation is needed to justify such claims and determine the characteristics of such systems.

6.1 Dealing with the cost/benefit issue

One way of justifying the profitability of our generic integrated modelling framework is to distinguish between the three areas we tackle, UCD, reliability and safety in order to weigh out its profitability in terms of cost/benefits. Our evaluation of the cost/benefits of each development process highlights that we believe that the latter three will be cheaper, in terms of costs incurred during usage. This is explained in the following subsections. We propose various ways based on tools (for editing, verifying and reusing models) and methodological approaches (such as patterns and reuse of knowledge) to reduce the costs of system usage. In effect, the development costs increase which are represented as the black parts of the development bars presented in the figures in this conclusion.

6.1.1 Classical development processes

Figure 7.19 illustrates the cost of development of classical development processes, such as the waterfall approach (Royce 1970), the V-Cycle (Mc Dermid and Ripkin 1984), the Spiral model (1988) and the cost of User Centred

Design approaches. The grey parts of the bars (Figure 7.19), which are applicable to all processes discussed in this section, represent the development costs of projects. The classical development process has been divided into three parts, for the standard development costs, the white part for the cost of system use, and the patterned part for additional costs due to poor usability of the system. This patterned part is necessary to represent since end users are not directly considered in classical development process, the costs of system use once the product is ready will be high due to design issues that will surface relating to usability, error tolerance, designer model/user conceptual model mismatches etc. If there are too many problems for the end user, they may stop using the product and cause profits to drop. Alternatively, the designers may take these problems into account and reiterate the design process in order to satisfy user demands. This iterative development and redesign of the system is costly in terms of time, resources and patience of the users/customers.

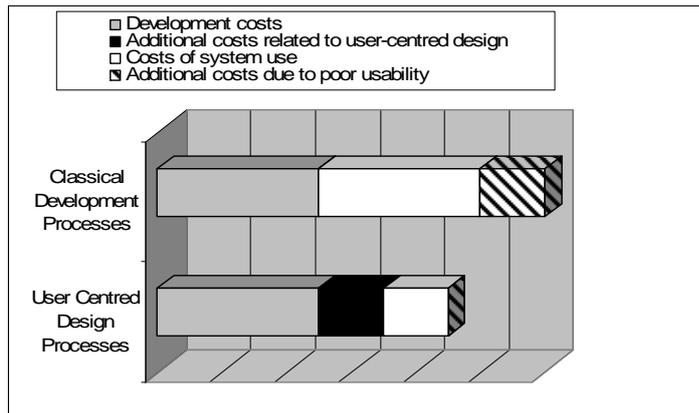


Figure 7-19. Comparing the Costs of Classical Development Processes with UCD Processes

6.1.2 Usability

As we have seen in this thesis, user centred development advocates the use of task analysis, surveys and questionnaires, field studies, focus groups etc in the early phases of development. Such approaches are counter typical software engineering techniques that promote iterative development processes in order to produce software as quickly and as cost-effectively as possible.

When considering the cost/benefits of development processes promoted in the field of HCI, not only should development costs be taken into account, but also the additional costs necessary to perform UCD approaches. The additional expense required is for user testing, task modelling, heuristic evaluations etc to design a more user centred, usable system, and is represented in the black part of the UCD bar. The black part of the UCD bar shows the additional costs directly attributed to user centred design.

UCD processes compensate additional costs by offering additional gains when the system is actually deployed and used. The precise evaluation of costs and payoffs for usability engineering can be found in (Mayhew and Bias 1994). By incorporating such techniques, the cost of system use will be minimised (white part of UCD bar) for example less user training will be required, because the users will have already been vastly considered, and potentially less usage errors would occur. The gains obtained when conducting UCD in terms of end user performance (for instance, by providing default values in the various fields of a computer form) or in reducing the impact of errors (by providing undo facilities, for instance), can only be evaluated if the actual use of the system is integrated in the computation of the development costs. For this reason, in addition to the grey bar of the UCD process in Figure 7.19, a white bar referring to costs of the user of the system is displayed. This white bar is also present in the “classical development processes” bar though for UCD it is smaller, because the system will be more adapted to the end users, more of the user requirements and user behaviour will have been taken into account. This is done by reducing user errors, frustration, increasing user performance by better organisation of tasks on the system and providing better interaction techniques for example.

Thus in comparison to the classical development processes, we can argue that UCD is more profitable in the usage part of the system, in terms of cost/benefits.

6.1.3 Safety

Another aspect we treat in our generic integrated modelling framework is safety by incorporating safety related data by means of safety cases and incident and accident reports as well as by applying investigation techniques to reports.

Probably the most important problem with research in accident investigation, and overall safety research, is that it is very difficult to validate the conclusions and suggestions made. This becomes more evident when considering that safety case practitioners use the phrase “acceptably safe”, rather than claiming a system is completely safe. We have suggested a way of dealing with diverse issues, ranging from human activity to complex command and control systems, and most importantly linking them as elements of a safety argument, and reasoning about how they interact towards causing an accident.

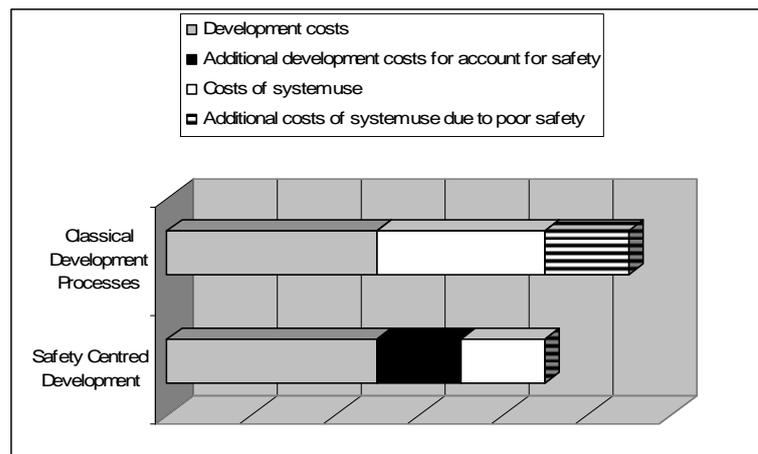


Figure 7-20 Comparing the Costs of Classical Development Processes with Safety Centred Development

This approach to developing in a safety centred way will include standard development costs but in addition to these will incur costs relating to the time and expertise required to perform the incident or accident investigation, study safety cases etc. Investigations of this kind typically involve lengthy documents, with data represented in various formats. Experts are required to extract data relevant for the system development and relevant in terms of system and task modelling (the two forms of modelling focused on in this thesis). This is represented in the black part of the safety development bar in Figure 7-20. The costs relating to system use are the white parts of the bars. It can be seen that the costs of system use in the safety-centred development bar are smaller than in the classical development process bar because we anticipate that the system will be safer and less prone to incidents or accidents.

The patterned part of the classical development process bar, additional costs of system use due to poor safety, refer to loss of life, loss of materials, destruction of environment, cost of insurance payouts etc. In the aftermath of an accident there may be limited finance. One might argue that using a minimal budget on accident analysis and systems design may not be the most appropriate way to use the available funds. However, we argue that this is precisely where investment should be made, in order to help prevent other similar accidents from occurring by learning from previous incidents or accidents.

6.1.4 Reliability

To achieve a more reliable system, formal specification techniques are required in order to ensure that the system under design is not underspecified, that it accounts for all system eventualities (for instance, by explicitly representing all possible system states and events) and that the system will run as expected. In safety critical systems design, the aim to improve reliability is necessary because the cost of failure is important.

While designing interactive software, the use of a formal specification technique is of great help because it provides non-ambiguous, complete and concise notations. The advantages of using such formalisms are widened if they are supported by formal analysis techniques that allow proving properties about the design, thus giving an early

verification to the designer before the application is actually implemented. In addition to these ‘standard’ benefits of FSTs, our framework introduces additional means for improving the reliability of the system by accounting for erroneous human and technical-related behaviour as well as incorporating technical, human and socio-technical barriers into the formal specification of the system model and semi formal specification of the task model.

An example of a serious accident, partly attributed to development faults, local faults and availability/reliability (Laprie 2005) is the explosion of the Ariane 5. On June 4th, 1996, the maiden flight of the Ariane 5 launcher exploded about 37 seconds after lift-off. Scientists with experiments on board that had taken years to prepare were devastated. For many software engineering researchers, however, the disaster is a case study rich in lessons. The disaster was caused by a software fault, due to an unchecked overflow of a 16-bit counter.

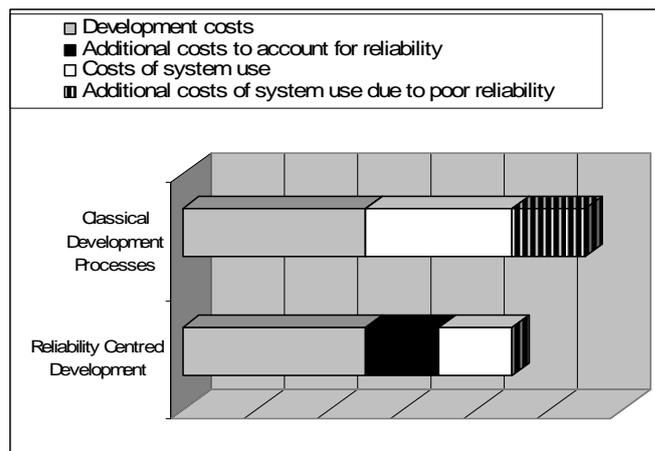


Figure 7-21. Comparing the Costs of Classical Development Processes with Reliable Centred Development

These benefits of FSTs enhance system reliability though incur costs in terms of additional development requirements. FSTs require experienced and knowledgeable designers capable of using notations. Furthermore, dedicated tool support is necessary for handling large models which may need to be developed to meet specific requirements of notations employed. This additional cost is also represented the black part of the reliability development bar which adds to the standard development costs.

The patterned part of the classical development processes bar in Figure 7-21 refers to the additional costs of system use due to poor reliability such as failures, system crashes, necessary rebooting, unavailability of services etc.

Since the development of a more reliable system will account for additional aspects than those in classical development techniques, we argue that the cost of developing a more reliable system is lower in the usage part of the system.

The three aspects presented as separate development processes do not provide gains in terms of development time or costs, though by dedicating more time to the development process will minimise the costs of system use once deployed. The following sections provide methodological and tool-based techniques for helping reduce the additional costs (the black parts) of the development process.

6.1.5 Methodological based cost-reducing techniques

One particular way that we tackle usability, reliability and safety is by methodological based cost reducing techniques. These include for example, model patterns for tasks, human error analysis techniques, barrier analysis and modelling techniques, incident and accident scenario identification, which advocate reuse and systematic application of established theories.

Of course, explicit error handling, such as the model patterns for tasks which account for known human related erroneous behaviour puts more burden on the designer, and increases the complexity and the cost of the specification. To summarise costs are:

- adding information to the specification
- diminution of the readability of the specification
- increase of costs if the specification has to be modified.

Benefits of the explicit modelling of errors are :

- increase of user's confidence in the system
- reduction of the number of system's failures
- increase of user performance in reaching the goals.

6.1.6 Tool based cost-reducing techniques

The generic integrated modelling framework suggests dedicated notations, such as ICOs, CTT, SML for the modelling of specific information. However, formal specification techniques, such as ICOs can be complex and designers may be reluctant to use them. For these reasons, several notations used within the framework are supported by dedicated tools. This is especially necessary for large systems development. The tools can help not only with the editing of models but also the verification of models.

For the modelling of systems, the ICO notation is fully supported by Petshop, which allows modelling editing, simulation and verification of properties such as T-invariants and P-invariants. The tool is fully capable of modelling large real life applications as has been demonstrated in (Barboni et al. 2006b), (Barboni et al. 2006a) in which a cockpit application has been fully modelled. This case study is further presented in Chapter 9. Further more, the tools JARP and ARP presented in previous chapters are able to support the automatic generation of marking trees and marking graph as well as provide additional property verification information such as model liveness, boundness etc.

On the task modelling side, the tool CTTe supports the editing and simulation of single and cooperative task models. Though it currently does explicitly support model patterns for tasks, we have used the tool to perform such an activity. Furthermore, in previous work,(Navarre et al. 2001a) provided a tool suite for the integration of task models and system models through scenarios by developing a “correspondence editor” which associates interactive tasks (on the task side) and user services (on the system side).

6.2 Summary

We have argued that our generic integrated modelling framework, is profitable in terms of costs/benefits in comparison to classical development processes because of the amount and type of data taken into account during the development process and the advantages these methods bring to the end product.

Of course, the cost/benefits arguments presented so far depend on the type of system under development. Performing safety modelling and barrier analysis on a system that is not safety-critical would be a waste of time and money. In this case, our framework is still applicable to the design of non safety-critical system by simply excluding such non-appropriate phases.

We argue that we wish various experts to continue using techniques they are experienced in and that are relevant to their background and domain of expertise. We do not encourage one person to learn all of the proposed techniques and methods. Instead, we provide a means for a multidisciplinary team, to represent important data necessary for building a safer, more reliable and more usable system in such a way that the individual teams can inform one another's models to provide completeness with respect to what has been taken into account. Initially however, training would be required in order for the various domain specialists to be able to learn the notations and tools and this would take a significant amount of time. It would not be expected that the person modelling the system would perform a task analysis. Furthermore, perhaps even the person modelling the tasks may not be able to apply the HERT analysis. Thus the task model could be handed over to a Human Factors expert, who could apply the HERT analysis to each subtask, and give the results back to the Task Analyst for model pattern integration. This error explicit task model can be simulated using the tool support and applied to the system model by the systems engineers.

An additional point to discuss, is that it is not reasonable to separate the diagrams into three separate development processes but we have done so for the purposes of explanation. The three aspects, usability, reliability and safety

Part 2 Conclusion

should be and have been tackled within the same process as part of an integrated framework. They are therefore likely to have synergistic advantages of all systems, for instance, when performing system modelling, we have shown that it is possible to generate state spaces for input to incident and accident investigation. This means, for the incident and accident investigation phase, and other phases such as usability testing and incident scenario testing on the system model, the techniques does not have to be carried out twice

Another interesting discussion lies on insurance. Some financial policies in companies may prefer to take the risk in spending less on development and potentially more on insurance payouts rather than investing on the design and increasing safety of the system.

PART 3 – CASE STUDIES

Chapter 8

Exploding Vessels Under Pressure Accident

“We can't solve problems by using the same kind of thinking we used when we created them.”

(Albert Einstein, 1879 – 1955)

Chapter Summary

This chapter presents the first of two case studies. Here we focus on a fatal mining accident. We provide an overview of the mining plant, the events and circumstances surrounding the accident and then apply the framework, presented in the previous chapters to the case study.

The aim of this chapter is to show that our generic integrated modelling framework can be applied to real life case studies.

1 Introduction

Control systems, which are devices or a set of devices to manage, command, direct or regulate the behaviour of other devices or systems, range from fuel line controllers to automated flight controllers. Such devices are increasingly being incorporated within complex software systems. Errors in such systems can have dramatic consequences, hence the urgent need to be able to ensure and guarantee their correctness. Formal specification of such safety-critical interactive systems aids us in representing their behaviour without ambiguities, in a clear and concise way, providing means to reason about system properties. In addition to system behaviour, human-computer interaction with systems involving command and control systems is also critical as we have seen in this thesis. There is an issue of over reliance on the command and control systems, this is well researched in the field of aviation, for example when pilots monitor the system's behaviour but may at times need to intervene if there is a problem.

We chose to focus on this specific accident because it is a strong example of how complex industrial accidents can be; (as we shall see later on) it was primarily caused because of a failure in a sophisticated control system which can be traced back to organisational, systemic and human factors.

The increasing complexity of many computer-controlled application processes is placing increasing demands on the investigation of adverse events. At the same time, there is a growing realisation that accident investigators must consider a wider range of contributory and contextual factors that help to shape human behaviour in the causes of safety-related incidents.

Command and control systems pose considerable challenges for accident investigators. From a systems perspective, these applications are becoming increasingly complex. For example, they now often involve programmable logic controllers (PLC) and other complex computational devices. From an ergonomic perspective, system complexity is exacerbated by the need to integrate the activities of multiple teams of workers who must cooperate and coordinate their activities across a plant. There is also a growing realisation of the need to consider a broad range of contextual and contributory factors in order to understand human behaviour both in the causes and the mitigation of adverse events.

These adverse effects of accidents are complex to analyse because of the combination of contributory factors (human and technical) that together caused the system (humans and machines) to fail. Trying to discover what went wrong and why, can be extremely multifaceted especially when dealing with complex, computer-based systems. We argue that involving opinions grounded on different backgrounds, as advocated in our generic integrated modelling framework, we can shed new light on the root causes of an accident. Lekberg's study of incident investigators within

the Swedish nuclear power industry has shown that analysts tend to identify root causes that are closely related to their own background and training (Lekberg 1997). Johnson has also shown that by recruiting investigators from different backgrounds, it is possible to identify recommendations that might otherwise have been overlooked (Johnson 2003)p215. By expanding the range of experience recruited to an investigation, it seems more likely that we will learn additional lessons from previous failures.

Industrial accidents often tend to be a combination of human activity with technology. In-depth investigation concludes in many cases that instances of ‘human error’ or technical failure are often caused by flaws in the initial system design (1995). Also, research has indicated that many accidents are socio-technical (Pidgeon 1997), and often attributed to political or even regulatory complications that designers could hardly have foreseen before an accident. Accident analysis techniques as for instance Events and Causal Factors Analysis (Buys and Clark 2001), or STAMP (Leveson N.G et al. 1991), are capable of identifying causation and relationship between causes of accidents to a great extent. However, the way with which different issues across high and low-level elements of an organization intertwine through time towards the occurrence of human error for example, can be too diverse and complex for only one technique to explain to a satisfactory level.

The problem surrounding system complexity is that the interaction of human agents and complicated technological systems is such that it is almost impossible to pre-consider all possible system states, for the sake of testing (Greenwell et al. 2004). However, once an accident has occurred, we are able to discuss how complexity has actually led to system failure. The approach undertaken in this paper is not simple, and it requires the combination of different expertise. From an accident report of 4 pages, we have produced over 30 pages of analysis. However, rather than finding out ‘what’ caused the accident, we were interested in ‘why’ all contributing factors emerged, and how they affected each other. For instance we have considered how the carrying out of a routine task can interfere with the performance of a technological component. How does the lack of training of personnel eventually cause an explosion? What does this mean to designers? These are questions that cannot be addressed with a simple and straightforward technique, but require in-depth multidisciplinary research.

2 The Case Study and the Generic Integrated Modelling Framework

In this section we highlight which parts of the generic integrated modelling framework have not been applied to the case study in this chapter. As mentioned in previous chapters, we do not tackle requirements modelling as this is beyond the scope of the PhD. We also do not perform a qualitative analysis as this case study is mainly targeted at presenting safety related aspects of the framework and that qualitative analysis mainly deals with reliability. In addition, this qualitative analysis is not specific to our approach and is largely dealt with in the field of formal methods. The interaction hazard analysis part of the system modelling phase is also exempt from this case study since we are primarily dealing with interaction between operators and a large hardware based system, though such analysis could have been performed on the software part of the command and control application in charge of supervising fuel flow variables. Furthermore, we do not apply any quantitative evaluation or usability testing on this case study since we do not have access to a working prototype on which such analyses should have been applied to. Figure 8-1 reproduces the framework and makes explicit using crosses the steps that are not addressed in this case study.

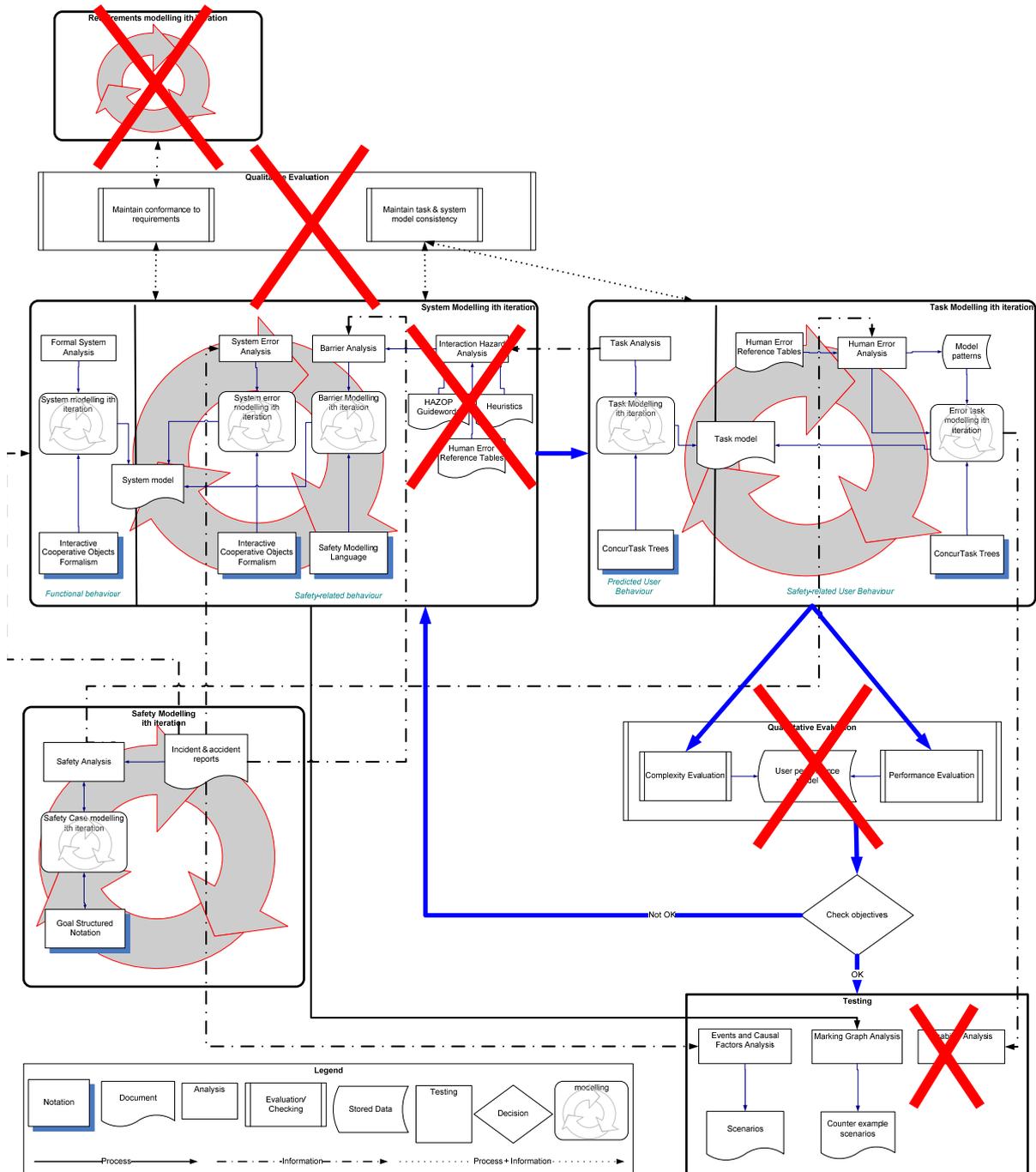


Figure 8-1. Phases of the Generic Integrated Modelling Framework not illustrated on this case study

3 Introduction to the Case Study

The case study we have chosen to illustrate the proposed multidisciplinary approach is a fatal US mining accident (Andrews et al. 2002; Andrews et al. 2002). The report is available at: <http://www.msha.gov/FATALS/2002/FTL02M34.HTM>. We will first introduce the plant to the reader and then describe the accident. A Quarry and Plant system is designed to produce cement. However, the part we are interested

in is the delivery of waste fuel used to heat the plant kilns. The Waste Fuel Delivery System is comprised of two separate liquid fuel delivery systems, the north and the south. Each system delivers fuel to the three plant kilns independently and cannot operate at the same time. See Figure 8-5 for a layout diagram. This particular application was chosen because it relies upon the operation of a complex, embedded command and control system.



Figure 8-2. Surface quarry and cement plant location (Photo from the National Institute of Standards and Technology NIST <http://www.cstl.nist.gov/acd/ashgrove.htm>)

Each delivery system contains the following components (presented here in the order of fuel flow):

- A fuel storage tank (right of the diagram)
- Two different sets of pumps, known as G and S (see Figure 8-3b and c), including motors and valves.
- A grinder, including motor (see Figure 8-3a)
- One $\frac{3}{4}$ " ball valve



Figure 8-3. Illustration of a) Grinder, b) Pump-G and c) Pump-S

In order to understand the events leading to the accident, we must first describe the interaction between the command and control system and the north Waste Fuel Delivery System. If the north fuel storage tank is open, fuel flows from the storage tank to north pump-S. The north pump-S motor pumps the fuel to the north grinder. The fuel is grinded and passes to north pump-G. The north pump-G motor then pumps the fuel into the three kilns. The waste fuel delivery systems also contain sensors located in different areas. Each delivery system has at least one temperature sensor and one fuel line pressure sensor. There is also a pressure sensor in the plant kiln area where the north and south fuel lines join (H symbol in the Figure 8-5).



Figure 8-4. Location of the accident within the plant (from the MSHA <http://www.msha.gov/FATALS/2002/FTL02M34.HTM>)

The sensors detect a number of abnormal conditions that will generate warning alarms in the control room. The command and control system will also automatically intervene to increase or decrease the speed of the pump motors when pressure is too low or too high. The fuel line pressure sensors also send information directly to the pump-S and G motors to maintain the correct pressure of fuel to the three plant kilns via the automatic step increase program.

The waste fuel delivery system has two independent but interconnected electronic systems for monitoring and controlling both the north and the south fuel delivery systems. The 'F' system receives signals from sensors located on fuel lines. The data is transmitted to a programmable logic controller (PLC), which raises audible and visible alarms in the control room and can also update more detailed process information on the monitoring screen.

3.1 Personnel

A number of different individuals contributed to the events that led to this accident. A Kiln Control Operator was located in the control room. Her responsibility is to monitor the fuel system via the screens and respond to any alarms. If necessary, an emergency shutdown of all pumps can be activated from the control room. The Kiln Control Operator uses a two-way radio to interact with other personnel located in the basement of the waste fuel containment area. The victim of the accident was responsible for monitoring the Waste Fuel Delivery Systems and for responding to instructions given by the supervisor who was located in the containment area. This supervisor monitors the Waste Fuel Delivery Systems and the worker. The supervisor communicates with the Kiln Control Operator via two-way radio. He receives information about the fuel flow from the Kiln Control Operator. The supervisor instructs the worker to make adjustments to the systems and switch the active Waste Fuel Delivery System from north to south or vice versa. The supervisor also assists the worker, if needed.

3.2 Engineering and the Case Study

Systems engineering is an interdisciplinary process referring to the definition, analysis and modeling of complex interactions among many components that comprise a natural system (such as an ecosystem and human settlement) or artificial system (such as a spacecraft or intelligent robot), and the design and implementation of the system with proper and effective use of available resources. In the mining case study, mechanical and automation engineers were involved. However, other types of engineers include hardware, software and systems engineers. The combination of these engineers assists in the system development process.

A combination of the work performed by the above mentioned engineers can be considered as partial cause for the fatal accident in the case study. One of the events leading to the accident was the failure of the PLC to automatically de-energise the fuel in the pipes when it received signals that the pressure was too high. This automated procedure operated as follows. A monitoring 'F' system received signals from temperature and pressure sensors located on fuel lines. The 'F' system transmits data to the PLC which raises audible and visible alarms in the control room. However, during the accident, the PLC was not connected and therefore did not automatically de-energise the pressure in the pipes.

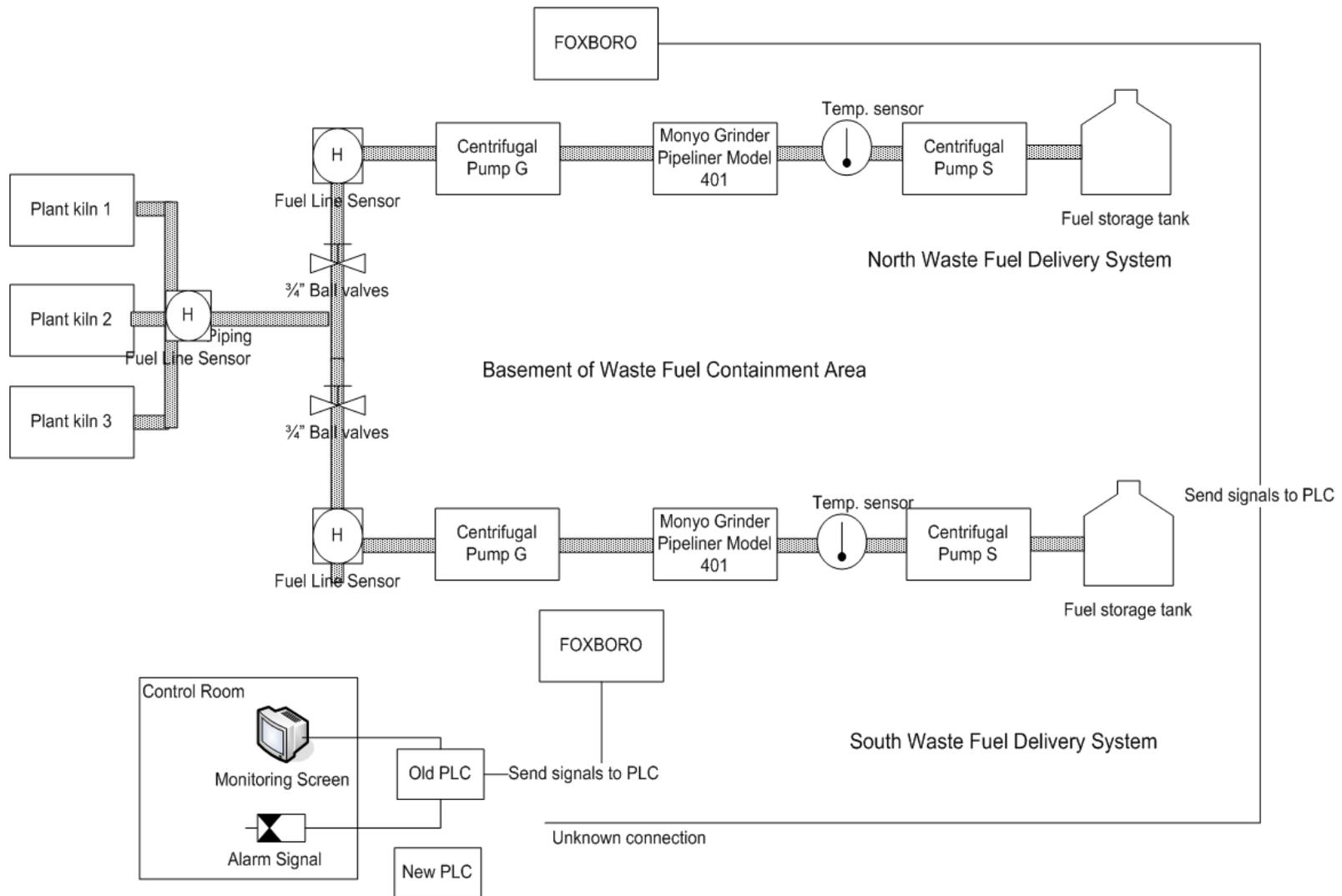


Figure 8-5. Simplified Layout Diagram of Waste Fuel Delivery Plant

3.3 Events Leading to the Accident

A seal on the north grinder overheated. The Kiln Control Operator and supervisor decided to switch waste fuel delivery systems from north to south. The worker switched delivery systems; however fuel did not flow to the plant kilns as planned. The personnel believed the problem was due to air being trapped in the south fuel pipes. They, therefore, bled the valves of the south system while the motors were running. In the meantime, due to the low pressure being sensed in the fuel lines, the automatic step increase program was increasing the speed of the motors on the south pumps in an attempt to increase pressure in the fuel line. These various factors combined to create a ‘fuel hammer effect’ in the pipe feeding the south pump. The hammer effect is caused by rebound waves created in a pipe full of liquid when the valve is closed too quickly. The waves of pressure converged on the south grinder. Figure 8-6 provides an overview of this ‘fuel hammer effect’.

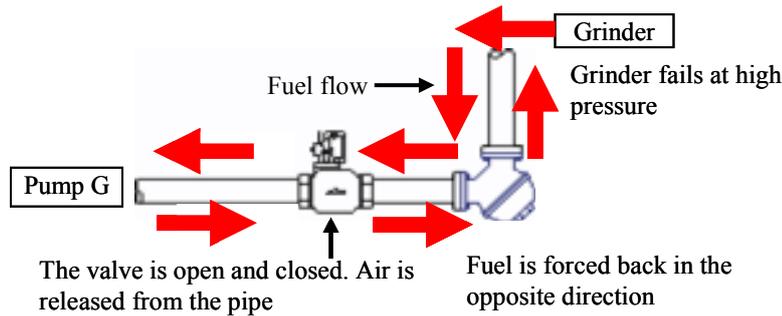


Figure 8-6. Fuel Hammer Effect

The grinder failed and fuel was sprayed on the two personnel. The fuel ignited and fire spread across the entire area. The supervisor managed to extinguish himself only. The worker later died in hospital.

4 Information Propagation through Models

The following diagram (Figure 8-7) provides an alternative view (with respect to the generic integrated modelling framework) of the way in which information propagates through the models applied to this case study. We can see that the system model is informed by task models. The diagram shows how the phases on task modelling, safety modelling (via safety cases), system modelling and barrier modelling become unified. The arrows between components provide an overview as to the kind of information that is related in these different perspectives of the same accident. For example, the task model can inform the system model by ensuring that the system model provides an interactive action as specified in the task model. We make explicit reference to this particular case study to give examples of the kind of information that is shared between models. We also highlight which figures in this chapter illustrate particular phases of the approach.

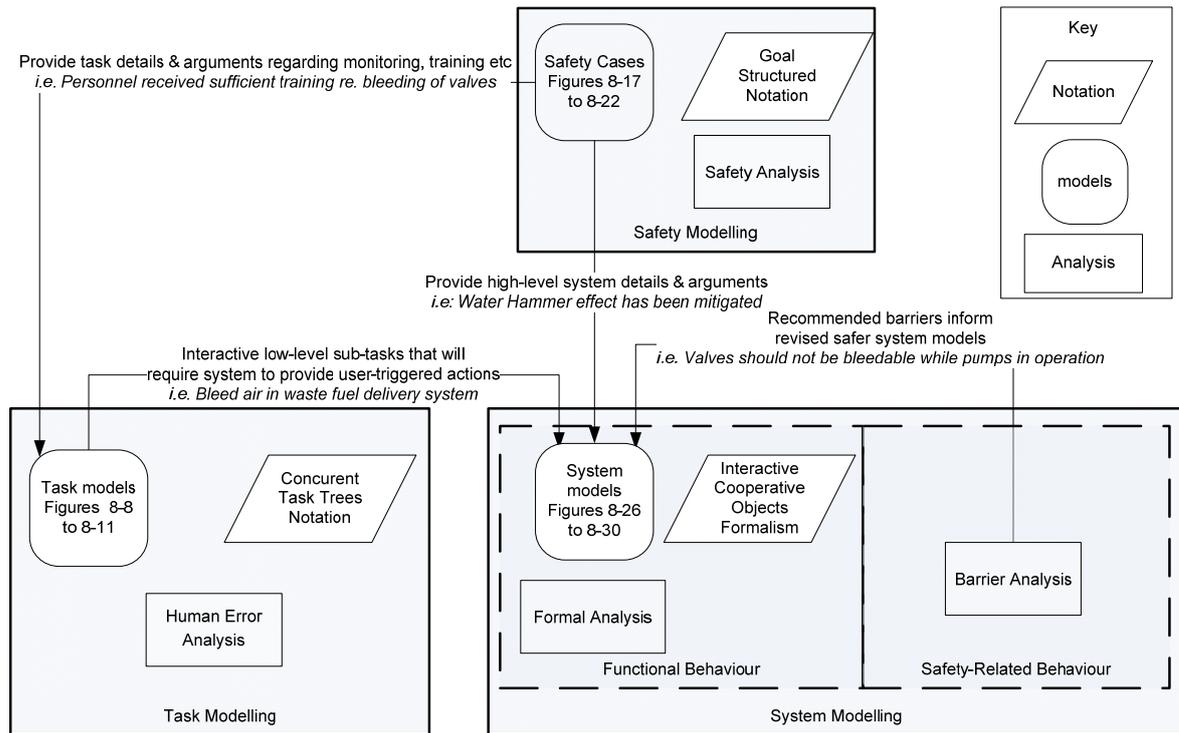


Figure 8-7. Overview of multidisciplinary perspectives and information propagation into models

5 Task Models of Operators

Task analysis techniques can be used to represent and reason about interaction with complex systems at a far lower level of detail than is typically possible with event-based modelling notations. In particular, recent years have seen the development of a number of these techniques that are specifically tailored for reasoning about human-human and human-system interaction in safety-critical environments. Such tools have not previously been linked to the forms of Barrier Analysis that might be conducted in the aftermath of an accident or near-miss incident. This is unfortunate because task analytical techniques provide a useful bridge between high-level event based models and the more sustained and detailed analysis that may be necessary in order to understand the complex ways in which human and automated defences are jeopardised within complex command and control systems. We have, therefore, used the ConcurTaskTree (CTT) notation (Paterno 1999) to model the collaborative task of switching from the north waste fuel system to the south in this case study. In next sections we first describe a task model for each individual; the Kiln Control Operator, the supervisor and the worker. We then produce an additional task tree indicating the collaboration that takes place between the individuals at a task level.

We now introduce the task models. Figure 8-8 models the Kiln Control Operator's involvement when an alarm is raised and the waste fuel delivery systems require switching. We can see that the Kiln Control Operator monitors the system until a problem is identified. The 'ID Prob(lem)' is an abstract task and interrupts the 'Monitor Sys' task. 'ID Prob' is decomposed into four parts. Alert type 1 and alert type 2 are necessary to specify the ways in which a problem can be identified. Either the Kiln Control Operator perceives a visual alarm and at the same time, but not necessarily, perceives an audio alarm. Or, the Kiln Control Operator perceives an audio alarm and at the same time, but not necessarily, perceives a visual alarm. In any case, the Kiln Control Operator then informs the supervisor of the problem. The Kiln Control Operator and supervisor discuss the problem and jointly decide to switch the waste fuel delivery systems. Once the waste fuel delivery systems have been switched, the Kiln Control Operator receives confirmation that the problem has been resolved. As can be seen, this level of analysis forces investigators to consider human-system interaction at a relatively low-level of detail. This is necessary given the increasing complexity of many interactive control systems. It is important also to observe that the Concurrent task tree in Figure 8-8 includes detailed synchronisation requirements, such as those denoted by the parallel composition operator 'III'.

The literature review presents a complete introduction to the ConcurTaskTrees notation. The key point is, however, that it would be difficult to express these more detailed concepts using more conventional event based notations.

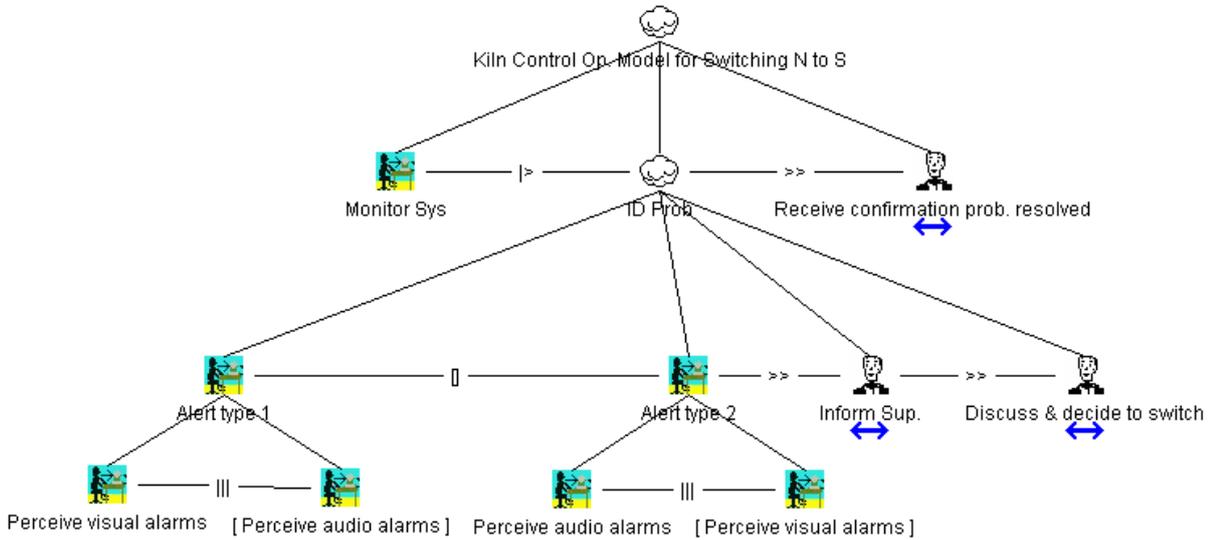


Figure 8-8. Kiln Control Operator Task Model

Figure 8-9 illustrates the tasks that the supervisor is involved in when the Kiln Control Operator informs him that a problem has been identified and the waste fuel delivery systems switching. Similarly, Figure 8-10 illustrates the tasks that the worker is involved in. As before, the key point here is that task modelling provides a convenient bridge between the initial event based modelling that often characterizes the early stages of an accident investigation and the more detailed output from a Barrier Analysis of system failures and human 'error'.

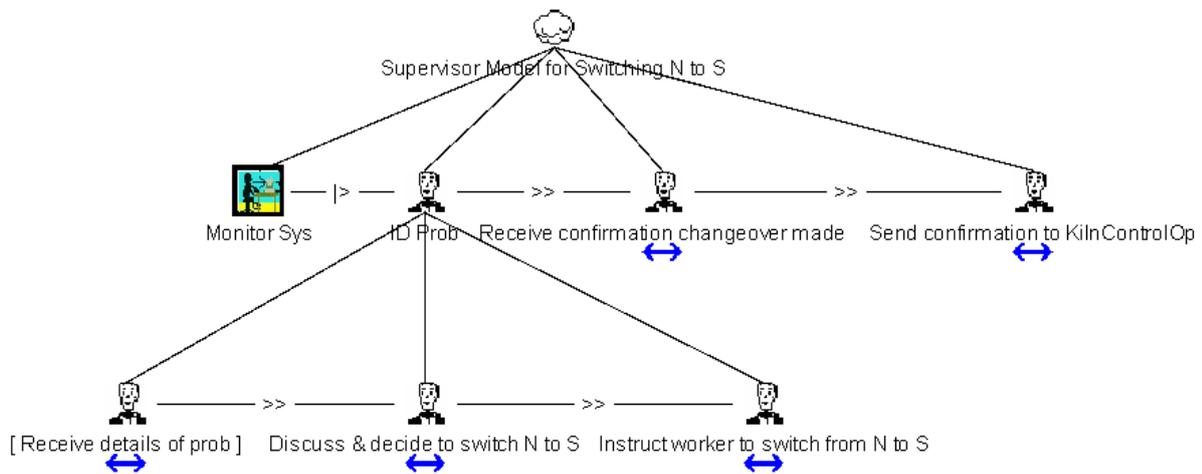


Figure 8-9. Supervisor Task Model

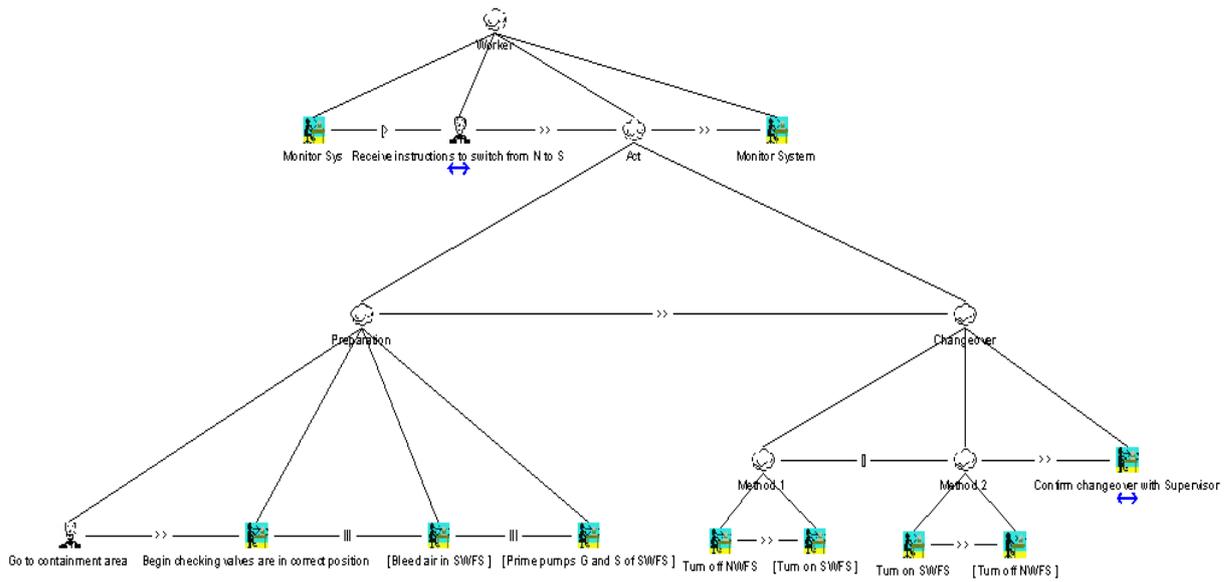


Figure 8-10. Worker task model

The previous diagrams have shown how ConcurTaskTrees can be used to model the detailed demands on individual operators during an adverse event. Each of these models can be composed to show how different activities overlapped over time. In addition, however, a number of tasks were common to each of these different individuals involved in the accident. Figure 8-11 illustrates these cooperative tasks.

As before, the key issue here is that accident investigators are provided with tools that can be used to move from high-level event based models through to more detailed representations of the interactive control tasks that faced the operators in the lead-up to an accident or near miss incident. The components of a ConcurTaskTree show how the individual activities develop over time. Common tasks that are shared between different members of a control team can also be represented in the same notation. This provides significant benefits over several alternate user-modelling notations, such as ICS, that provide a more detailed resource based model of cognitive and perceptual demands but which have the disadvantage of focusing on individual operators rather than teams of co-workers. It is important also to mention in passing that a number of alternate approaches to CTT could have been used here and the comparative benefits of these remain a focus for ongoing research.

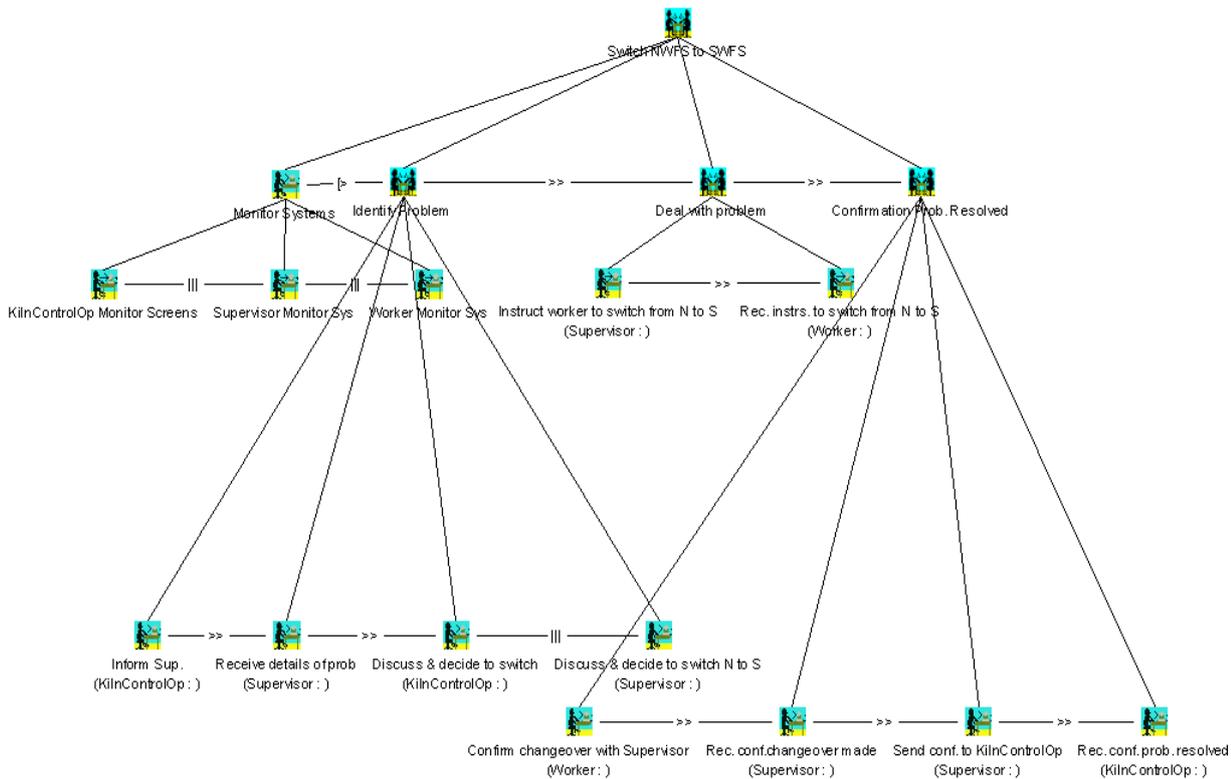


Figure 8-11. Cooperative Task Model for Switching from the North to the South Waste Fuel Delivery System

6 Human Error Analysis

Previous sections have argued that task modelling techniques (an in particular Concurrent Task Tree notation) enable investigators to take a more detailed look at the role of individuals and teams of operators in the events leading to an accident or incident. A further justification for using this approach during accidents and incidents investigation is that it has recently been extended and is now able to support the identification of different types of common interaction ‘failures’. Table 1 documents the outcome of this process for the ‘bleed air from valves’ worker subtask using the HERT. The key benefit of this human ‘error’ analysis however, is that analysts are guided in their investigation by the structure of operator tasks that are embedded in the task notation. A further benefit of this style of analysis is that not only does it help to identify potential interaction problems that led to a particular incident, it can also be used to identify other alternate problems that might jeopardise the future safety of an application but which did not arise in this specific accident.

Table 1 Skill-based human error analysis on ‘bleed air in SWFS’ worker subtask in Figure 8-10

Error Type	Example	Eventuality
Strong Habit Intrusion	Bleeds air from valves that he normally bleeds air from, but incorrect for the task	Don’t bleed
Strong Habit Capture	Intend to bleed air, but ‘remember’ too late in the process	Don’t bleed
Strong Habit Exclusion	Check valves with intention to bleed, but do not bleed them	Don’t bleed
Branching Error	Intend to bleed valves, check valve position, which leads to other activities	Don’t bleed
Overshooting a stop rule	Intend to bleed valves, but perform another routine activity	Don’t bleed
Programme Counter Failure	Intend to bleed valves, get distracted, do not bleed valves	Don’t bleed
Trip	Intend to bleed valves, something happens, do not bleed valves	Don’t bleed
Detached Intention	Intend to bleed valves, bleed incorrect valves	Don’t bleed
Environmental Capture	Intend to bleed valves, but perform another routine activity in same	Don’t bleed

		environment	
Lost Intention, Sidesteps	Multiple	Intend to bleed valves, perform several other routine activities, then realise didn't bleed valve	Don't bleed
Lost Intention, Failure	Retrieval	Intend to bleed valves, stare at them, forget why he is there	Don't bleed
Fumble		Intend to bleed valves, but do it badly	Don't bleed
Post Completion Error		Bleed valves, but do not close the valves	Bleed, leave open
Under Motivation		Know that he should bleed valves, but cannot be bothered too	Don't bleed
Description Error		Bleed wrong valves	Don't bleed
Input/Misperception Error		Bleed valves, but unsure of how to do it, so perform task incorrectly	Bleed not fully
Data Driven Error		Intend to bleed valves, but read different instructions, and perform task incorrectly	Bleed not fully
Associative Error	Activation	Intend to bleed valves, but think of another task, and perform task incorrectly	Bleed not fully
Omission		Do not bleed valves	Don't bleed
Repetition		Bleed valves repeatedly	Bleed - good
Reversal		Switch system on before bleeding air	Bleed too late
Insertion		Check another part of the system and bleed valves – no affect	Bleed - good
Replacement		Perform a different action, do not bleed valves	Don't bleed
Premature action		Bleed valves too early, before expected to do so	Bleed too early
Logical Phenotype Action	Correct	Bleed valves at correct time within process	Bleed - good
Logical Phenotype Forward	Jump	Bleed valves before checking positioning of valves	Bleed - good
Logical Phenotype Backward	Jump	Bleed valves after turning fuel system on	Bleed too late
Logical Insertion	Phenotype	Perform another action, unassimilated to the task	Don't bleed
Order Errors		Bleed air after turning fuel system on	Bleed too late
Over Motivation		Bleed valves too quickly and do not bleed them fully	Bleed too late

This form of analysis can help to identify a large potential set of interaction problems even for relatively simple sets of tasks. It is, therefore, often necessary to continue the analysis by grouping potential interaction problems into more generic types. In our case study, the analysis identified five different forms of problem. These included difficulties in the bleed task. We have added the “normal” expected interaction which is the successful bleeding of the valve which will be used to construct the model patterns.

1. bleed correctly
2. a failure to begin bleeding the system
3. a failure to completely bleed the system
4. a failure to close the bleeding valve
5. bleeding the system too early
6. bleeding the system too late.

Of these, a failure to completely bleed the system seems the most likely to have led to the hammer effect observed in the immediate interval before the accident. The fourth of these task problems can also lead to a hazard with fuel being leaked. Such observations illustrate the dual nature of task models in the aftermath of an adverse event. They can help to elucidate the particular chain of events that lead to an accident but can also be used to identify alternate failure scenarios that might cause future adverse events. It can be argued that because of this potential spreading of fuel the personnel are tempted to close valves as quickly as possible. But closing the valves too early (air remaining in the pipes) might trigger the hammer effect.

In many cases, such as the one we are investigating, a sophisticated command and control system has often caused the accident by performing an interlock or another measure as instructed, due to a confusion originating in another

part of the system. Complex interactions among system components are often unforeseen, and only recognised once an accident has occurred.

7 Model Patterns for Error Tolerance

Based on the identified interaction problems associated with the bleed valve task, we have produced 4 model patterns for error tolerance of the subtask. These are illustrated in the following figures.

Pattern 1 = 1 (bleed) or 2 (don't bleed) or 3 (bleed not fully) or 4 (bleed, leave valves open) or 5 (bleed too early) or 6 (bleed too late)

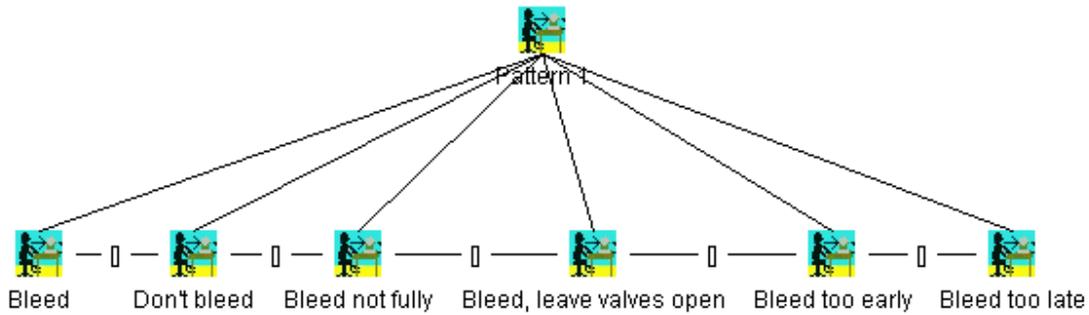


Figure 8-12. Pattern 1 for bleed valve subtask

Pattern 2 = 2 then 1, or 2 then 3, or 2 then 3 then 1, or 2 then 4, or 2 then 4 then 1

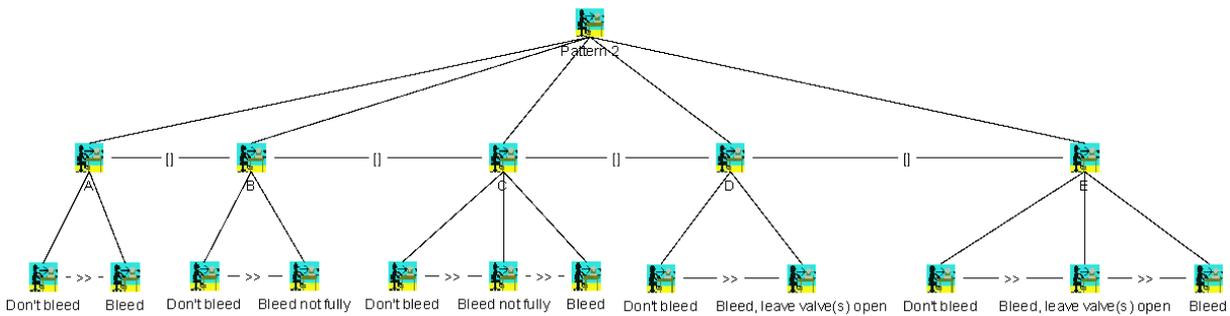


Figure 8-13. Pattern 2 for bleed valve subtask

Pattern 3 = 3 then 1, or 3 then 4, or 3 then 4 then 1

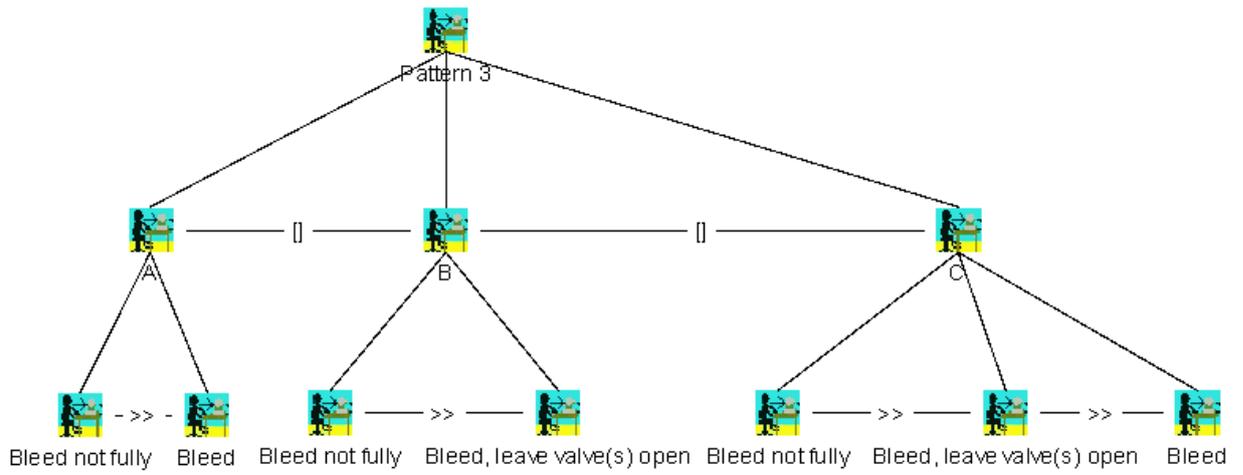


Figure 8-14. Pattern 3 for bleed valve subtask

Pattern 4 = 4 then 1, or 4 then 3, or 4 then 3 then

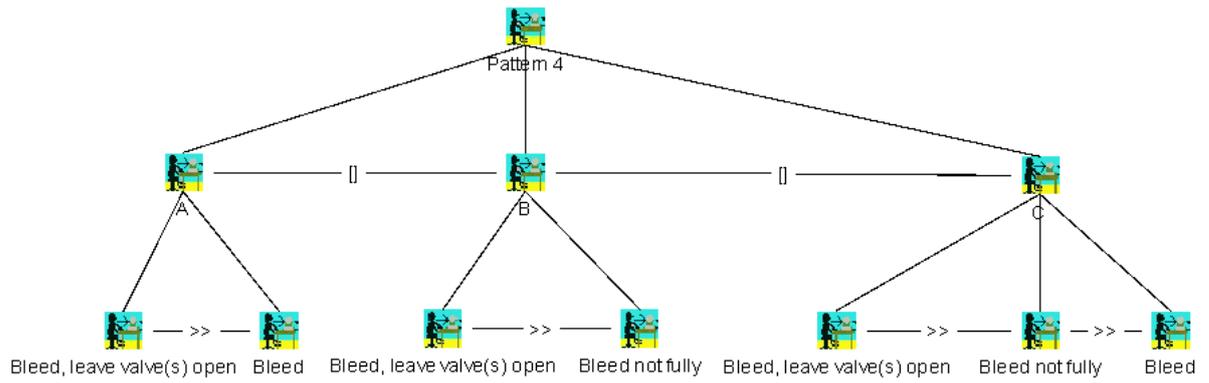


Figure 8-15. Pattern 4 for bleed valve subtask

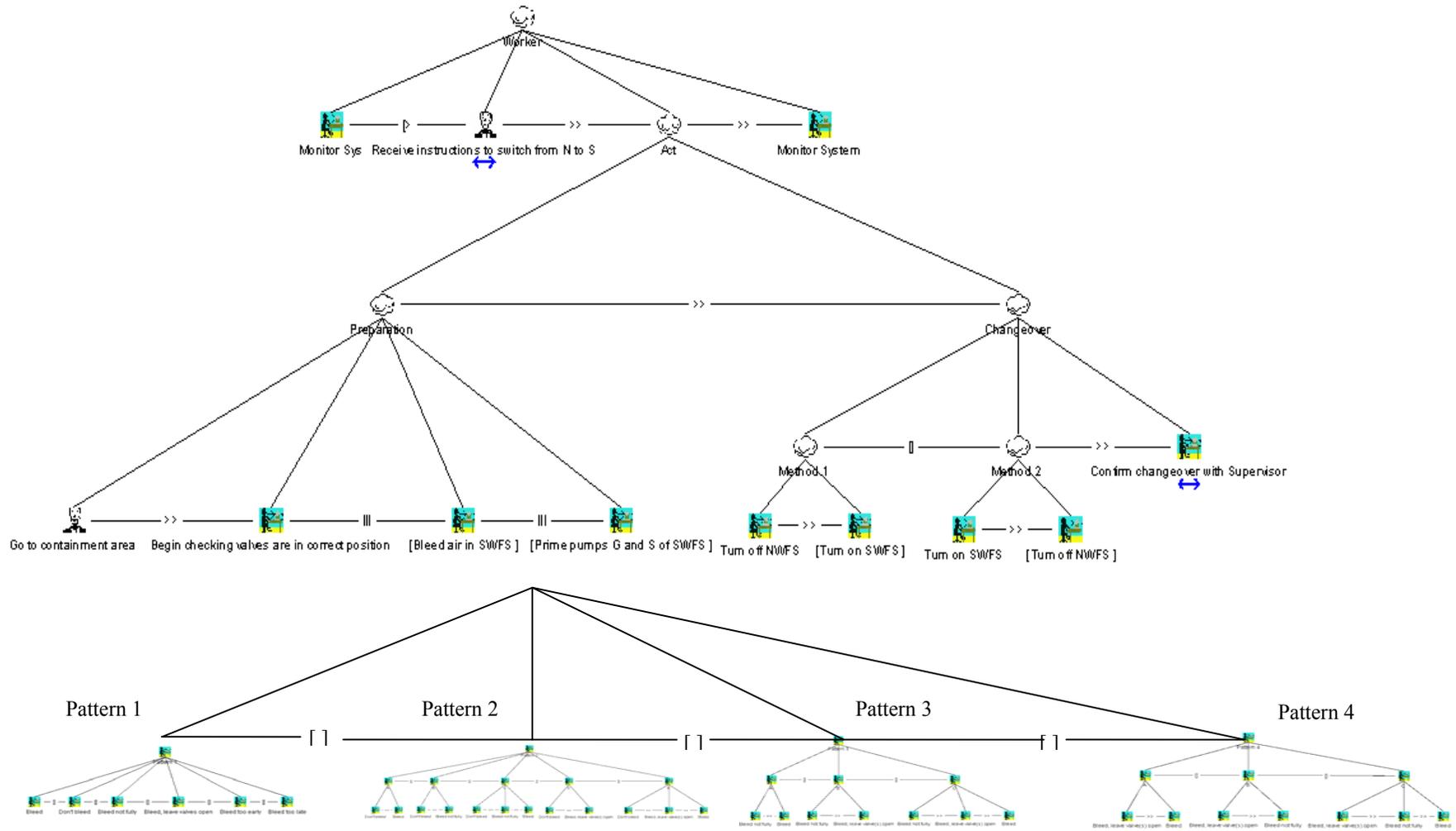


Figure 8-16. Plugging in patterns to “bleed air in SWFS” subtask of worker task model

8 Safety Case Analysis

Safety Cases are often required by regulatory agencies before they will grant approval for the operation of complex command and control systems. These cases describe the arguments that a company will use to convince the regulator that their proposed application is ‘acceptably safe’ (Bloomfield et al. 2004). For instance, they can include information on the risk assessment techniques that have been used. They may also describe the barriers that are deployed as a result of a risk assessment. Recently, a number of techniques that have been developed to support the construction of these documents have also been used to support accident and incident investigation (Greenwell et al. 2004). Accidents help to expose those safety arguments that were either unwarranted or fallacious. This is important because the same weaknesses that were exposed by a particular accident may also exist in other areas of an application process. Hence, by examining the broader Safety Case it is possible to identify further weaknesses in similar areas of a command and control system. Rather than develop this approach, the following pages present an alternative use of Safety Cases in accident investigation. The intention is to show that if an accident involved the failure of multiple barriers, it is also possible to trace the common causes of those failures back to the assumptions and arguments that are embodied within a Safety Case.

The diverse analytic techniques we have proposed in the framework need a way to be connected as to how they all relate to the accident that occurred. In order to do so, we used the Goal Structuring Notation to develop safety cases for each of the issues under investigation. This provides the reader with a higher-level view of how for instance, training, communication and a technical failure all relate to each other towards the compromise of a specific safety process.

There are two dimensions of this accident. Firstly, the workers interfered with the pump in a manner against manufacturer’s guidelines, setting off the chain of events that led to the ‘water- hammer effect’ occurring and thus the accident. The second dimension is the failure of the monitoring system to identify the pressure increase in the piping of the system, and initiating the control mechanisms that would have probably prevented the accident from happening. In this section we will discuss how and what assumptions about safety regarding these two aspects were false, and what implications they have about system design, by constructing Safety Cases about them.

Figure 8-17 illustrates the highest level of the Safety Cases, arguing that the waste management piping system (also referred to as the waste fuel delivery system in this chapter) is acceptably safe.

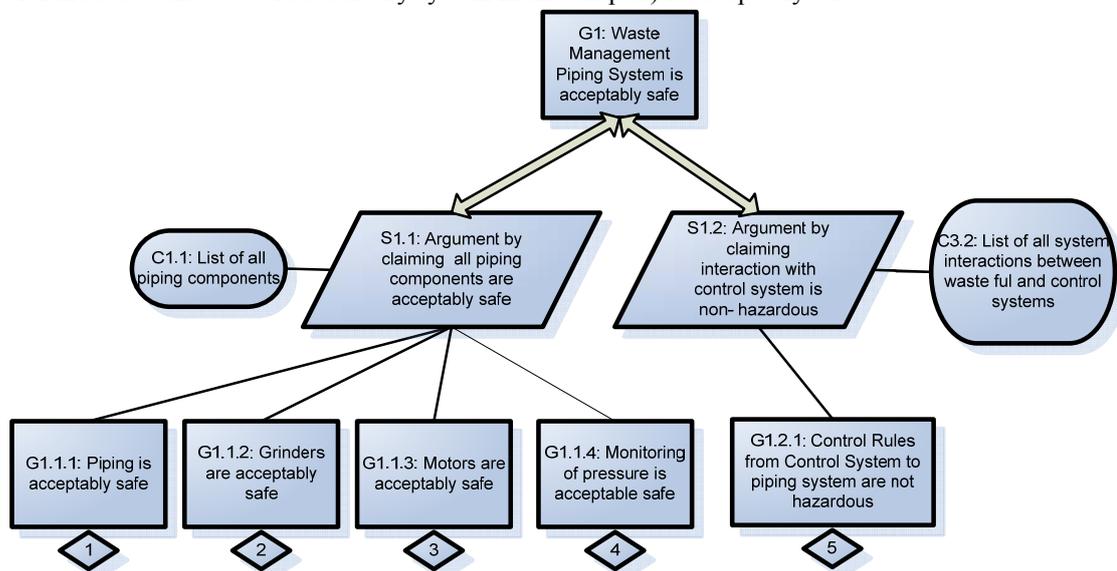


Figure 8-17. Safety Case Arguing the Safety of the Waste Management Piping System

The following subsections break parts of this Safety Case down illustrating how particular arguments were fallacious in terms of events leading to the accident.

8.1.1 Water-Hammer Effect

The ‘water- hammer effect’ refers to the generation of a pressure peak to a pipe. This is caused by a sudden fluctuation on pressure. It is a known phenomenon that can damage equipment and engineers have suggested a

number of approaches to prevent it from occurring at times where pressure is rising dangerously (Haiko et al. 2000). During the design of the fuel handling system we are investigating, the water-hammer effect was probably not taken into account. Although it is a result of high pressure, which is monitored by the command and control system, applications such as ‘pressure intensifiers’ (Haiko et al. 2000). have been suggested as barriers against the water-hammer effect. In any case, we can assume that when considering all possible hazards that could reach the piping, the water-hammer effect, along with other phenomena such as ‘mass oscillation’ could have been taken into account (Thorley 2004). These two pressure issues are represented in the Safety Case in Figure 8-18.

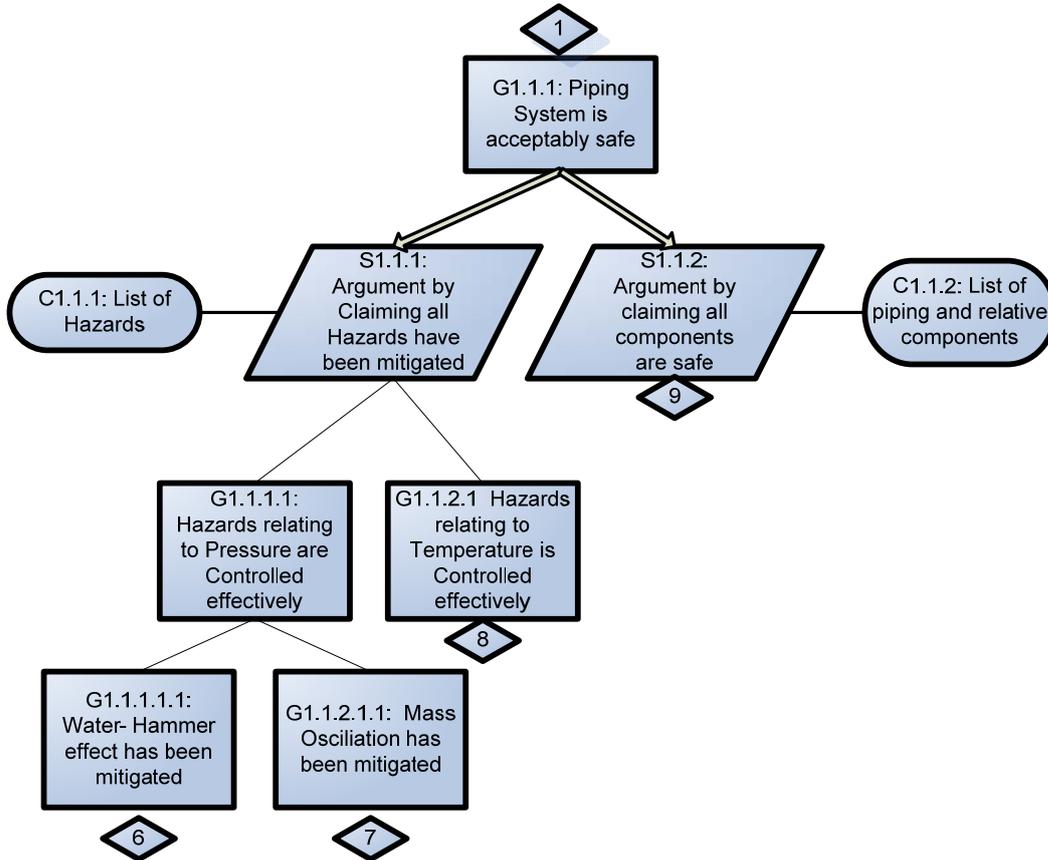


Figure 8-18. Considering all hazards that can arise from High-Pressure

Since the waste fuel delivery system is large and complex, we will not fully elaborate on the whole of the Safety Case previously shown in Figure 8-17. We have chosen to examine the command and control system that failed, as it is the most obvious factor that led to this accident. In order to do so, we have used Barrier Analysis which will follow after presenting the Safety Cases regarding the control system. High pressures and temperatures are the most important hazards for the monitoring system to mitigate. Therefore, the two primary safety functions are the *Monitoring of Pressure* and the *Monitoring of Temperature*. The function that failed was *Monitoring of Pressure* thus in Figure 8-19 we present how a Safety Case using GSN would look.

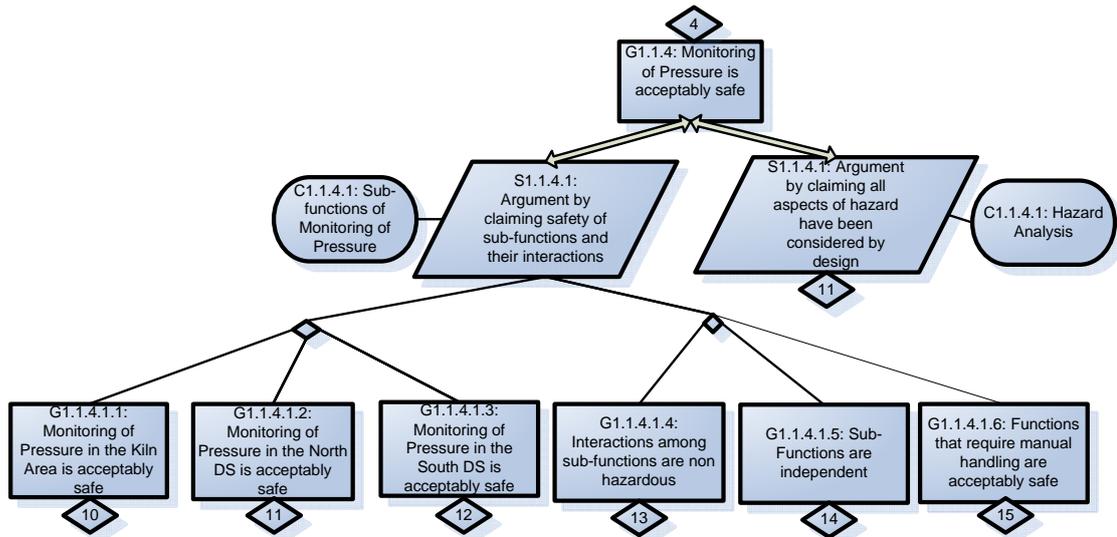


Figure 8-19. Safety Case for the Monitoring of Pressure function

The accident occurred due to high pressure in the south waste management system. The monitoring of pressure in the south waste management system encompasses the following material barriers: One sensor, alarms, the F-system and the PLC. However, it should also be stressed that barriers can also include the knowledge and competency of system operators, which suggests that human activities such as training and maintenance are very important for the success of the barrier system.

This safety case (Figure 8-19) argues that, in order to achieve safety of the ‘Monitoring of Pressure in the South Waste Delivery System’, all components of the monitoring system should be functioning properly, and that their interaction is non- hazardous. In order to establish this claim, the safety of each component would then be elaborated in a new safety case, and so on, until a safety goal cannot be broken down to any other sub-goals. We elaborate on the monitoring of pressure in the south delivery system in Figure 8-20.

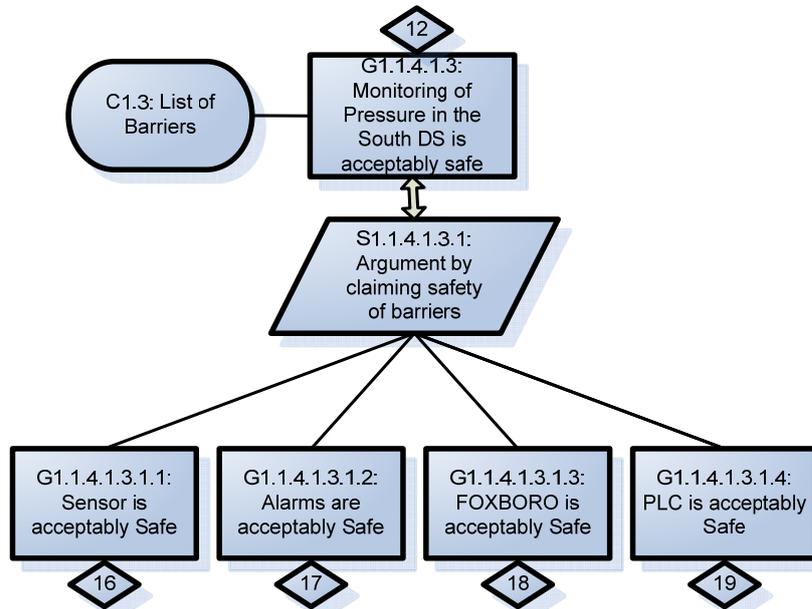


Figure 8-20. Safety Case for the ‘Monitoring of Pressure in the South Delivery System’ Function

8.1.2 Manual Handling

In this section we are examining issues regarding human operator communication and collaboration and while in the previous section 6, we looked at a method for analysing human ‘error’ in relation to the task models. In both, manual operations were guided by the monitoring and supervision provided by the Kiln Control Operator to the workers in the Waste Management System. The following safety case (Figure 8-21) presents a high level view of

the relation between monitoring and training, as they are both important for the collaboration and task performance.

Regarding monitoring activities requiring manual handling (Figure 8-21), the two workers involved in the accident were been watched by the control operator in the control room and had radio contact. However, their training did not include the manufacturer’s guidelines not to bleed air from the pumps while they were in operation, nor were they aware of the automatic shutdown process the system initiated on its own. Therefore, when discussing the role of human error in this case, it should be taken into account that the workers were not trained accordingly. In fact, other workers had performed the exact same activities, ignoring manufacturer’s guidelines in the past.

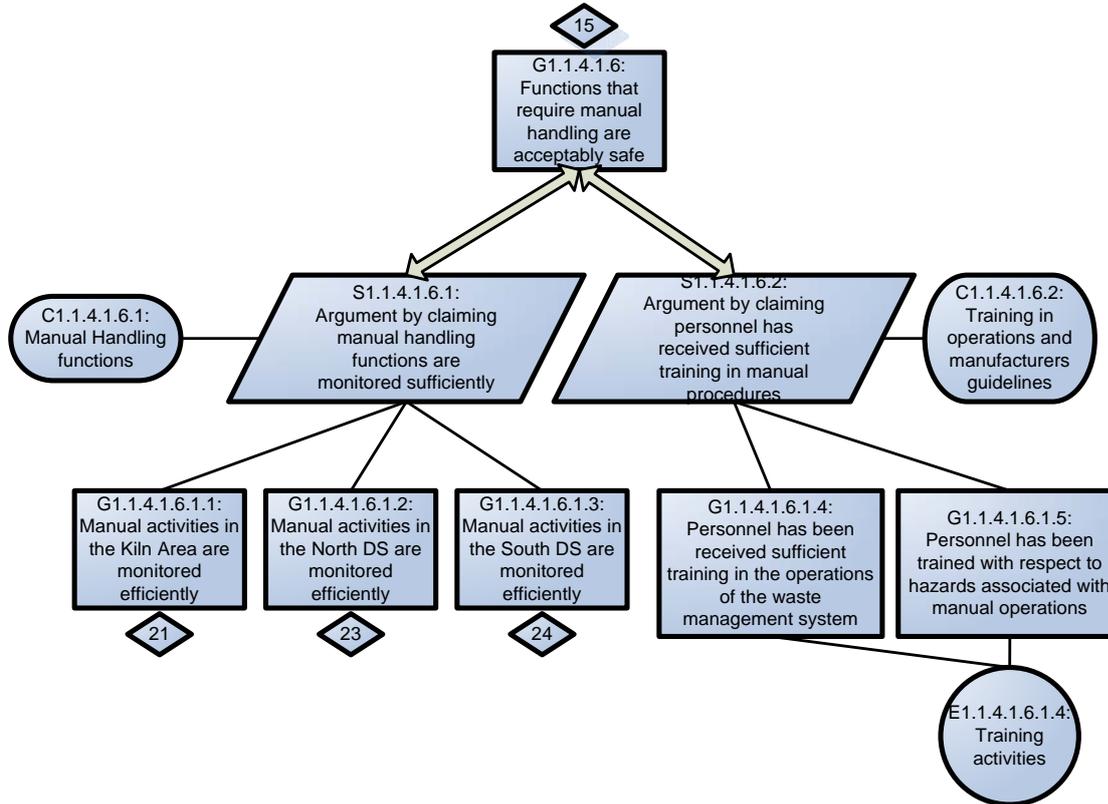


Figure 8-21. Safety of Manual activities

8.1.3 Failure of the Command and Control System

As mentioned, safety cases consist of claims that are based on the design and use of the system. For instance, in order to argue that a component of the system is ‘acceptably safe’, claims about training practices regarding the use of the specific component, as well as maintenance adhering to manufacturer’s guidelines would also have to be incorporated in the argumentation (Figure 8-22). In this way, we can visualize how technical, human and organizational aspects are interconnected in order to assure safety. On the other hand, we can also see at a high level how these issues relate to each other in a failure.

The safety case in Figure 8-22 argues the safety of the PLC, the component of the Command and Control System that was identified in the accident report as a major cause of the accident. Although this can be viewed as a technical failure, it also has organizational implications: Maintenance and testing were not carried out as they were supposed to be. This allowed for the technical failure to occur, as the PLC was not connected for a period of three months.

Figure 8-22 illustrates part of the argument that might be used to demonstrate that the PLC does not contribute to the risks associated with the Cement application. As mentioned, however, investigators can proceed by examining the evidence that these arguments rest on. In this case, suspicion falls on the maintenance and testing of the PLC component. Three months prior to the accident, a new PLC was installed. The interface units were not however updated. Subsequent examination of the data logs revealed that the PLC did not receive data from the F-system sensors at the time of the accident. It could not therefore, command an automatic shutdown.

Figure 8-22 illustrates part of the argument that might be used to demonstrate that the PLC does not contribute to the risks associated with the Cement application. As mentioned, however, investigators can proceed by examining the evidence that these arguments rest on. In this case, suspicion falls on the maintenance and testing of the PLC component. Three months prior to the accident, a new PLC was installed. The interface units were not, however, updated. Subsequent examination of the data logs revealed that the PLC did not receive data from the F-system sensors at the time of the accident. It could not, therefore, command an automatic shutdown.

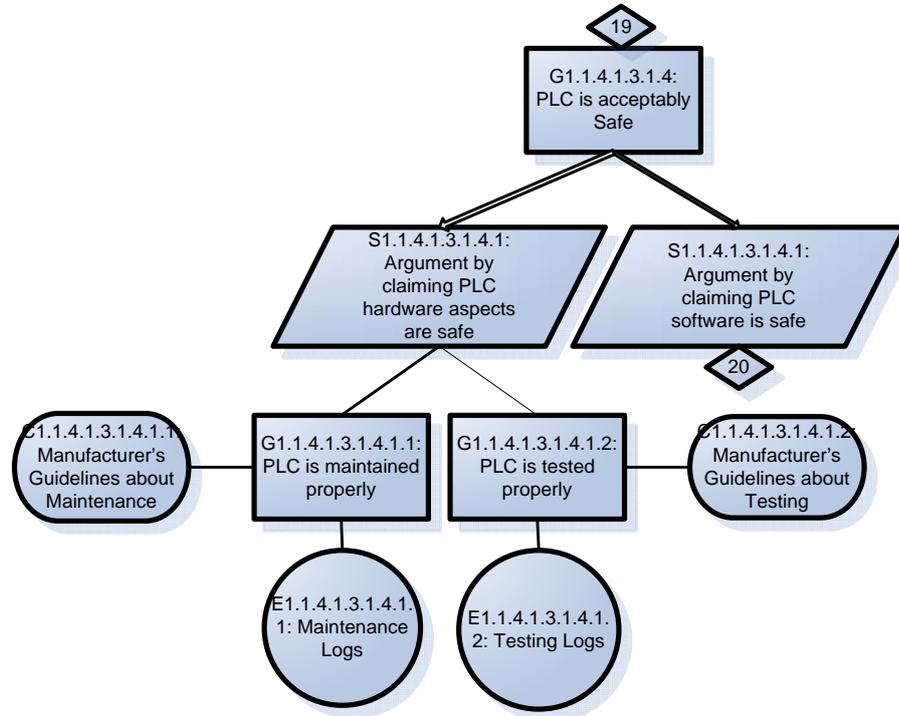


Figure 8-22. Arguing the safety of the PLC

This section on Safety Cases has extended the application of GSN from safety case generation to provide a very high level framework for the analysis of accidents and incidents. It can be used to identify any underlying problems in the arguments that were intended to demonstrate the safety of a complex system. It also encourages analysts to think beyond the specific events in an accident to consider whether they are symptomatic of wider failures. For instance, a failure to maintain key components in a programmable system might indicate the need to look for similar failures in other areas of an application by looking for similar maintenance arguments to G1.1.4.1.3.1.4.1.1 (PLC is maintained properly) in other areas of the GSN.

Unfortunately, the level of analysis supported by the GSN is not detailed enough to work out the specifics of any subsequent redesign. It is for this reason that we advocate a multidisciplinary approach to safety-critical interactive systems design and redesign to deal with the complexities.

8.2 Barrier Analysis based on Safety Case Analysis

Following the Safety Case analysis, we can proceed with barrier analysis in order to identify the way in which potential hazards threaten the functional components of a safety-critical application. The investigation typically proceeds through the elaboration of a number of high-level tables such as those illustrated in this section.

As can be seen, Table 8-1 shows the products from an initial analysis of the case study incident. It distinguishes the North and South Waste Fuel Delivery Systems (NWDS and SWDS) from the kilns and piping that can be considered as targets that will be affected by a potential hazard. The barriers that protect these targets include the sensors, control systems and human operators mentioned in previous paragraphs. Barrier Analysis proceeds by analyzing potential reasons why each of the barriers might fail. This can be done at a relatively high-level abstraction during the early stages of an accident investigation so that analysts do not prematurely close down a particular avenue of investigation into an adverse event.

Table 8-1. Hazards, targets and barriers

Hazard	Target	Barriers
--------	--------	----------

Temperature and pressure increase	Kiln area, NWDS, SWDS, point of merge	Sensors, alarms, F-system, PLC, personnel training
-----------------------------------	---------------------------------------	--

Table 8-2 presents the results of such a preliminary application of the approach to our case study. The way with which the issues presented in Table 8-2 are interconnected is best illustrated with the Safety Case represented in Figure 8-22.

Table 8-2. Detailed barrier analysis and correspondence to Safety Cases

Barrier	Reason for failure	Safety Case Component
Sensors	There were no sensors in the waste fuel containment area Sensors for the fire suppression system were mounted much higher than they should be, thus being unable to detect the fire in a timely manner	G1.1.4.1.3.1.1 (Figure 8-20) S1.1.4.1 (Figure 8-19)
Alarms	There were no alarms in the waste fuel containment area	S1.1.4.1 (Figure 8-19)
F-PLC	F- functioned OK throughout PLC connectors were not installed properly - Maintenance and testing had been postponed	G1.1.4.1.3.1.4.1.1 (Figure 8-22) G1.1.4.1.3.1.4.1.2 (Figure 8-22)
Personnel training	Involved workers had not been trained regarding the manufacturer’s warning not to bleed air while the pumps were operating. Involved worker had not been trained to recognise that pumps should shut down automatically	G1.1.4.1.6.1.4 (Figure 8-21) G1.1.4.1.6.1.6 (Figure 8-21)

9 Communication Sequence

As can be seen, Safety Cases provide important clues about the manner in which safety-critical systems can fail. However, they are poorly situated in the context and everyday events that lead to an accident or near incident. Investigators are often forced to conduct lower level interviews and elicitation exercises to build up a more complete picture of what happened. The previous introduction to our command and control case study provides a false impression of the mass of contextual and causal information that must be considered following an accident or near miss incident. For example, Figure 8 shows how sequence diagrams can be used to provide a basic overview of the exchange of information between operators and their systems. The Kiln Control Operator noticed that a seal on the north grinder was overheating and therefore informed the supervisor. Together, the Kiln Control Operator and supervisor decided to switch the Waste Fuel Delivery System from the north to the south circuit. The supervisor instructed the worker to switch Waste Fuel Delivery Systems from north to south. The worker acknowledged the instructions, switched delivery systems and confirmed the changeover to his supervisor. The supervisor acknowledged the switch of systems and informed the Kiln Control Operator. The Kiln Control Operator noticed that fuel was not being delivered to the kilns and therefore informed the supervisor. The supervisor and worker agreed there must be a problem with air in the pipes and decided to bleed the ¾” ball valve and the valves located on pump S and G. The supervisor then radioed the Kiln Control Operator to find out if fuel had begun flowing into the kilns. The Kiln Control Operator informed the supervisor it had not. The supervisor and worker noticed the south grinder begin to shake and vibrate.

Figure 8-23 illustrates the event-based sequence diagram that can be used to map out *what* happened in the lead-up to an adverse event. There are a host of similar notations for modelling accidents or incidents, these range from Multilinear Event Sequencing through to Events and Causal Factors chains. However, all of these techniques suffer from significant limitations when they are applied to complex command and control system failures that involve significant team-based interaction. For instance, the previous analysis included observations about the Kiln Control Operator and Supervisor decision to switch the Waste Fuel Delivery System from the north to the south circuit. The decision-making process that led to this intervention cannot easily be illustrated using such sequence diagrams. Alternative techniques, including cooperative task models must be recruited to represent this as we have shown in section 5.

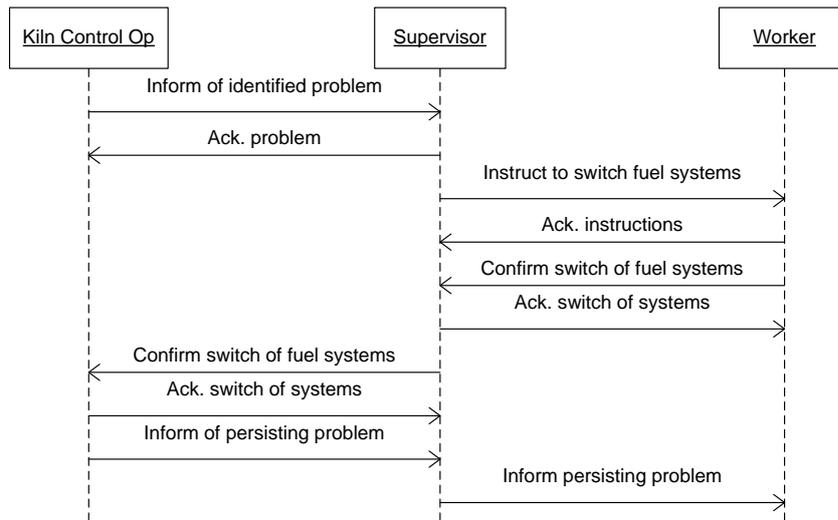


Figure 8-23 High-level Sequence Diagram

10 Events and Causal Factors Analysis

It was necessary for the purposes of our approach to have an overall perspective of how different factors and conditions, such as pressure, temperature and different system states evolved throughout time towards leading to this accident. Events and Causal Factors Analysis (ECFA), does exactly the above; in a chronologically sequential representation of events, ECFA can provide with the analyst an overall picture of how different factors relate to one another and how they all contributed to the accident. The approach is also known for its flexibility of interpretation of data, something that was helpful for the purposes of our overall design approach. Figure 8-24 illustrates an ECFA for the events and conditions leading to the grinder explosion. The ECFA will be used to inform the system model to ensure that system model accurately represents the main events leading to the accident and later to ensure that our improved system model does not allow the same sequence of events to occur.

Building on our previous publications in which we describe the method for our multidisciplinary perspective to accident investigation and informing the system model, we have presented here, an approach for informing the re-design of an interactive safety-critical system with information from incident and accident investigation. Events and Causal Factors Analysis charting has been used. This technique has been widely adopted, for instance by organisations as diverse as NASA, Eurocontrol and the US Department of Energy. The results of this initial analysis are then used to inform more detailed systems modelling using Petri Nets. It is important to stress that other investigatory tools might have been used just as there is a broad range of alternate techniques for systems modelling.

It is clear however, that some events, conditions and actions cannot easily be represented in the system model. These would include human cognitive processes for example. In contrast, these events are more conveniently considered in the natural language annotations of the investigation techniques rather than in the places and transitions of a Petri Net. For this reason, the two techniques complement each other and reveal different perspectives on the accident.

The ‘official’ accident report argues that a water-hammer effect was the most likely cause. Busse (Busse 1999) notes that the identification of hazardous human involvement in accidents does not, however, imply the identification of the actual ‘cause’ of the accident (Busse 1999). The grinder may have exploded without the intervention of users bleeding valves while the pumps were in operation.

Our approach is driven by the information contained in the accident report. These documents provide a convenient starting point for the construction of ECF diagrams. They often include these diagrams as a by-product of the investigation process. There is, however, a danger that our approach will bias redesign work towards the particular causes identified in an official report rather than the wider contextual factors that can easily be overlooked in the aftermath of an accident or incident (Koester, 2001). It is apparent that our integration of accident investigation and detailed systems design techniques can only form one part of a wider approach to redesign in the aftermath of adverse events.

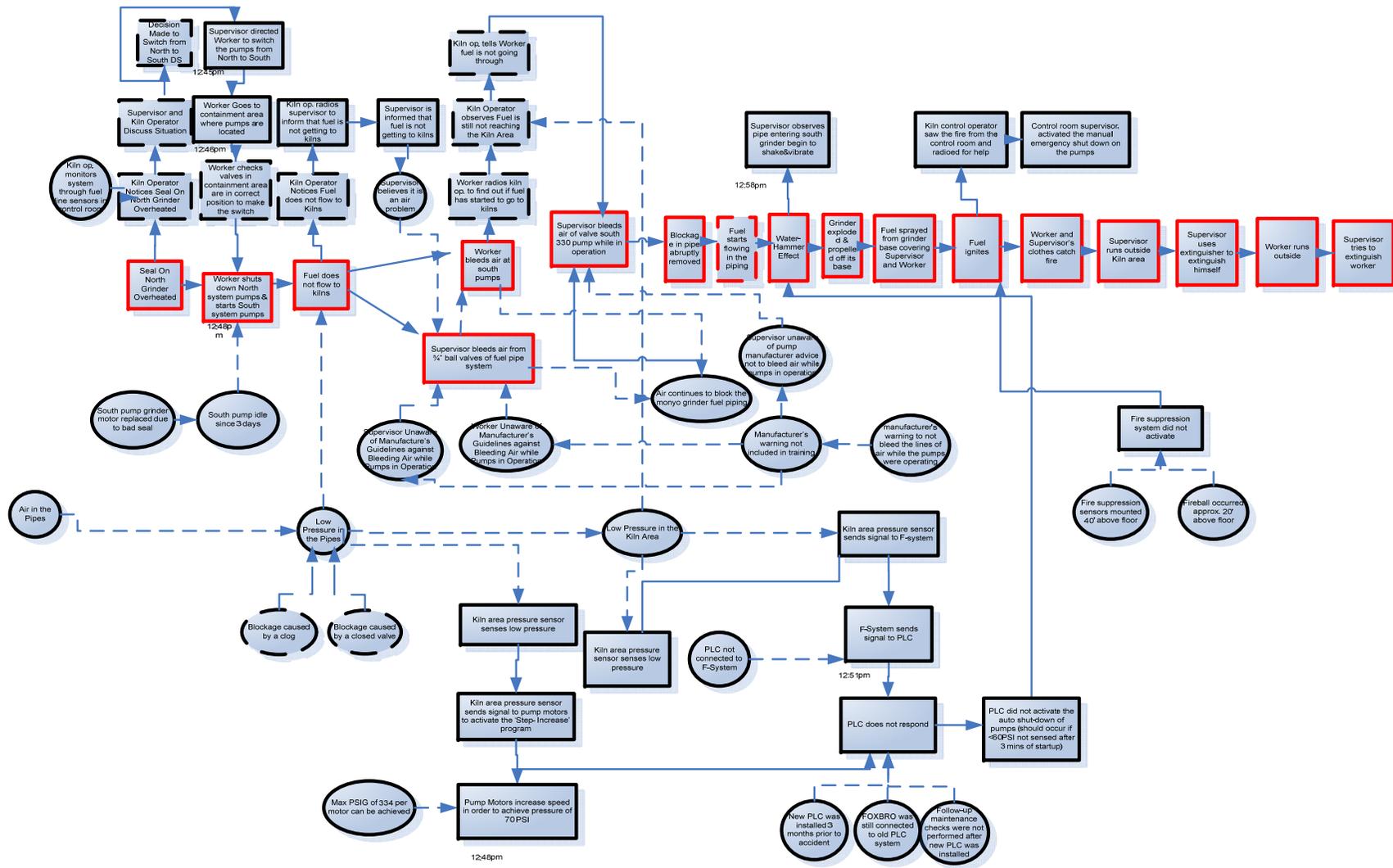


Figure 8-24. ECFA Chart of the Accident

11 System Model of WFDS

The introduction argued for a complementary approach to the investigation of command and control system failures. The event based model (ECFA) focussed not simply on the interaction between individual operators but also with different automated systems. In contrast, the task models have looked more narrowly on the demands that faced different system users.

In the immediate aftermath of an adverse even, it is important to piece together the available evidence in a systematic manner. One traditional means of doing this is to conduct a functional analysis of the system being considered. In our example, the waste fuel management system was composed of a monitoring system that has been installed to monitor high pressures and temperatures in the two separate systems and in the kiln area. The monitoring process is performed with means of sensors that report any anomaly of the two variables to the command and control application, known as the F-system previously described. The F-system monitors and records all operating parameters and is connected to a PLC which performs the automatic start-up/shutdown process. The F-system is also connected to audible and visible alarms (see Figure 8-25).

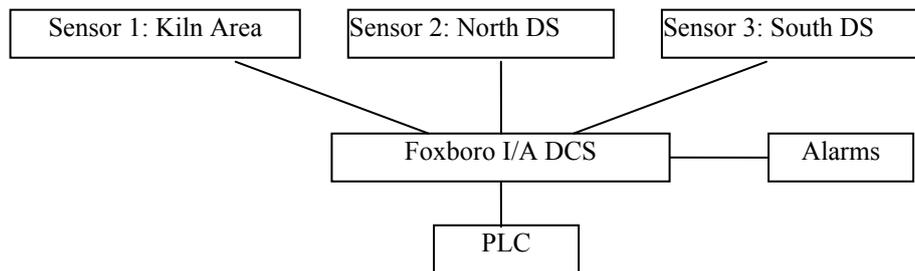


Figure 8-25. Connections among components in the Monitoring System

The following sections, therefore, extend this analysis and adopt a more narrow focus on a detailed model of the system itself. In order to do this, we must go beyond the task modelling facilities provided by the ConcurTaskTrees notation. One way of doing this is to exploit the Interactive Cooperative Objects (ICO) formalism (Navarre 2001) based on Petri nets. As with the previous task models, this is only one of several system modelling techniques that we might have used. The decision to adopt this approach is largely justified by the way in which Petri nets have been used in conjunction with Concurrent task Trees to look at both the system and operator aspects of interaction in the design of safety-critical systems. A further justification is the degree of tool support provided for these complementary modelling techniques using the Petshop environment (Navarre et al. 2003) and the fact that this is one of the only notations able to model and analyse interactive system behaviours.

The waste fuel delivery system has been modelled using the ICO notation to deliver a relatively high level of detail of the behaviour of the system. In the next pages, these models are used to denote that operators can command the system to send fuel to the plant kilns via the north or the south waste fuel delivery system. The system model has been built based on the schematic diagram (see Figure 8-5) of the waste fuel delivery system and reflects the same topology. Thus, we will describe the behaviour of each part of the system in terms of a Petri net and its functionality within the plant. Please note, unless otherwise stated, each component has the same functionality in the north and the south of the plant

Petri nets are widely used in systems engineering. This does not, however, eliminate the concern that a complementary approach based on Petri nets and ConcurTaskTrees may impose unrealistic expectations on the skill sets that would be required for any investigation. Equally, a range of recent initiatives have begun to recognise the importance of explicit and detailed training if accident investigation agencies are to cope with the increasing complexity of many safety-critical application processes. For example, the NTSB recently opened its Academy for investigators in Virginia. This section aims to present the system model in order (in later sections) to show how it can be fruitfully combined with the other elements of the method. An informal reading will be provided for each of the nets introduced in the remainder of the paper. In the following analysis, we use the following symbols:

- States are represented by the distribution of tokens into places 
- Actions triggered in an autonomous way by the system are represented as  and called transitions

- Actions triggered by users are represented by half bordered transition t1

Fuel Storage Tanks

Figure 8-26 a models the ability to switch from the north waste fuel storage tank to the south waste fuel storage tank. The north tank has been set to ‘open’ by default to correspond to the accident report which means there is a token in the NFTankOpen place (N signifies the north waste fuel system) and in the SFTankClosed place (S signifies the south waste fuel system). The manual shut off valve allows the user to switch from the north to the south fuel tank and vice versa. According to the initial state, only transition ManShutOffValve_2 can be fired (transition ManShutOffValve_1 is shown as greyed out). If fired, a token is removed from place NFTankOpen and from place SFTankClosed (that are the input places for transition ManShutOffValve_2. Similarly a token is set into NFTankClosed place and the SFTankOpen place (that are the output places of transition ManShutOffValve_2).

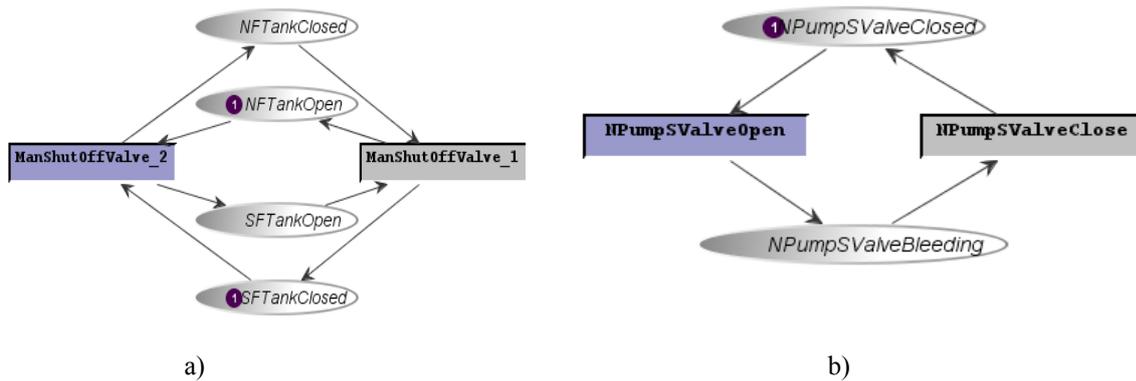


Figure 8-26. a) Fuel storage tanks (Applied to the north and south) b) North Pump S Valve

Pump Valves

Each pump consists of a motor to pump the fuel through the pipes and a valve. In total there are four pumps (two in the North called NPumpS and NPumpG and two in the south called SPumpS and SPumpG). The motor can be ‘on’ or ‘off’ and the valve can be ‘open’ or ‘closed’ for the purpose of bleeding air. Figure 8-26b describes the behaviour of the valves. The valves have been set by default to be ‘closed’, thus there is a token in the NPumpSValveClosed place and the NPumpSValveOpen transition is available. Opening the valve results in a token being set in the NPumpSValveBleeding place and the NPumpSValveClose transition becomes fireable. Please note, the 3/4” ball valves operate in the same way as the valves at pump S and G.

Pump Motors

The Petri net model describing the behaviour of the pump motor (Figure 8-27a) is composed of two connected parts, the motor and the corresponding fuel flow, i.e. when the motor is running, the fuel flows provided the tank is open. This condition is not modelled here as we only describe for now the independent behaviour of each component. The global Petri net modelling the interconnection of components is dedicated to the explicit description of such conditions.

The north pump motors are set by default to ‘on’ to allow the fuel to flow to the kilns since the model also begins with the north fuel tank ‘open’. Thus there is a token in place NPumpSMotorRunning and in place NPumpFlowsFuel. If the NPumpSMotorOff user action transition is fired, a token is placed in NPumpSMotorStopped and the fuel therefore stops flowing and a token is set in place NPumpNoFuel.

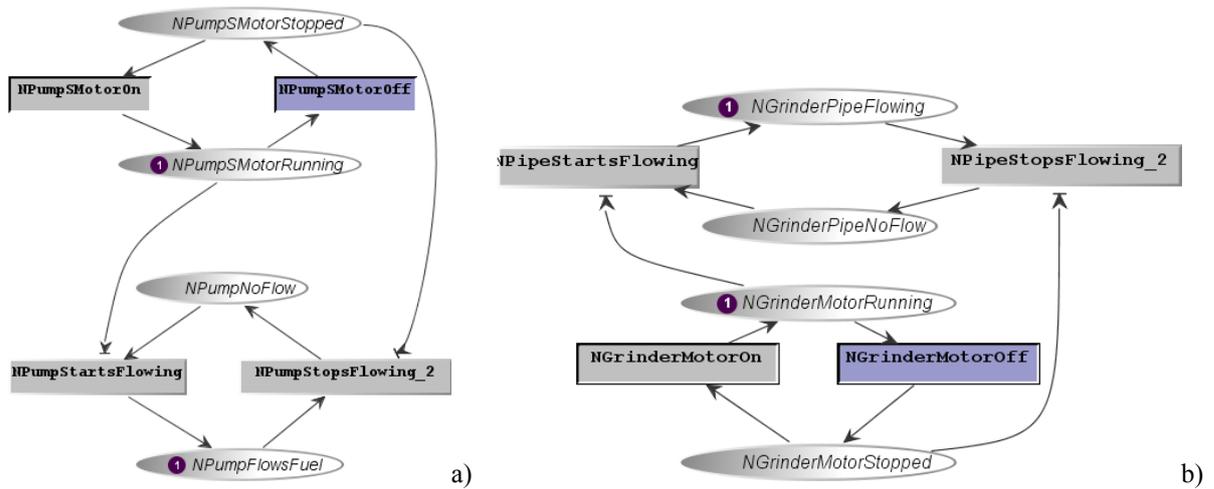


Figure 8-27. a) North Pump S Motor b) The North Grinder

Grinders

The grinders (Figure 8-27b) do not contain a valve as the pumps do. They have been modelled as a motor which can be on or off and an indication as to whether the fuel is flowing through the grinders or not. The model shows the north grinder motor is set by default as running and therefore the north grinder pipe is flowing. The user can trigger the NGrinderMotorOff transition. In which case, a token is set to place NGrinderMotorStopped and to NGrinderPipeNoFlow (as transition NPipeStopsFlowing_2 is automatically triggered when there is a token in place NGrinderMotorStopped and in place SGrinderPipeFlowing).

Plant Kilns

Fuel flows to the three plant kilns via the north or the south waste fuel system. The kilns have been modelled in two independent parts (see Figure 8-28) describing four possible states: receiving fuel from north, from south, from both or not receiving fuel at all. According to the accident report, initially the kilns are receiving fuel from the north. This is modelled by setting a token both in place PlantKiln123ReceivingN and in place PlantKiln123NotReceivingS.

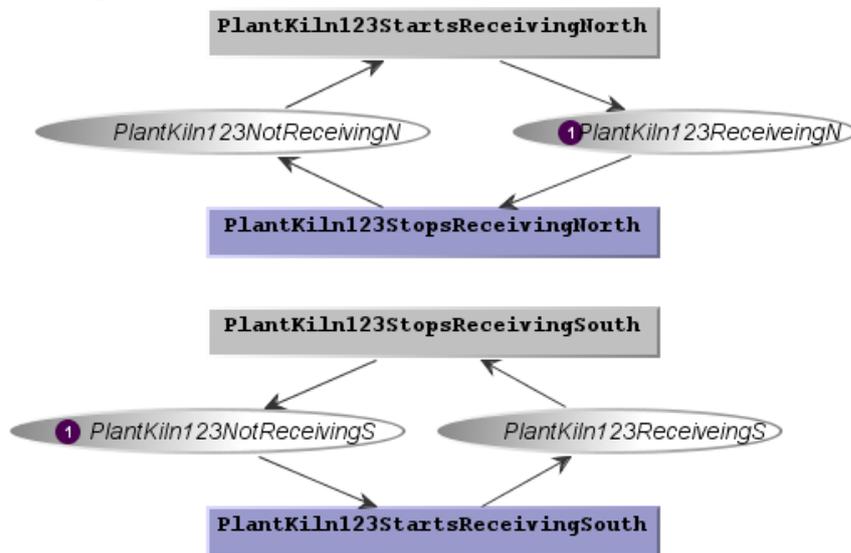


Figure 8-28. Fuel Plant Kilns

These various Petri nets have been connected to model the entire system (see Figure 8-30). The connections are based on fuel flow and are meant to model it. Fuel can either stop flowing if a motor is not running or fuel from the previous component is not flowing. For example, fuel cannot flow from a grinder into a pump G, even if the grinder motor is running and the correct fuel storage tank is open, if the fuel is not flowing from the previous pump S.

PLC

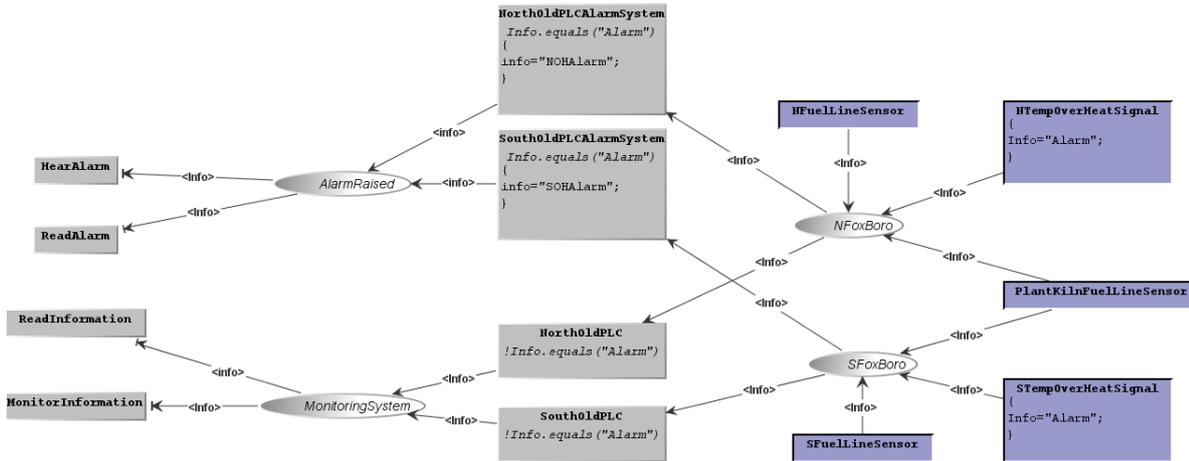


Figure 8-29. Petri net model of the PLC

This Petri net model of the PLC describes how information is received by the PLC and how it is processed in order to trigger alarms that are, in turn, meant to be sent and dispatched to the control room.

Dark transitions on the right-hand side of Figure 8-29 show the set of sensors that are connected to the PLC. This is modelled using timed transitions as the information available in the sensors is periodically received (or requested) by the PLC. Such information can be either related to the current flow of fuel in the plant through three sensors, one for North line (transition `NFuelLineSensor`) one for the South line (`SFuelLineSensor`) and one for the kiln (`PlantKilnFuelLineSensor`). The other two transitions provide the PLC with temperature information for the North line (transition `NTempHeatSignal`) and the South line (`STempHeatSignal`).

When information is received from these sensors it is then processed in order to decide (according to its current value) how it should be presented. This processing is done by the 4 transitions in the middle of Figure 8-29. Using preconditions (shown in the transitions) like “`Info.equals(“Alarm”)`” transition `NorthOldPLCSystem` will raise a alarm called “`NOHAlarm`”. This is done by setting a token with the alarm type as value. This token can then be used by the controller using the two transitions modelling the monitoring activity namely “`HearAlarm`” or “`ReadAlarm`”. If the received information is not an alarm (precondition “`!Info.equals(“Alarm”)`”) then no alarm will be raised but, instead, the information will be forwarded to the monitoring system (via transition `SouthOldPLC` (if the information has been provided by sensors from the South line)). Then the controller will be able to access this information (a token will be set in place `MonitoringSystem`) through the transitions `ReadInformation` and/or `MonitorInformation` (at the left-hand side of Figure 8-29).

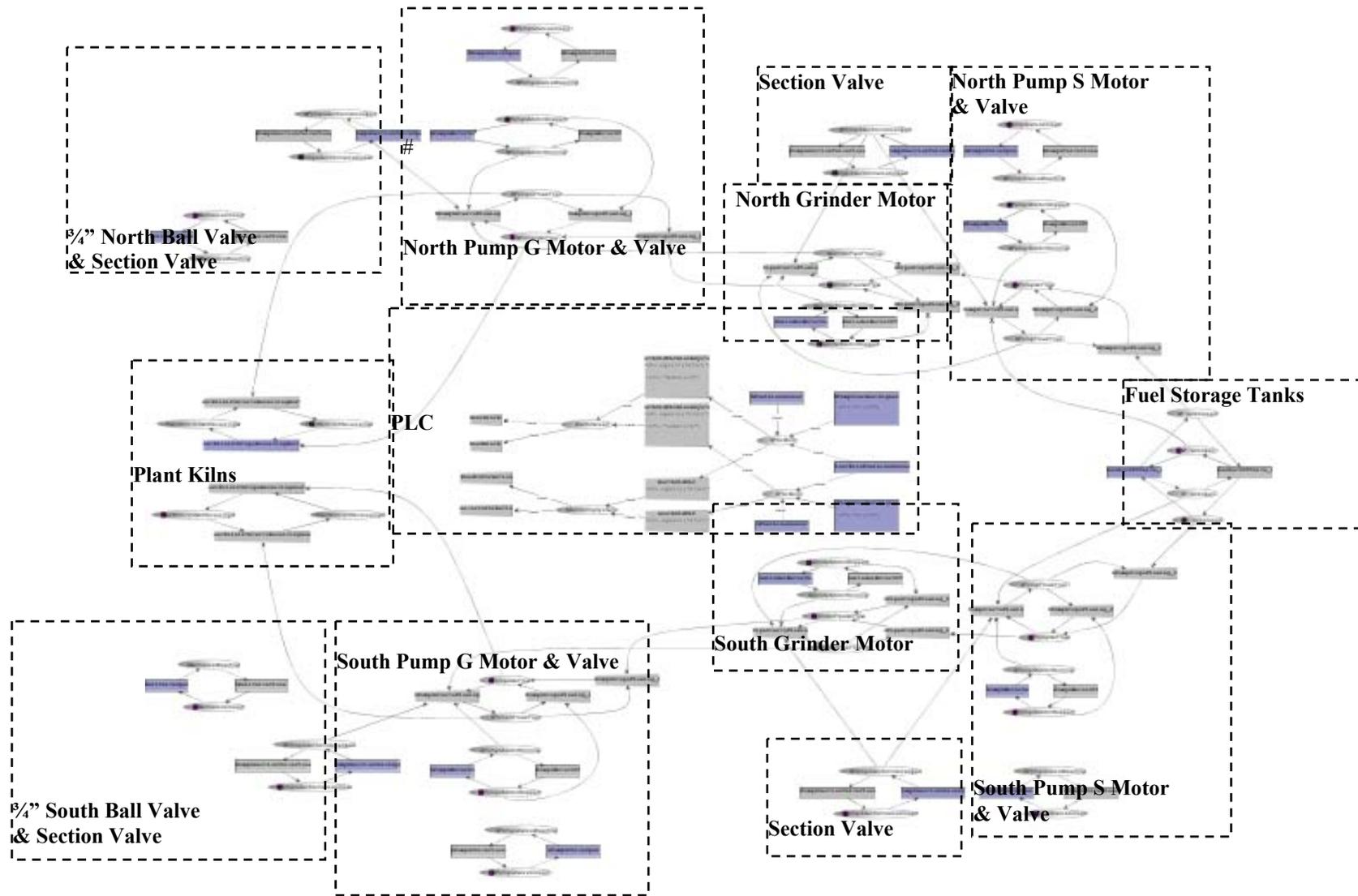


Figure 8-30. Complete System Model

12 Mapping Erroneous Events to System Model

In this section, we show apply the process of relating the ECFA to the system model as well as exploiting information from the Safety Cases to inform the system model.

12.1 Relating ECF and System Modelling

Although the ECFA chart that we have produced depicts only the path of events that led to the accident, reverse engineering the process can produce numerous possible scenarios that could lead to the same accident. This is because there can be more than one way to reach a certain condition or system state which then resulted in a specific cause. Table 8-3 details the relationships between primary events modelled in the ECFA chart, and events in the Petri net systems model. As previously mentioned, the ECFA chart is read from left to right thus the table follows the same approach with 'event 1' in the table being the left most primary event etc. Since we are only considering the primary events, the table does not include any conditions.

Table 8-3. Relating ECFA with System Model

ECFA Number & Name	Event Type	Petri net Equivalent	Petri net Token Representation
1- Seal on North Grinder overheated	System	Not represented in the system model	
2- Worker shuts down North system pumps & starts South system pumps	User	The south tank has been left permanently open	1 x STankOpen, 1 x SPumpGMotorOn, 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 in SGrinderPipeFlowing, 1 x SPumpGMotorRunning,
3- Fuel does not flow to kilns	System	The fuel kilns not represented. However, new fuel flow to kilns is shown as South Pump G Not Flowing	1 x SPumpGNoFlow, 1 x SPumpGAir
4- Supervisor bleeds air from 3/4" ball valves of fuel pipe system	User	3/4" ball valves not represented	
5- Worker bleeds air at south pumps	User	South Pump S and Grinder valves not represented.	1 x STankOpen, 1 x SPumpGMotorOn 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 x SGrinderPipeFlowing 1 x SPumpGMotorRunning., 1 x SPumpGNoFlow 1 x SPumpGAir, 1 x SPumpGValveBleeding 1 x SPumpGHazard
6- Supervisor bleeds air of valve south 330 pump while in operation	User	South Pump G Valve	1 x STankOpen, 1 x SPumpGMotorOn 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 x SGrinderPipeFlowing 1 x SPumpGMotorRunning, 1 x SPumpGNoFlow 1 x SPumpGAir, 1 x SPumpGValveBleeding 1 x SPumpGHazard,
7- Blockage in pipe abruptly removed	System		1 x STankOpen, 1 x SPumpGMotorOn 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 x SGrinderPipeFlowing 1 x SPumpGMotorRunning, 1 x SPumpGFuelsFuel, 1 x SPumpGAir 1 x SPumpGValveBleeding, 1 x SPumpGHazard
8- Fuel starts flowing in the piping	Assumed System Event	SPumpG Starts Flowing Transition	1 x STankOpen, 1 x SPumpGMotorOn 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 x SGrinderPipeFlowing 1 x SPumpGMotorRunning, 1 x SPumpGFuelsFuel, 1 x SPumpGAir 1 x SPumpGValveBleeding, 1 x SPumpGHazard
9- Water-Hammer Effect	System/User	SPumpG Hammer Effect Transition	1 x STankOpen, 1 x SPumpGMotorOn 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 x SGrinderPipeFlowing 1 x SPumpGMotorRunning, 1 x SPumpGFuelsFuel, 1 x SPumpGAir 1 x SPumpGValveClosed
10- Grinder exploded & propelled off its base	System	Grinder Explosion	1 x STankOpen, 1 x SPumpGMotorOn 1 x SPumpGFuelFlows, 1 x SGrinderMotorRunning, 1 x SGrinderPipeFlowing 1 x SPumpGMotorRunning, 1 x SPumpGFuelsFuel, 1 x SPumpGAir 1 x SPumpGValveClosed, 1x GrinderExplosion

12.2 Remodelling the System Exploiting Safety Cases

In this section, we present how the system model can be extended to take into account the hazards highlighted in the Safety Cases. The system model for this case study has been extended to explicitly represent hazardous states and actions identified from the accident report and also from the human 'error' analysis on the task models. The aim is not to model every possible problematic situation that can occur in the system model. It is highly reasonable that not all of the information provided in a Safety Case can be applied to a system model, for example information regarding personnel and training. However, we aim to show how the behaviour of the system can be influenced by hazardous events (such as a water hammer effect) and also lead to hazardous states.

One of our aims for including data from incident and accident investigations in safety-critical interactive systems design is to ensure that the same incident or accident analyzed will not occur again in the re-modelled system. That is, that an accident place within the network will be blocked from containing a token. To this end, we extend the system model presented in Figure 8-30 to include hazardous states identified in the safety-cases in section 8.

Table 8-4 presents several hazards that are useful for representing in the system model. It is important to stress that the hazards shown in Table 8-4 are not explicitly represented in the high-level safety-case shown in previous sections. They do, however, appear at more detailed levels of our analysis for example in the human error analysis on the subtask of the operator task model.

Table 8-4. Summary of Hazardous States Identified from Incident and Accident Investigation

Hazard Name	Brief Description	Type of Dimension
Water Hammer Effect	Caused by rebound waves created in a pipe full of liquid when a valve is closed too quickly.	System
PLC Not Connected	The new PLC was not connected to the F-System.	Interactive
Valves Bled While System in Operation	The valves should not be bled while the system is in operation	Interactive
Motors Running Dry	Occurs when motors are turned on but fuel is not in pipes	Interactive
Fuel Spreading	Occurs when the valves are bled while motors and fuel are running	Interactive
Gathering Air	Air can gather in a pump if the valve was not bled before start-up of the system.	Interactive

Additional modelled hazardous situations, as listed in Table 8-4 are "motors running dry" and "fuel spreading" from opened valves. We have also included the possibility that the pipes contain air. By system run dry, we mean a pump motor is running but fuel is not being received from the previous component. Fuel can spread when a valve is left open, the motor is running and fuel is flowing. Air can gather in a pump if the valve was not bled before start-up of the system. The following Petri Net (Figure 8-31) shows the extended model for north pump-S motor and valve. It can be compared to the previous model (see Figure 8-30 and Figure 8-27), which contains the same components without hazards and where three possible hazards have been highlighted.

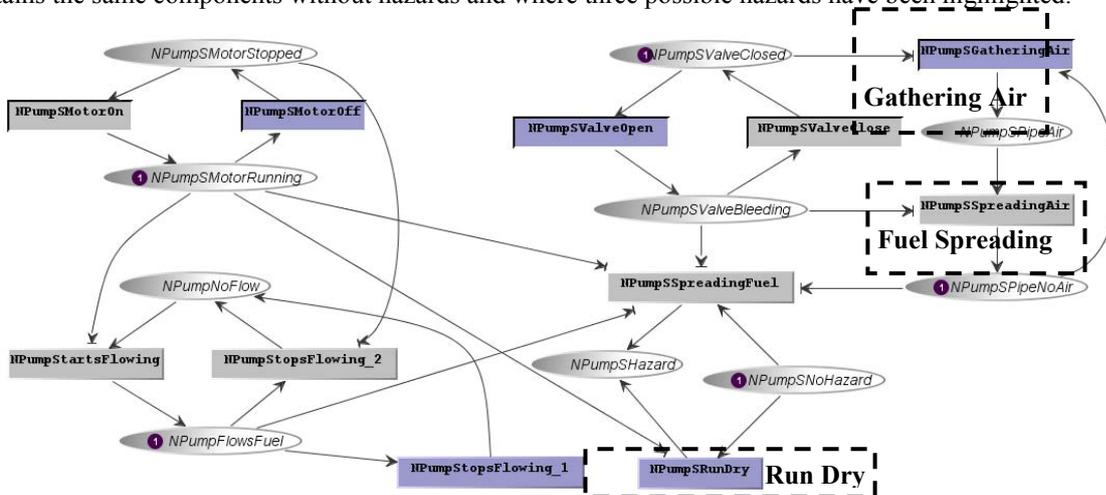


Figure 8-31. North pump S motor & valve including hazards

While incorporating potential hazards into the system model makes the Petri Net more complex it also provide means for simulating the behaviour of the systems and showing which sequence of event lead to the accident. This simulating task is fully supported by PetShop environment through actions on the user interface simulating the real actions on the real system.

Figure 8-32 illustrates the complete system model including hazards. Though the model is not fully legible, it provides an idea of the complexity of including such additional hazardous information.

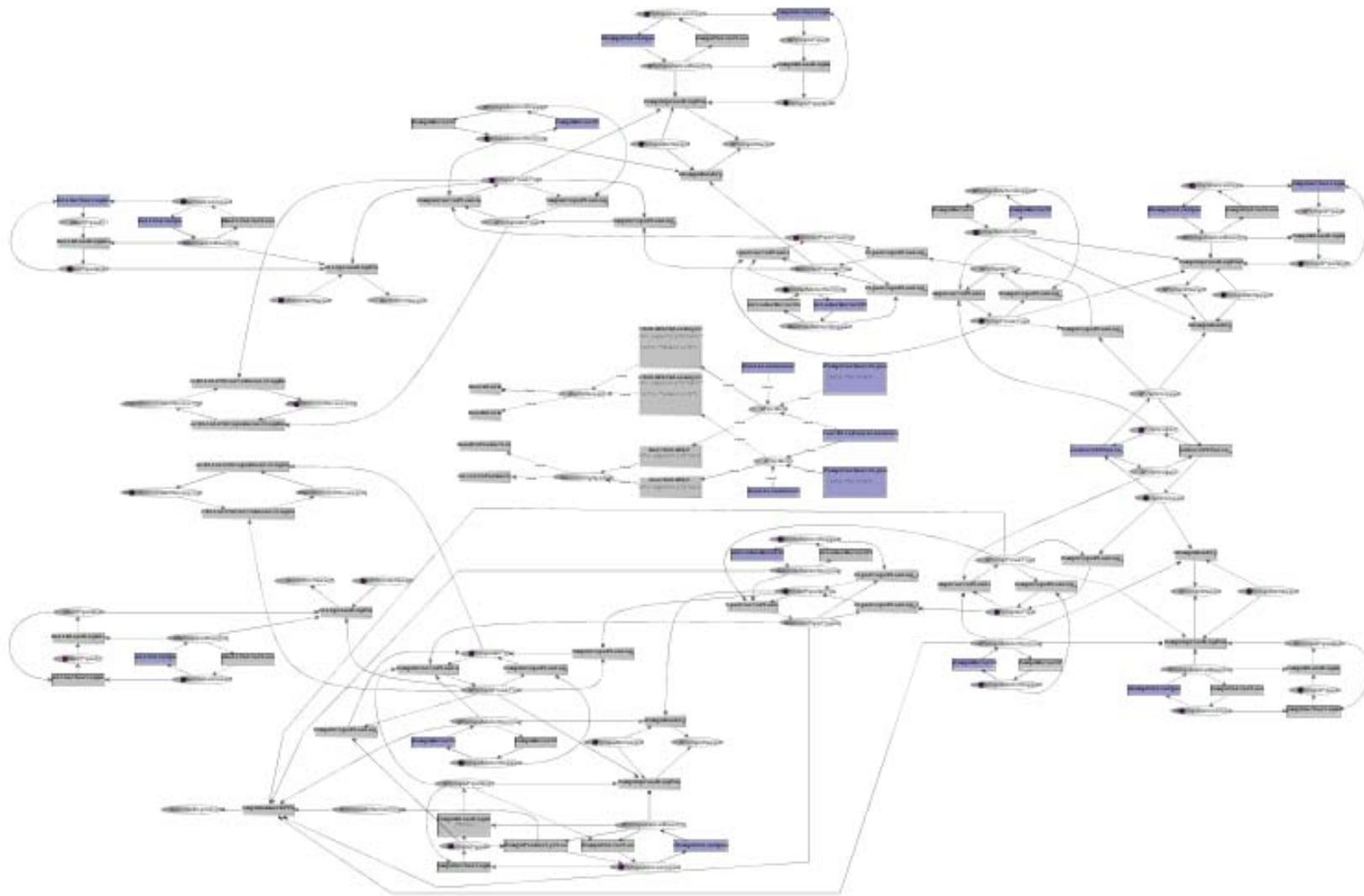


Figure 8-32. Complete System Model Including Hazards

12.3 Adapting System Model to Prevent the Accident

In order to support the construction of reliable interactive systems it is critical to provide ways of learning from experience when designing a new system. This is one of the aims of the approach presented in thesis. We cannot, however, provide automatic means of redesigning the system to prevent any recurrence of an accident or incident. Our approach, in contrast, provides tools to the designers to help them check whether the new model makes the re-occurrence of the accident/incident impossible.

Once the system model has been adapted according to the accident scenarios identified, it is possible to manually run the scenarios on the new system model in order to prove that the system model does not allow the same sequence of events to occur and thus not re-trigger the accident. By applying such development techniques, the validation of the safety-critical system appears earlier in the design process than in more classic techniques. A formal validation occurs through the use of verification techniques and analysis tools. This validation is possible due to the algebraic foundation of the Petri nets. This simulating task is fully supported by PetShop environment through actions on the user interface simulating the real actions on the real system

12.3.1 Marking Graph Analysis

Figure 8-33, is a part of the system model of the south waste fuel delivery system in which the accident occurred. The diagram shows a Petri net representing the south grinder motor and corresponding fuel flow. The initial state, indicated by a (1) in a place, shows there is one token in *SGrinderMotorStopped* and one token in *SGrinderPipeNoFlow*. This means, that transition *SGrinderMotorOn*, a user triggered event, can be fired. If it is fired, the token from *SGrinderMotorStopped* will be removed and a token will be placed in *SGrinderMotorRunning*. When *SGrinderMotorRunning* contains a token (and assuming the south waste fuel tank is open) the *SPipeStartsFlowing* transition is fireable which means the token in *SGrinderPipeNoFlow* is removed and a token is placed in *SGrinderPipeFlowing*.

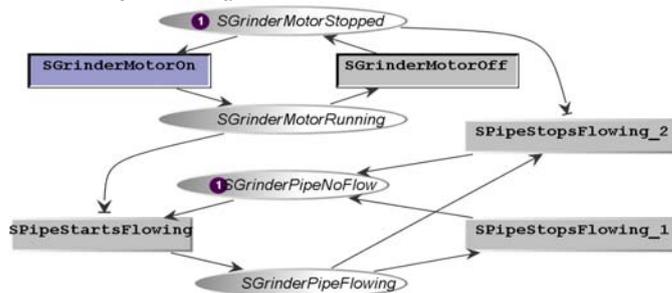


Figure 8-33. Petri net modelling a north pump-s and its motor

The marking tree (represented in textual format below) for the above Petri net indicates that there are 4 reachable states (M0, M1, M2 and M3 respectively), or 4 nodes, and that the net can always return to the initial state, M0.

- M0: {SGrinderMotorStopped, SGrinderPipeNoFlow}
- M1: {SGrinderMotorRunning, SGrinderPipeNoFlow}
- M2: {SGrinderMotorRunning, SGrinderPipeFlowing}
- M3: {SGrinderMotorStopped, SGrinderPipeFlowing}

The marking graph for this Petri net can be thought of in terms of a state transition diagram as follows. For example, the following clauses show that there are two possible transitions available from state M1: *SGrinderMotorOff* and *SPipeStartsFlowing*. If *SGrinderMotorOff* is fired the Petri net will return to state M0, and if *SPipeStartsFlowing* is fired, the Petri net will move to state M2.

- M0: {SGrinderMotorOn: M1}
- M1: {SGrinderMotorOff: M0} {SPipeStartsFlowing: M2}
- M2: {SGrinderMotorOff: M3} {SPipeStopsFlowing_1: M1}
- M3: {SGrinderMotorOn: M2} {SPipeStopsFlowing: M0} {SPipeStopsFlowing_1: M0} {SPipeStopsFlowing_2: M0}

As discussed in the literature review, the marking tree and marking graph of a Petri net can become far more complex when the Petri-net is not 'bounded' (i.e. a place can gather an infinite number of tokens) and a large

number of places and transitions. Since the number of places and transitions of the system model of the waste fuel delivery system of the mining plant is greater than 150, we have restricted our analysis to the south waste fuel delivery system (SWFDS) only which is the part in which the accident occurred. Within the SWFDS, we have also removed several other components including the fuel tank, which we have assumed to be open to the south, the $\frac{3}{4}$ " ball valves, and the plant kilns which should eventually receive fuel if the system was operating correctly. This is again due to the immensity of the marking graph which would contain over 2000 states. The extracted model includes the relevant parts of the system to show the sequence of events leading to the accident. Figure 8-34 shows the extracted Petri-net that we have used to deduce all possible scenarios leading to the accident, i.e. a grinder explosion.

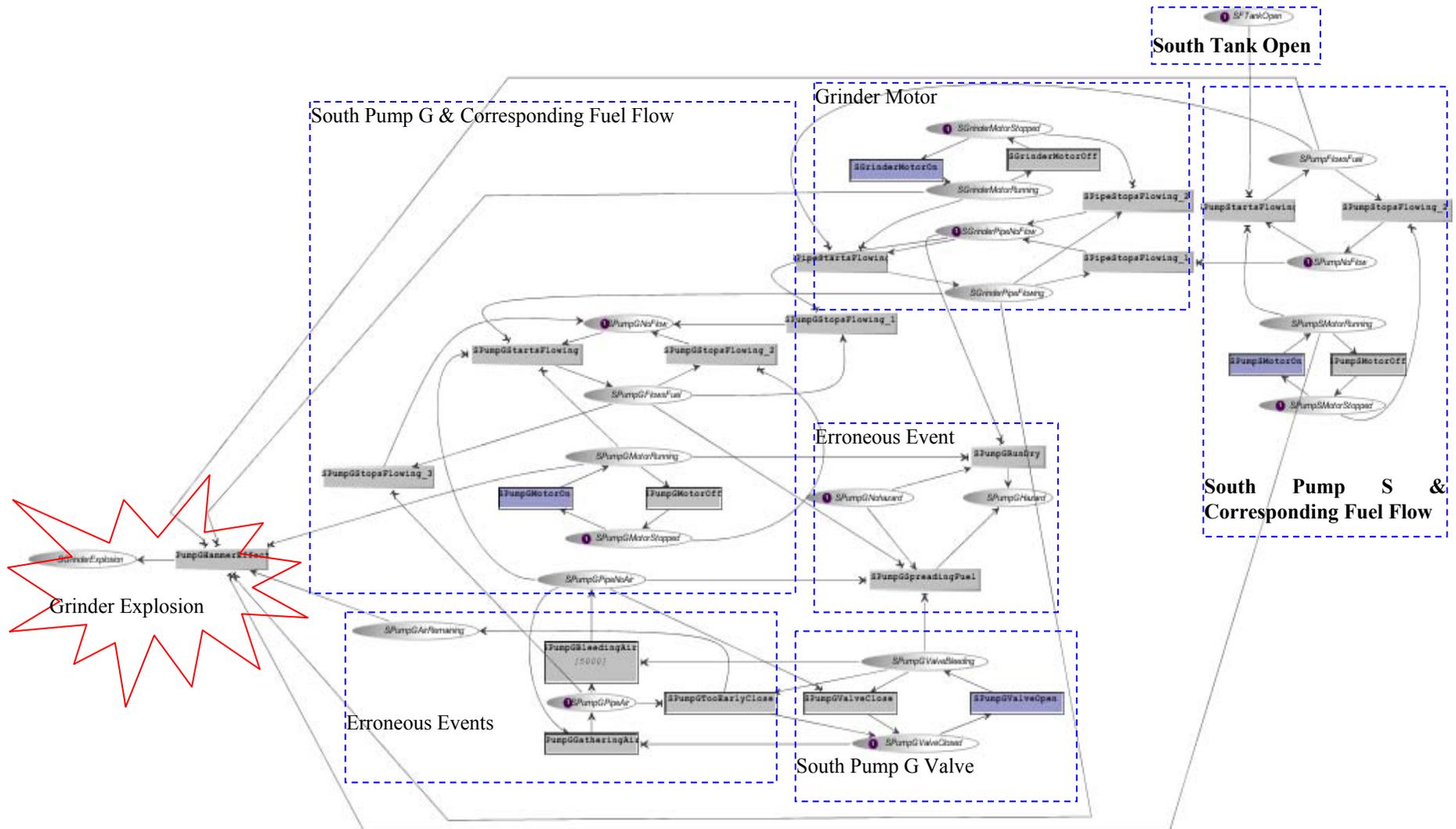


Figure 8-34. Reduced Petri net for SWFD

As mentioned, our analysis focuses on an abstract representation of the sub-system in which the accident occurred. We are primarily interested in the sequence of transitions that lead to a single token being present in the grinder explosion place, which we will call an ‘explosive state’. Thus all states containing 1 token in the grinder explosion place were extracted from the marking tree. We must then identify which transitions are fired in order to reach an explosive state from a non-explosive state. We are not interested in explosive states leading to explosive states since accident will already have occurred. These are the key transitions that will be used to redesign the system model.

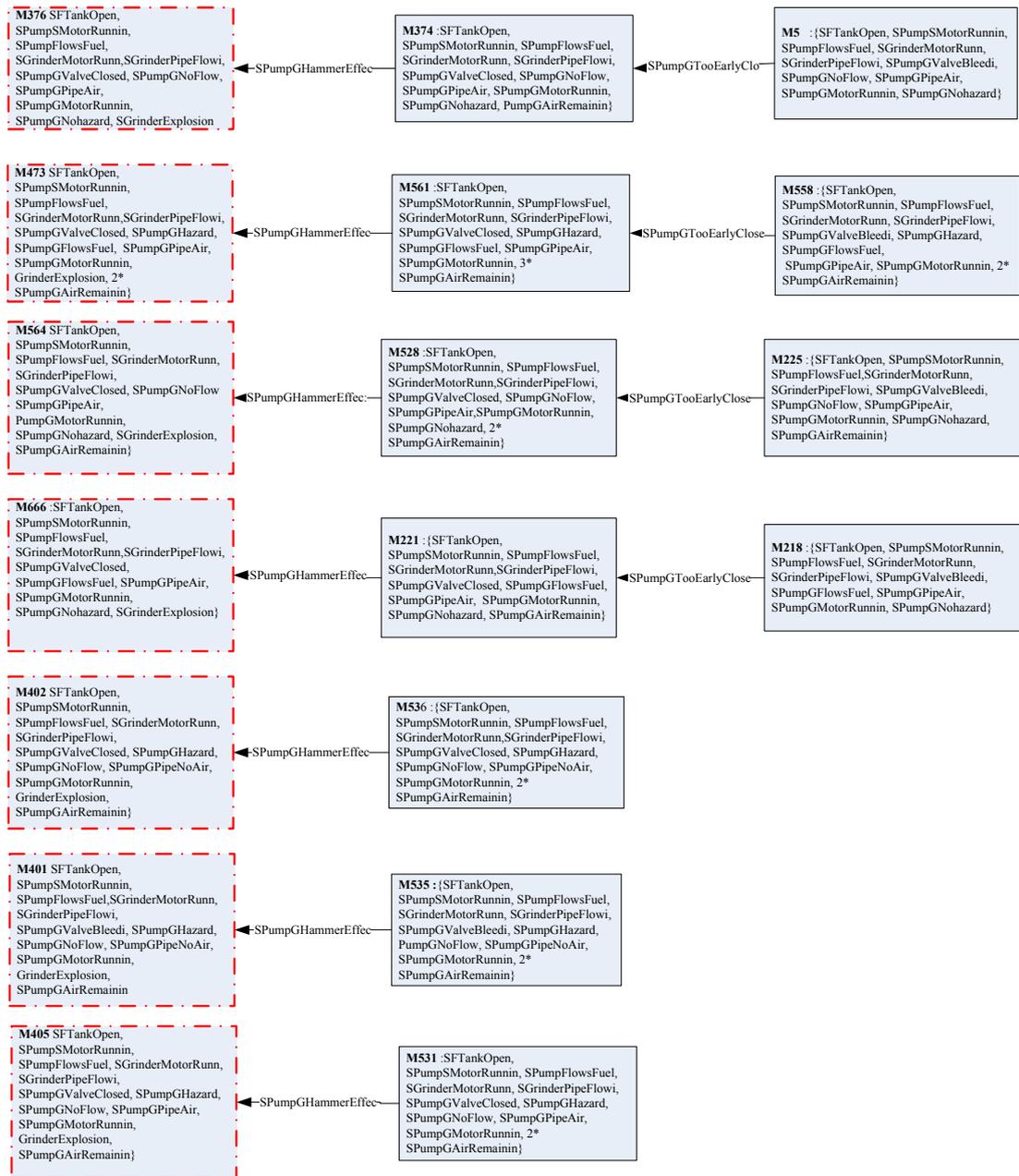


Figure 8-35. Extract of Marking Graph Showing Non-Explosive States Leading to an Explosive State

Following this identification process, the marking graph can be built. It makes explicit the states and transitions leading to explosive states. Figure 8-35 illustrates an excerpt of the complete marking graph representing relevant information. The dashed boxes indicate that the states involve a grinder explosion. The diagram shows that this

particular explosive state is triggered by the same and similar sequences of events as that which led to the fatal accident in the case study.

The top four rows of states are ones which involve the South Pump-G valve being closed too early (as occurred in the accident). This transition leads to state M374, from which transition ‘SPumpGHammerEffect’ is available. This is the water hammer effect that leads to explosive state M376. The bottom three rows do not involve the same valve transition however are provided to show other possible scenarios. The diagram shows that explosive states are triggered by the same and similar sequences of events as that which led to the fatal accident in the case study. The token distribution of state M5 indicates that the south fuel tank is open, the south pump-s motor is running, south pump-s fuel is flowing, south grinder motor is running, south grinder fuel is flowing, south pump-g motor is running, south pump-g fuel is not flowing, south pump-g pipe contains air, south pump-g valve is bleeding and south pump-g does not have a hazard.

The transition, ‘south pump-g too early closed’ from state M5 denotes that the valve being bled at the south pump-g was closed before all of the air was bled.

The identification of scenarios using the marking graph technique to backup the results of the ECFA means that the current system model, including erroneous events extracted from safety cases (in previous work), fully models the actual system and the actual accident that occurred.

The use of these marking graph techniques and the ECFA helps to ensure that the system model captures the behaviour that was observed during an accident. However, we have also identified additional scenarios that could lead to a grinder explosion. Thus, the system model has to be adapted so that the grinder explosion cannot be triggered by a user or by the system in the future. We do not present these modifications that are rather simple but we will show such modelling aspects in the following section which presents our approach to barrier analysis and modelling i.e. how to modify system models by adding barrier models in order to mitigate erroneous behaviour.

13 Barrier Analysis

In addition to the barriers identified in Section 8.2 from the Safety Cases (Alarms, Sensors, Fire, Personnel Training), this section provides a means of identifying further barriers (and sometimes the same barriers) directly from the accident report.

We focus here on the start-up procedure of the fuel line. We will analyse why some of these tasks should have been defined as safety critical, and how our method is of help here. We will investigate how using barriers improves design by helping to define safety critical operator tasks. Lastly we will discuss the integration of these barriers with the system.

In a real world design situation a natural starting point for our analysis would exist, provided by drawings or other documentation of the design. We therefore shortly discuss the design of the fuel line before proceeding with explaining the proposed method. At this point, hazards and barriers are not yet known, hence we start from a purely a functional description. We will discuss a hazard analysis as well, though this is not strictly part of our method.

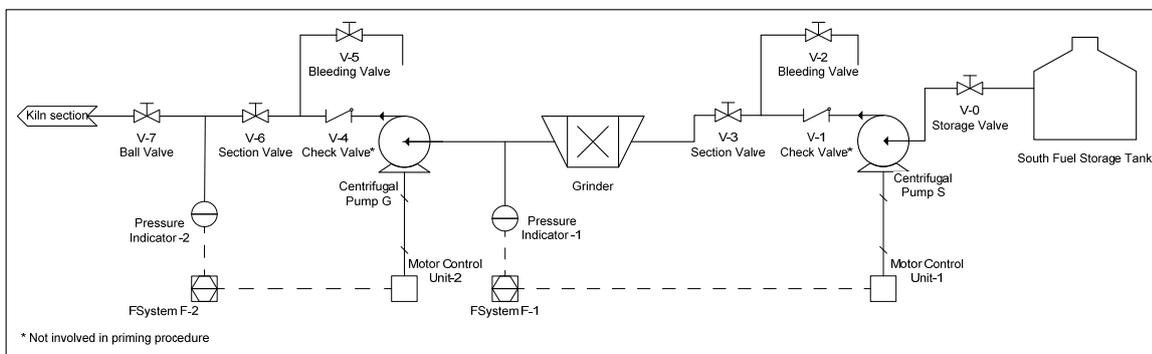


Figure 8-36: Piping and Instrumentation Diagram of fuel line. See text for explanation.

The exact manner in which the design is represented is domain dependent. Here we use a Piping and Instrumentation Diagram (P&ID). The P&ID is only required to understand the design, and may be substituted or complemented by other forms of documentation. In a real world situation, the formal model may have to be created before proceeding.

Figure 8-36 shows the Piping and Instrumentation Diagram (P&ID) describing the Fuel delivery system. V-1 and V-4 represent check valves, V-3 and V-6 are section valves, V-2 and V-5 are for bleeding, and V-7 is a ball valve used to control the start-up procedure. Bleeding is done before system start-up to make the pipes free of air, and to prime the pumps. Without priming, the pumps cannot create suction, and thus do not pump. MCU-1&2 are the Motor Control Units, PI-1&2 are Pressure Indicators, FSystem-1&2 are the implementations of the pump control loops.

The full approach for our barrier analysis and modelling has been presented in Chapter 7. The following sections apply this approach to the case study.

13.1 Step 1: H-B-T analysis

In this example hazard identification is relatively straightforward. A group of experts using a common identification method (e.g. a Hazop) would for instance quickly realise that both too much static pressure and pressure waves which may compromise containment can occur in this system. Though other hazards might be identified as well, we will concentrate on the pressure related hazards here.

Our method however becomes important after hazard identification. Now two steps must be taken. Firstly, potential consequences and risk must be estimated and then barriers to reduce the risk must be designed. Though both steps are supported by the method, we focus on designing the barriers, and in particular at integrating and understanding tasks carried out by humans as part of these barriers. In this case study, typical barriers may include surge arrestors, emergency shutdown or procedural barriers (e.g. limiting pump power during start-up). The method helps to select, specify and verify these.

In Figure 8-37 the results of a Hazard-Barrier-Target analysis using SML are shown. The primary hazard is a fire hazard, as this will cause harm to for instance, operators. Many potential barriers are available that may stop them from receiving such harm. Fire normally is caused by the presence of fuel, air and an ignition source. In this case fire is prevented by an Inherent Barrier (IB1), containment, which keeps the fuel separated from air and ignition sources. In case containment (IB1) fails, and fire occurs, a sprinkler system is present to mitigate the effects of the fire. Figure 8-37A shows a SML representation of this. More complicated and precise models are probably appropriate here, for instance to include failure modes of the barrier (e.g. spraying fuel or just leaking).

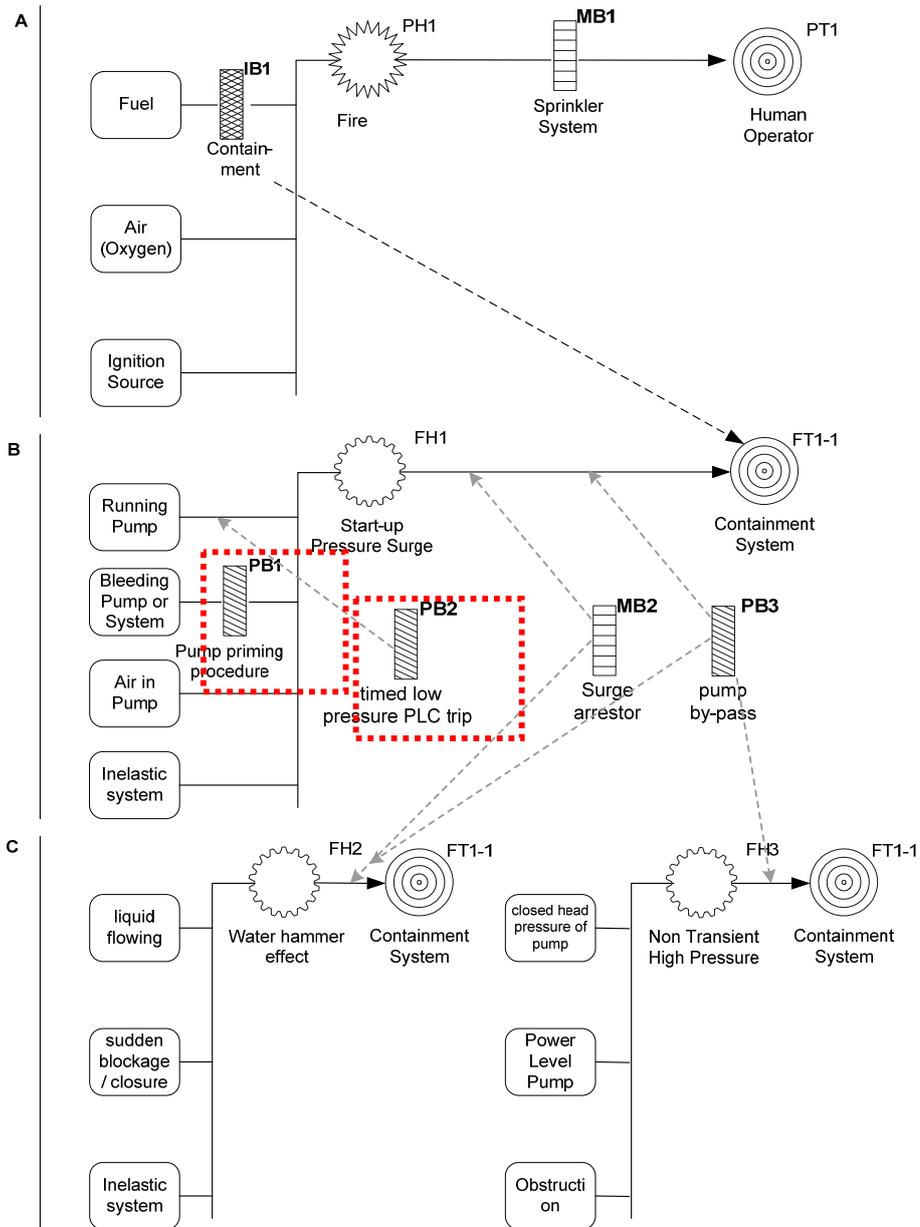


Figure 8-37. SML representation of hazards, barriers and targets discussed in this text (PB1 and PB2 modelled in this chapter)

.Containment thus is an important barrier here. Our further discussion focuses on protecting this barrier against functional hazards. That is, the integrity of pipe work and casing of equipment such as the pumps and the grinder must remain. This may however be compromised by high pressure in the system. Therefore pressure surges must not occur in the system. In other words, these are functional hazards. A pressure surge may for instance occur because of starting a pump in the wrong way. For this to happen, four causal elements must be present; the pump must be running, but it must contain air as is shown in Figure 8-37B. Then the air must be bled from it, which will cause sudden flow. In an inelastic system, this will cause a pressure surge. Hence, bleeding the pump (also called priming; fill it with liquid as it cannot otherwise create suction), and starting the pump must not occur in the wrong order. Two alternative causes of high pressure are possible as well; water hammer due to sudden stop of flow, and static high pressure, see Figure 8-37C.

The system designers may now evaluate alternative barriers that potentially reduce the risk of a pressure surge. SML notation allows system designers to conceptually understand which barrier is best to use. Although PB1, the pump priming procedure, will prevent pressure surges (FH1) altogether, MB2 will protect against most transient pressure waves (FH2), now also including water hammers. PB3 will protect containment against both transient (FH1 and FH2) and non-transient (static) high pressure (FH3), as the dotted arrows indicate.

In deciding, designers also have to take into account factors like probability of failure on demand, economic viability, and ease of design, construction and operation. Since we are interested in a sociotechnical barriers, we continue building an ICO model of PB1 (MB2 and PB3 only involve hardware, no human action).

13.2 Step 2: Barrier modelling using Petri Nets & ICO formalism: Pump Priming Procedure

In this section we discuss the analysis and design of PB1 and PB2. In this step the focus thus is on the barrier, not on the system as a whole. In our view, this is an important improvement over current design practices as the design of barriers and systems often becomes intermingled, causing people to loose track of which elements of the system actually are part of barriers.

13.2.1 ICO modelling of PB1: Pump Priming Procedure

As discussed a pump in this system should not run dry or be started unless it has been sufficiently "primed". Before describing the model of this procedural barrier, we first present informally the priming process exemplified on the fuel delivery system (that is graphically presented in Figure 8-5 and Figure 8-36)

The principle of this procedural barrier is simple. First prime the pump, then switch it on. As the system contains two pumps and some other components, the actual barrier is more complex. A domain expert might design it as follows:

1. Before and during the priming procedure, no motor must be on (i.e. both pump motors and grinder motor)
2. Open the fuel storage tank by opening V0 Storage Valve. (Assumption: gravity will cause fuel to flow through the piping until V3 Section Valve)
3. Open V2 Bleeding Valve to release any air in piping.
4. When all air is removed, close V2 Bleeding Valve
5. Open V3 Section Valve. (Assumption: gravity will cause fuel to continue flowing to the next Section Valve V6).
6. Open V5 Bleeding Valve to release any air in piping
7. When all air is removed, close V5 Bleeding Valve
8. Open V6 Section Valve. (Assumption: gravity will cause fuel to continue flowing to the kiln section).
9. If required, close V3 and V6 Section Valves (when the system is not to be used immediately).

Notice that this barrier reuses some functionality already present in the system. For instance, the pumps do not pump without being primed, therefore priming is part of the functionality with or without this barrier. The PB1 barrier imposes constraints on this existing functionality to ensure safety. This becomes explicit by modelling the barrier this way. At the same time the barrier also defines new functionality, and implements it in other system components. For instance, the first step demands that the pump motors must remain off until the procedure defined by PB1 is completed. This is simple to implement in the barrier model, however in practice this can be more complicated, and detailed discussion of this is beyond the scope of our discussion. It may perhaps be achieved using signs near the switches referring to the procedure.

We have modelled PB1 using the ICO formalism - see Figure 8-38. The diagram has been segregated into several sections for explanatory purposes. The model of this barrier is a combination of hardware and human actions. The hardware concerns the three motors, (modelled in part 1 of Figure 8-38) the Pump-G motor, Grinder motor and Pump-S motor. It also concerns the valves. However these are modelled together with their interaction with the required operator's actions in the remainder of the figure.

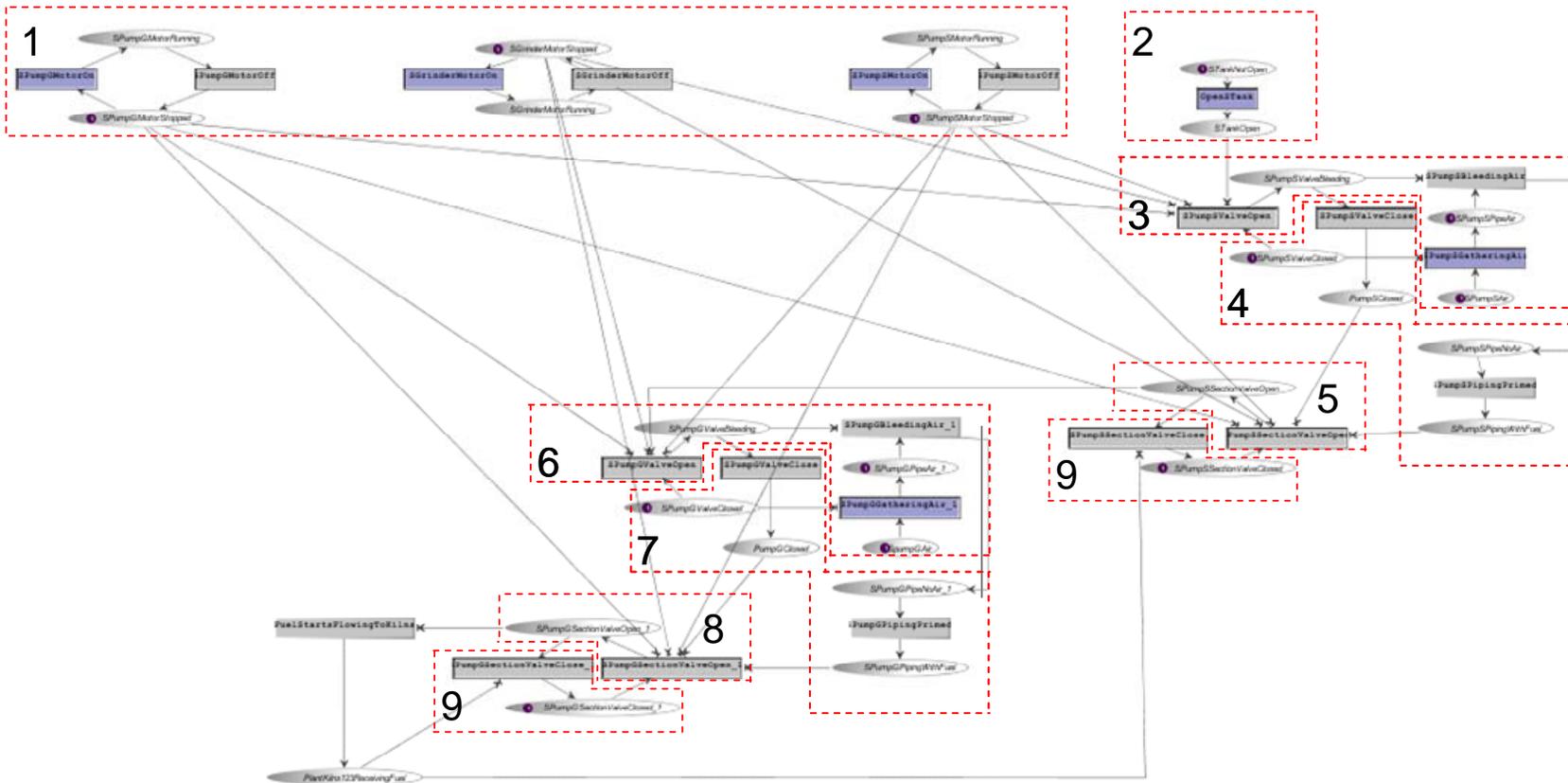


Figure 8-38. PB1 Pipe Priming Procedure Barrier

The barrier is mainly made up of arcs connecting transitions and places rather than the transitions and places themselves. It can be decomposed into three main parts. The first part concerns preventing the three motors (part 1 of Figure 8-38) from running before the procedure is completed. This is modelled by means of test arcs which impose a pre-condition on operating the bleeding valves (V2 and V5) or the section valves (V3 and V6) preventing them from being opened during the priming procedure if the motors are running.

The second part of PB1 is the obligation of order of events for the operator’s interaction with the system. For instance part 5 shows the opening of the 1st section valve which cannot be opened before air has been bled from the first section of piping (part 3).

The third part is also an obligation of order. It prevents the operator from closing the section valves (as shown in part 9) before fuel arrives at the plant kilns (a token is set into place *PlantKiln123ReceivingFuel*). Once fuel arrives at the plant kilns, the task is complete. The closing of the section valves (step 9 in the procedure) is optional and depends on whether the fuel delivery system will be started immediately in which case the section valves are left open, or later, in which case they can both be closed.

13.2.2 ICO modelling of PB2 Auto-Shutdown of Pumps

The second Primary Barrier (PB2) has also been modelled using Petri nets and the ICO formalism. Figure 8-39 illustrates the Petri net representing this barrier. It is interesting to that it is very different from the priming procedure barrier previously described.

As previously mentioned this is also one of the barriers designed and present in the existing mining plant. One of the commands that the F-System was programmed to send to the PLC, was to de-energize all pumps in the fuel delivery system if a pressure of approximately 60 pounds per square inch (PSI) was not sensed at the line in the kiln floor area within 3 minutes of system start-up. The 3-minute set point was based on a normal delay of 3 minutes for pressure to reach approximately 60PSI at the kiln area from the time the pumps were started. The accident occurred 10 minutes after the pumps started. When no pressure was detected in the line at the kiln area, the F-System functioned properly in signalling the PLC to shut down the pumps 7 minutes before the accident, but the PLC did not respond. It failed the day of the accident because three months prior to the accident, an older PLC system was replaced by a new PLC, and the F-System connections to the older PLC system were never changed over to the new PLC.

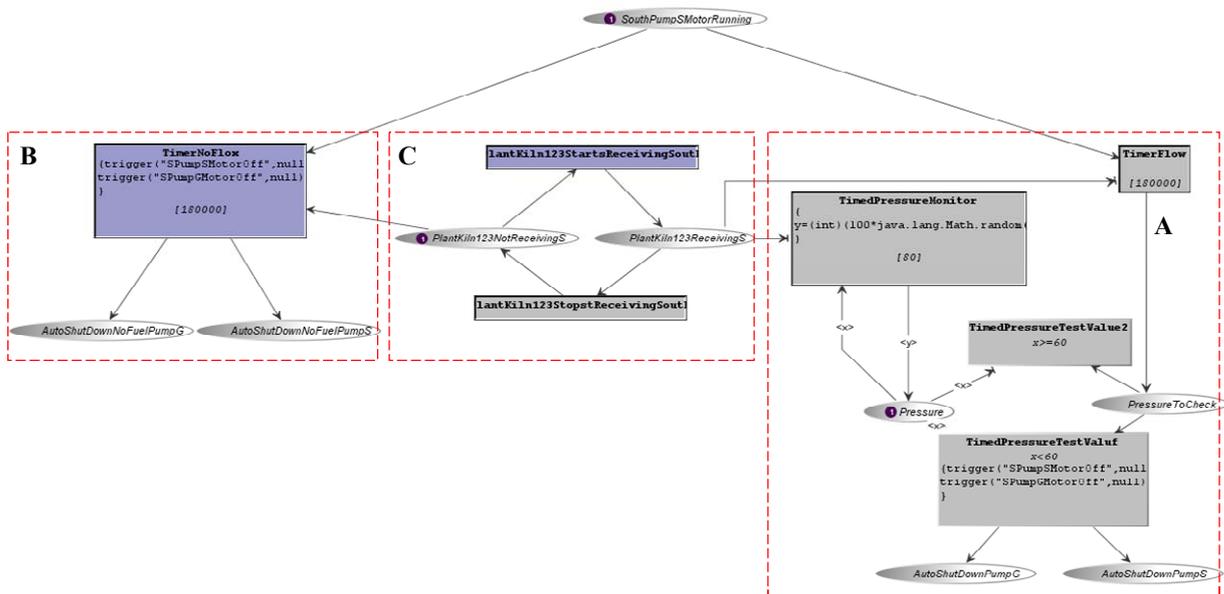


Figure 8-39. PB2: Auto shut down of pumps after 3 minutes barrier

Figure 8-39, diagram of PB2, has been separated into three sections for explanatory purposes. In order to model this barrier, we have assumed that the F-System system starts counting to 3 minutes from the moment the South Pump-S Motor is started (the first motor in line from the fuel tank according to the order of fuel flow). The place at the top of the diagram containing a token, SouthPumpSMotorRunning, represents the behaviour that the motor is running. With respect to the barrier, there are main two possibilities at the end of

the 3 minutes; either no fuel arrived at the plant kilns (modelled in parts A and B) or fuel arrived at the plant kilns (modelling in parts C and B). Part B of the diagram represents the plant kilns. Fuel flows to the plant kilns via the north or the south waste fuel system. The component in this diagram represents the kilns receiving or not receiving fuel, from the south delivery system only. The initial marking (represented by setting a token in the place PlantKiln123NotReceivingS) of the kilns in this model shows that the kilns are not receiving fuel from the south delivery system.

In the first scenario, where no fuel arrives at the kilns by 3 minutes, the TimerNoFlow transition which has a timer of 180,000ms will fire, remove the token from SouthPumpSMotorRunning and from PlantKiln123NotReceivingS and set a token in AutoShutDownNoFuelPumpG and AutoShutDownNoFuelPumpS.

In the second scenario, where fuel does arrive at the kilns by 3 minutes, the auto shut-down of the pumps depends on the pressure of the fuel in the pipes. If the pressure is less inferior to 60PSI, the pumps must be shut down. To simulate the scenario in which fuel arrives, the PlantKilns123StartReceivingSouth transition can be fired which will remove the token from PlantKiln123NotReceivingS and set a token in PlantKiln132ReceivingS. The moment fuel begins to arrive at the kilns, the transition TimedMonitorPressure sets a token in the Pressure place and gives it random values (representing the PSI pressure) every 80ms (in this model).

In the top right hand corner of part C of the diagram, we have another timer (the same as in part A) lasting for 180,000ms. After 3 minutes, this transition will fire removing the token from SouthPumpSMotorRunning (Note, the token containing a random value is not taken from PlantKiln132ReceivingS because fuel continues to arrive. This is modelled using a 'test arc'). The timer transition sets a token in the PressureToCheck place. From this state, two further transitions will become fireable, depending on the random value that the token in the PressureToCheck place has been set to. If the value of the token is superior to 60, the upper transition, TimePressureTestValue2 will fire automatically, and the system will simply continue to function. If on the other hand, the random value of the token is inferior to 60, the lower transition TimePressureTestValue will fire and set a token in AutoShutDownPumpG and AutoShutDownPumpS.

It must be noted however, that the barrier model of PB2 could be considered as incomplete because it currently does not shutdown the pumps. The complete model of the barrier includes events and arcs connected to the pump motors in order to represent the connections between the models and thus automatically turn pumps G and S off.

Now the barrier models have been established they can be analysed further. For instance human factors methods can be used to understand whether humans can achieve the tasks that the barriers specify.

13.3 Step 3: Connecting technical and human barriers in the system model

Integrating the barrier in the system in the real world obviously involves much more than just integrating the ICO models. However, we will continue our discussion on the integration of barriers with the system model. As discussed, the initial state required by PB1 might be implemented by adding a sign to a power switch on pump, which refers to the procedure specified by PB1. Such implementation issues are beyond the scope of this paper however, and require additional methods. Here we only have space to briefly discuss what happens to the ICO models.

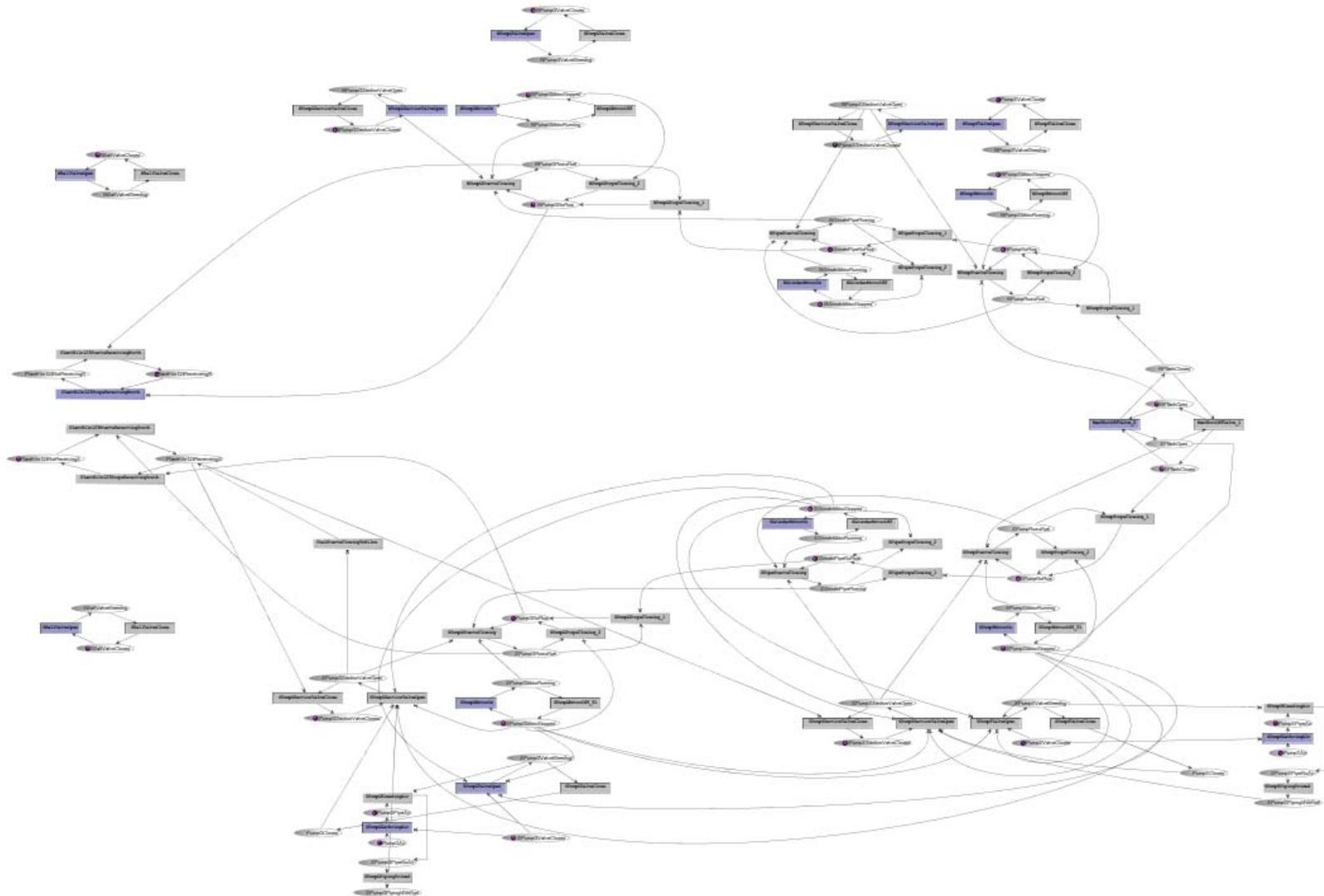


Figure 8-40. ICO System Model + PB1

The initial ICO system model as presented in Figure 8-30 significantly changes because of the connection of PB1. The system model, including PB1 is presented in Figure 8-40 while the system model including PB2 is presented in Figure 8-41. Unfortunately, they are illegible, but show that the complexity of the model has significantly increased, mainly because of the number of added arcs. Also several places and transitions have been added for instance to represent the existence of air in the pipes which was previously not represented in the original system model as this has nothing to do with the initial functional specification of the system.

Two further issues must be noted. Firstly the addition of the barrier to the system model must not change the behaviour of the system, except of course for excluding the hazardous state it is designed for. That is to say, an action or task that was previously available on the system side or via interaction with the user must not be changed and must be still available.

Secondly, the actual modelling of the barrier inevitably involves direct integration with the system model because the barrier not only takes existing components, but may also rely on adding extra arcs to existing components. Hence it is not possible to model the barrier without taking notion of the system model; barriers exploit a subset of the system model, sometimes with extra transitions and places. Thus step 3 of the approach is the addition of the complete barrier, including any additional components that were not present in the existing system model.

Using the ICO model representing the system behaviour, it becomes possible to verify that the system still works, to prove that the hazardous state is no longer reachable and to analyse in which way the newly integrated barrier may fail.

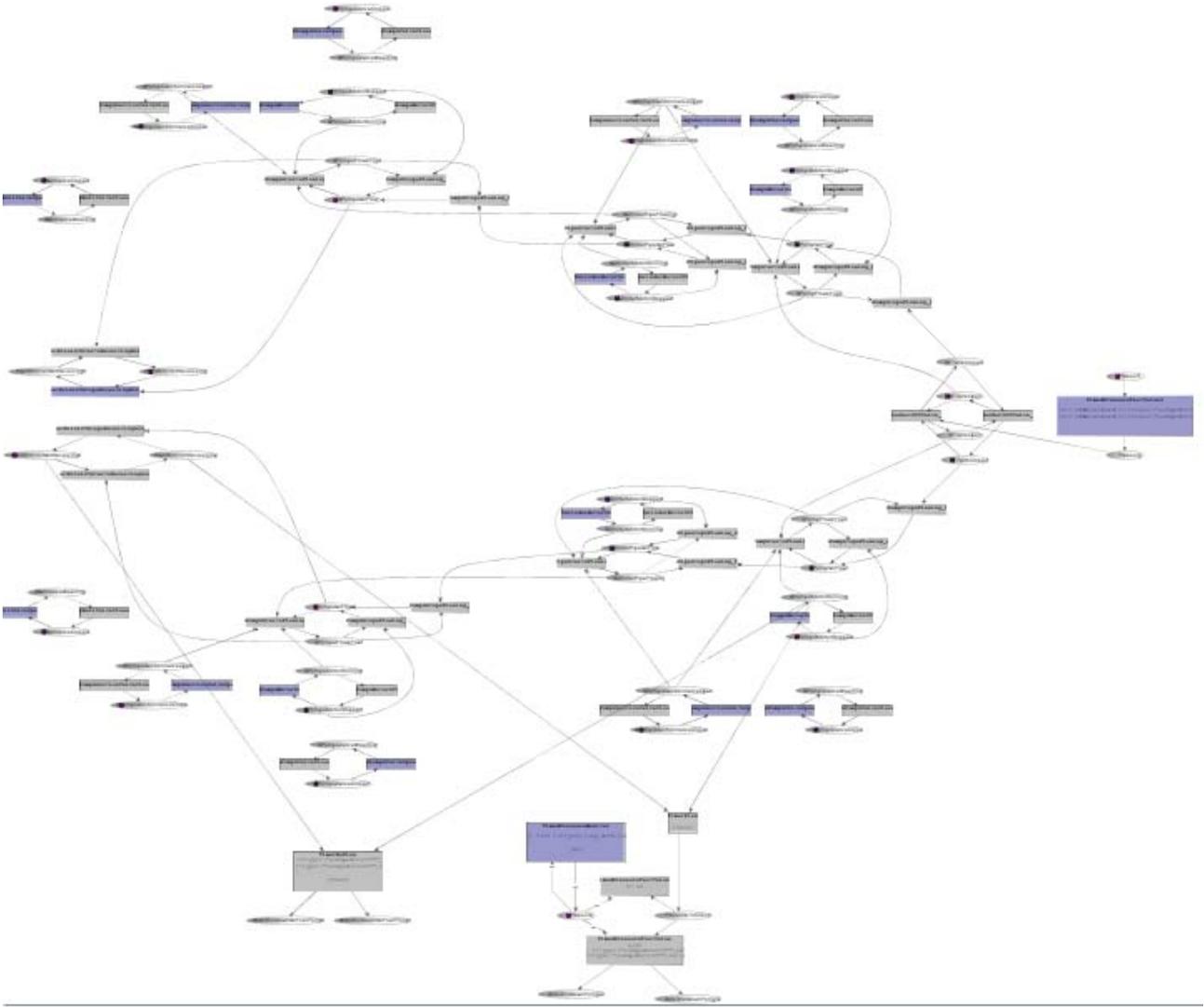


Figure 8-41. ICO System Model + PB2

14 Conclusion

The increasing complexity of many computer-controlled application processes is placing increasing demands on the investigation of adverse events. At the same time, there is a growing realisation that accident investigators must consider a wider range of contributory and contextual factors that help to shape human behaviour in the causes of safety-related incidents.

This chapter has presented the use of a range of techniques that are required and have been developed to address these issues. For example, task-modelling techniques have been extended from human computer interaction and systems design to analyse the causes and consequences of operator 'error'. Similarly, Barrier Analysis has been widely used to identify the way in which defences either protected or failed to protect a target system from potential hazards. Many barriers fail from common causes, including misconceptions that can be traced back to early stages in the development of a safety-critical application. For instance, unwarranted assumptions can be made about the impact of training on operator behaviour in emergency situations. Task models often uncover deep-rooted problems, for instance, in workload allocation across many different aspects of an interactive control system. It can be difficult to use Barrier and Task Analysis to trace these common causes that lie behind the failure of many different defences. The system modelling part also presented how the information gathered by the other techniques can be integrated altogether and how the analysis of accidents and incidents can be exploited for system improvement.

This chapter has therefore, shown how an analysis of Safety Case arguments can be used to support the application of Barrier and Task Analysis during the investigation of a command and control failure. Many countries require that Safety Cases demonstrate a system is 'acceptably safe' before they grant regulatory approval. These documents and the associated analytical techniques, therefore, provide a rich source of information about why command and control failures occurred.

Probably the most important problem with research in accident investigation, and overall safety research, is that it is very difficult to validate the conclusions and suggestions made. This becomes more evident when considering that safety case practitioners use the phrase "acceptably safe", rather than claiming a system is completely safe. We have suggested a way of dealing with diverse issues, ranging from human activity to complex command and control systems, and most importantly linking them as elements of a safety argument, and reasoning about how they interact towards causing an accident.

This chapter has provided the first steps towards the use of accident investigations to inform system modelling and verification. We deliberately chose to integrate approaches that are pitched at different levels of abstraction. The accident analysis focused on a novel application of the GSN technique for the development of Safety Cases. This decision was deliberate because GSN provides means of representing the implicit assumptions that are often difficult to capture in formal models of interactive systems. For instance, previous diagrams have represented assumptions about maintenance intervals and the adequacy of training for particular system operators.

The decision to use GSN in the preliminary stages of this work creates numerous problems. It would have been far easier to use an event based reconstruction technique such as Events and Causal Factors analysis, promoted by the US Dept of Energy. There would then be a relatively automatic translation from this analysis into the places and transitions of the systems models. We chose not to do this because we believe that the higher-level analysis provided by GSN complements the more detailed systems modelling promoted in previous DSVIS workshops. However, there is no automatic translation from the insights provided by the application of GSN to the properties that we might want to verify using the ICO models. This is the subject of current research where, for example, task analysis has been used as a stepping-stone towards the identification of theorems to assert against particular system models (Leveson 2004). We can take high-level safety goals from the GSN, such as the need to ensure effective maintenance. These can be broken into a large number of different sub-tasks. We can then analyze the systems support that is available for each of these subtasks. However, it is unlikely that we will ever be able to automate the process of deriving putative theorems from the often highly contextual and situated observations of accident investigations.

The second stage of our approach relies upon system modelling using the underlying Petri Net formalism embedded within the ICO approach. This offers numerous benefits. Firstly, the process of translating high-level observations from the safety case analysis helps to ensure that the meta-level arguments about system failures can be instantiated in terms of specific subsystems. In other words, the construction of the systems model helps to validate technical assumptions in the accident report. If we cannot develop a coherent system model then there may well be important details that have been omitted from the account of an accident. Secondly, the

development of the system model provides an abstract representation that is amenable both to animation and various forms of proof. This enables us to go beyond the specific scenarios of an individual accident. We can begin to experiment with new designs to ensure that they would not suffer from the same problems as previous designs. In other words we can conduct various forms of reachability analysis to determine whether a new or revised system might reach the same unsafe state. This style of analysis offers other benefits because we can begin to look for other transitions that were not observed in the particular accident being analyzed but that might lead to the same undesired outcomes in future accidents.

Using an accident report to inform the design of this safety-critical interactive system in a systematic way allows us to achieve two things, 1) obtain and 2) deduce important information that could be embedded into future waste fuel delivery systems of mining plants.

Such information obtained includes:

- Add additional fire sensors in the waste fuel containment area to detect heat from fire and activate the fire suppression system more rapidly. Ensure the Programmable Logic Controller (PLC) connectors are properly installed.
- Implement procedures requiring all equipment operators and their supervisors to review manufacturers' instructions and recommendations to ensure machinery and equipment is operated according to manufacturer's guidelines.
- Install audible and/or visual alarm systems in the waste fuel containment area.
- Ensure equipment is installed according to the manufacturer's requirements. Develop procedures and schedules and monitor them to ensure that the required maintenance is performed

Information deduced after implementing and analysing the results of various safety analysis techniques resulted in the following findings. The system should be designed such that:

- A waste fuel delivery system cannot be started without being primed first.
- Motors cannot be turned on without fuel available in the pipes.
- Air is bled from the pipes before a fuel delivery system is turned on.
- Air cannot be bled while a waste fuel delivery system is on.
- An emergency shutdown button should available to operators.

In the aftermath of an accident there may be limited finance. One might argue that using a minimal budget on accident analysis and systems design may not be the most appropriate way to use the available funds. However, we argue that this is precisely where investment should be made, in order to help prevent other similar accidents from occurring by learning from previous incidents or accidents.

Chapter 9

Multi Purpose Interactive Application

“The only reason for time is so that everything doesn't happen at once.” (Albert Einstein, 1879 - 1955)

Chapter Summary

In this second case study, an interactive cockpit application, called the Multi Purpose Interactive Application that is compliant with the ARINC 661 specification standard, we focus principally on the interaction hazard analysis part of the framework and software barrier modelling. In order to do so, we first present the MPIA, then the ICO system model for the part of the application we focus on following by the analysis and barrier modelling.

1 Introduction

In the aviation industry, cockpits are becoming increasingly interactive with the introduction of the keyboard cursor control unit (KCCU) combined with graphical display units (DU) to be embedded in the Airbus A380 for commercial flights in 2006 and later in the Boeing 787 aircraft (see Figure 9-1).

Flight Control Unit



Figure 9-1. Interactive Cockpits

There exists extensive literature on types of human computer interaction failures that can occur while a user is operating an interactive system. Examples include mode confusions which are a kind of automation surprise (Rushby et al. 1999). Evidently, the consequence of such “errors” in a safety-critical context can potentially be devastating. The roots of such “errors” are often the result of poor design. Typically, there is a mismatch between the designer’s conception of the system and the user’s interpretation of the system. From an organizational perspective, such “errors” could also be a consequence of poor training, management etc.

Ideally, the possible existence of these “errors” should be considered throughout the design process, as early as during requirements gathering to mitigate them and minimize development costs. Techniques for analyzing software safety such as Preliminary Hazard Analysis (PHA) including Fault Tree Analysis (FTA) and System and Subsystem Hazard Analysis (SHA and SSHA) including deviation analysis such as Failure Mode Effects

and Criticality Analysis (FMECA), mode confusions analysis and state machine hazard analysis are targeted for such early analysis. (Leveson et al. 1997b).

The identification of potentially dangerous software aspects can be used to inform changes to the design, and training procedures etc. With respect to the design, such modifications could be considered as software barriers.

While barriers have been used quite intensively in the field of hardware and socio-technical systems (Hollnagel 1999b) their use in the field of software systems remains very limited as for instance in (Leveson 1991). The intrinsic nature of interactive software systems that involve users in their daily operation require the same level of reliability but, surprisingly, a barrier approach has not been applied to this area so far. This might be related to the fact that interactive applications have been kept away from command and control safety critical areas or to the fact that redesigning the system has been considered as a more adequate alternative than integrating barriers to the extant system.

Interactive applications embedded in cockpits, is the current trend of evolution promoted by several aircraft manufacturers both in the field of civil and military systems (Faerber R et al. 2000;Marrenbach J and Kraiss K-F 2000). Classic cockpits generally have an entry system and associated visualisation. The display is on a screen and the interaction is done external to the visualisation (buttons, knobs). Interactive cockpits integrate the entries and visualisation in the same way as standard software (mouse, keyboard and graphical display).

With respect to technology currently deployed this evolution might be seen as a small step forward. Reality is very different. Embedding interactive applications in civil and military cockpits is expected to provide significant benefits to the pilots by providing them with easier to use and more efficient applications increasing the communication bandwidth between pilots and systems. However, this technological enhancement comes along with several problems that have to be taken into account with appropriate precautions. Questions such as: what kind of interactive components should be used in a cockpit? How to design such embedded interactive applications? What is the impact of such interaction techniques on the reliability of the application? What is the impact on the certification process?

Such questions have been raised since the mid 90s and there is already an ongoing standardization process. The ARINC specification 661 (ARINC 661 2002) (ARINC 661-2 2005) aims at providing a common ground for building interactive applications in the field of aeronautical industry. This standard will be presented in section 4. However, this standard only deals with part of the issues raised. In previous work, we have proposed the use ICOs to be used as a complement to ARINC 661 for the specification, design, implementation and validation of interactive application.

We have shown in the previous chapter, on the fatal mining accident, that the *expressive power* of the Interactive Cooperative Objects (ICO) formalism is sufficient for describing interactive applications, in addition to our previous work (Navarre et al. 2004) and (Ould et al. 2004) illustrating the expressive capabilities of ICOs on interactive cockpits or ground segments for satellite control rooms.

The literature review also discussed how the ICO formalism has been adapted with three innovative techniques, to specifically deal with such large and real-life safety-critical interactive systems. The solutions include models restructuring and visualization techniques (labels hiding, virtual places and mini-view).

We have seen in this thesis that in the field of safety-critical interactive systems, the use of a formal specification technique is extremely valuable because it provides non-ambiguous, complete and concise ways of describing the behaviour of the systems and that the advantages of using such formalisms are widened if they are provided with formal analysis techniques that allow proving properties about the models, thus giving an early verification to the designer before the application is actually implemented. Usually, one of the points put forward while designing such formal description technique is to provide a notation powerful enough for describing/specifying the systems under consideration.

1.1 Contributions on this case study

The design of a usable, reliable and error-tolerant interactive safety-critical system is a goal that is hard to achieve but can be more closely attainable by taking into account information from previous usages of the system. One such usually available and particularly pertinent source is the outcome of an incident or accident investigation. Designs of any nature can be improved by taking into account previous experiences, both positive and negative. However, it is not always the case that an accident report will be available, especially when new technology is involved. With a more pro-active perspective, we aim to analyse a safety critical interactive

application, similar to those that will be embedded in the new interactive cockpits (like the Airbus 380 cockpit or Boeing 787) that are compliant with the ARINC 661 specification, for which we have no accident data. We aim to identify flaws in the application and suggest software barriers that could minimise potential erroneous user interaction with the system.

With the objective of increasing safety, in safety-critical interactive systems, previous research in the field aimed at trying to eliminate the error completely by identifying its source. It has now been widely accepted however that human errors are inevitable due to the unpredictable behaviour of humans and we must instead try to manage errors. The perspective of blame has also changed from isolating an individual operator to having a wider outlook on the organisation as a whole. However, the broader the perspective, the more information has to be gathered and thus making it more complex not only to organise it but also to reason about it.

In chapter 8, we focused on task models and error analysis of sub-tasks to identify interaction problems and issues identified from the accident report. We argue that although it is important for human “error” analysis and consideration for human factors to be included throughout and in early stages of the development process, further problems can occur during the interaction with the system.

In addition to the skill-based Human Error Reference Table (HERT) defined and exemplified in previous chapters, we apply a typical heuristic evaluation to the UI and a Hazard and Operability Studies (HAZOP) analysis. The identified interaction hazards are used to propose socio-technical barriers to mitigate these hazards. The proposed barriers, which may take the form of improved training, software implementation modifications, or UI design improvements for example, are also categorised according to Leveson’s (Leveson 1991) and Hollnagel’s (Hollnagel 1999b) barrier classifications. By doing so, the barriers can be filtered and represented in their relevant design model. For example, barriers relating to the behaviour of the software can be represented in the system model whereas barriers relating to improved training could modify the operator’s task and thus be modelled in the task model. Barriers relating to warning signs, personal protective equipment or fire extinguishers for example should be represented in a dedicated model.

In this chapter, we focus on barriers related to the behaviour of the system. The relevant software barriers are extracted from the identified list and are modelled using the Interactive Cooperative Objects (ICO) (Navarre et al. 2003), a formalism dedicated to the description of interactive systems based on a dialect of Petri nets.

The approach has been applied on an interactive cockpit application called Multi Purpose Interaction Application (MPIA) whose formal description has been presented in (Navarre et al. 2004). The following section briefly describes the case study. Section 5 briefly presents the system model of the MPIA. Section 6 details the interaction hazard analyses and results followed by section 7 which presents the proposed barriers. Finally, we present the modelling of the barriers and their integration with the previously modelled system model of the interactive cockpit application in section 9.

In this chapter, we illustrate on an interactive cockpit application, compliant with the ARINC 661 *Cockpit Display System Interfaces to User Systems* standard (ARINC 661-2 2005), how we can further improve the specification of a safety-critical interactive system by anticipating potential interaction hazards and suggesting and modelling software barriers to inform the system model.

2 The Case Study and the Generic Integrated Modelling Framework

In this section we highlight which parts of the generic integrated modelling framework have not been applied to the case study in this chapter. As mentioned in previous chapters, we do not tackle requirements modelling as we believe this is a research area alone. However, requirements engineering is part of the framework since this is a fundamental phase of systems development. The task modelling phase has not been applied to this case study since we lack information concerning the tasks carried out by pilots, though this would be fairly simple to perform with sufficient data.

The interaction hazard analysis part of the system modelling phase is the main part to be illustrated on this case study since we require a software based system to show this.

Furthermore, we do not apply any quantitative evaluation though we do perform a usability analysis as part of our interaction hazard analysis.

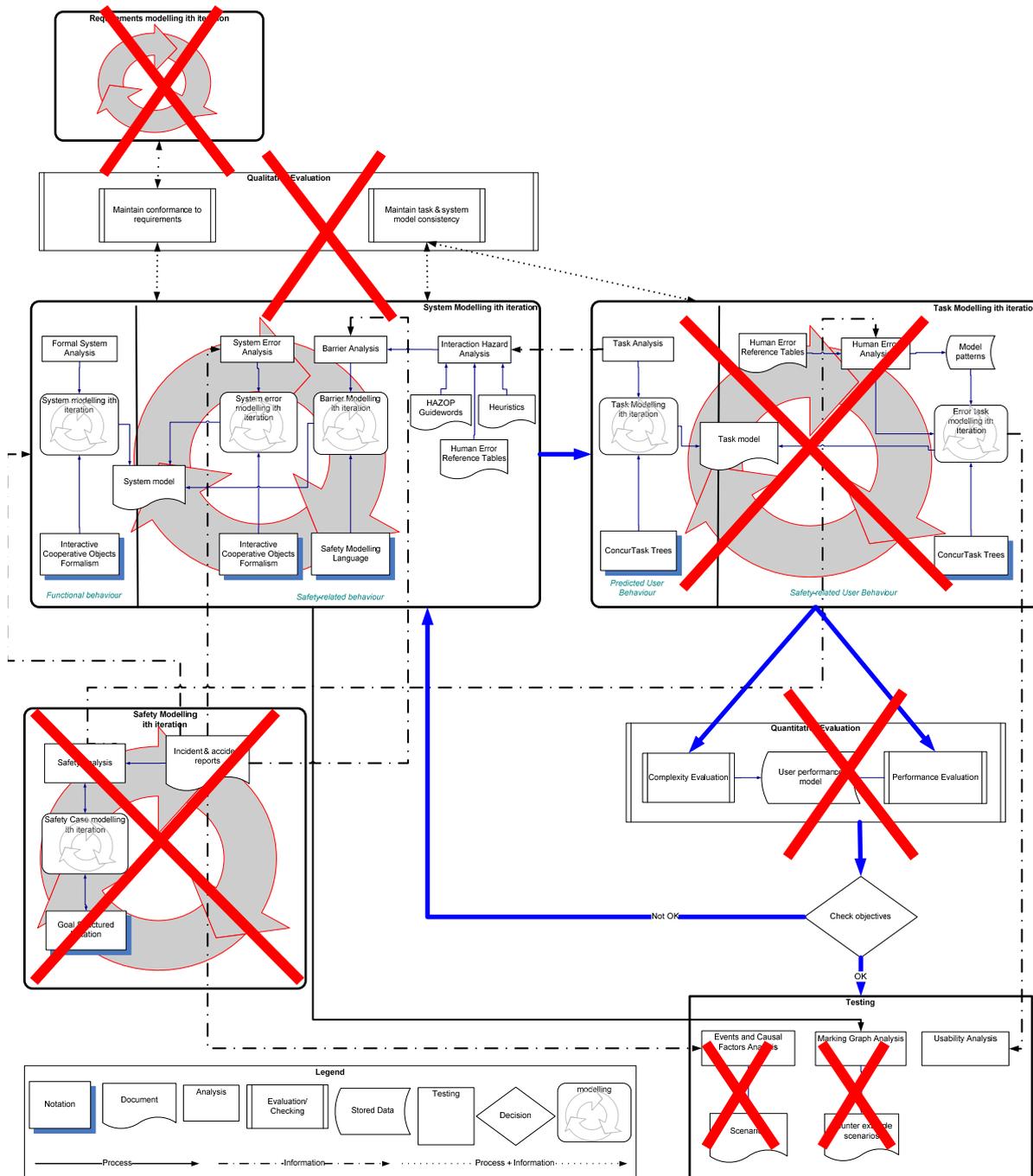


Figure 9-2. Phases of the Generic Integrated Modelling Framework not illustrated on this case study

3 Introduction to the case study

The aim of this section is to present the case study, the Multi Purpose Interactive Application (MPIA) User Application (UA) compliant with the ARINC 661 specification. The Airlines Electronic Engineering Committee (AEEC) (an international body of airline representatives leading the development of avionics architectures) formed the ARINC 661 Working Group to define the software interfaces to the Cockpit Display System (CDS) used in all types of aircraft installations. The standard is called ARINC 661 - Cockpit Display System Interfaces to User Systems (ARINC 661 2002; ARINC 661-2 2005).



Figure 9-3. MPIA Application in THALES Test Bench for interactive cockpits

The MPIA is a User Application (UA) that aims at handling several flight parameters. It is made up of 3 pages (called WXR, GCAS and AIRCOND) containing 12 different widgets as defined by the ARINC 661 specification. The crew member is allowed to navigate between the 3 pages using 3 buttons (as shown in Figure 9-5). The WXR page is responsible for managing weather radar information; GCAS is responsible for Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning.

The application can be controlled (though not at the same time) by the pilot and the co-pilot via keyboard and mouse interaction. Each MPIA window contains three main parts: The information area (on the top), the menu bar (on the bottom) and the workspace (central part).

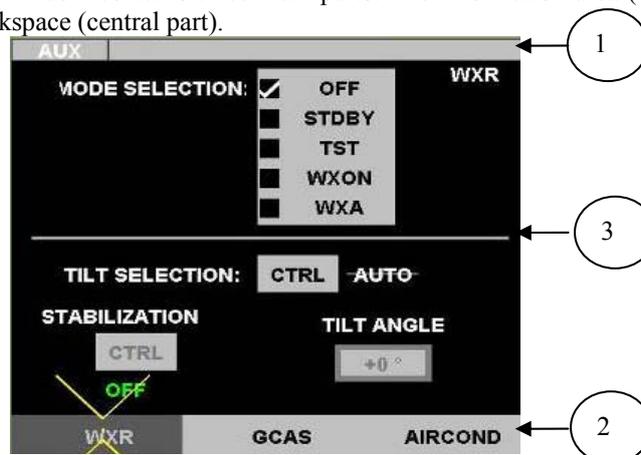


Figure 9-4: Multipurpose Interactive Application (MPIA)

- (1) **Information area**, split in two parts, left displays current state of application, right is set by default blank but displays error messages, actions in progress or bad manipulation when necessary.
- (2) **Workspace area**, display changes according to 'interactive control panel' selected. WXR workspace is dedicated to all modifiable parameters of the weather radar sensor. GCAS workspace is dedicated to

some of the working modes of GCAS. AIRCOND workspace is dedicated to selection of temperatures inside the aircraft.

(3) **Menu bar**, includes 3 tabs for accessing 3 main ‘interactive control panels’.

The following paragraphs present informal overviews of the operations of these three pages of the MPIA, however, we will focus on the WXR page for which the entire system is presented. The other pages are only presented here for completeness.

Informal Description of the MPIA application: WXR Page (weather radar)

The mode selection widget of this page behaves as a radio button widget, i.e. only one option can be selected at a time. The working mode, the tilt selection mode can be AUTO or MANUAL. (The interface symbol representing MANUAL is ~~AUTO~~). The CTRL push-button allows to swap between the two tilt selection modes. The stabilization mode can be ON or OFF. The dedicated stabilization CTRL push-button allows to swap between these two modes. Access to the stabilization CTRL button is forbidden when in AUTO tilt selection mode. The tilt angle: a numeric edit box permits to select its value within the range [-15°; 15°]. Modifications are forbidden when in AUTO tilt selection mode

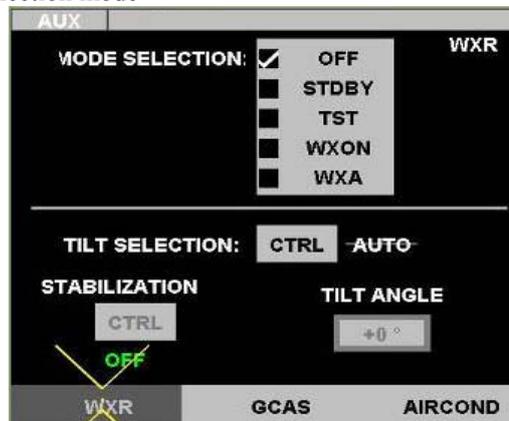


Figure 9-5 Snapshot of the page 1 of the MPIA UA (WXR) - a simple highly interactive application based on the look & feel of THALES

Informal Description of the MPIA application : GCAS Page (Ground Anti Collision System)

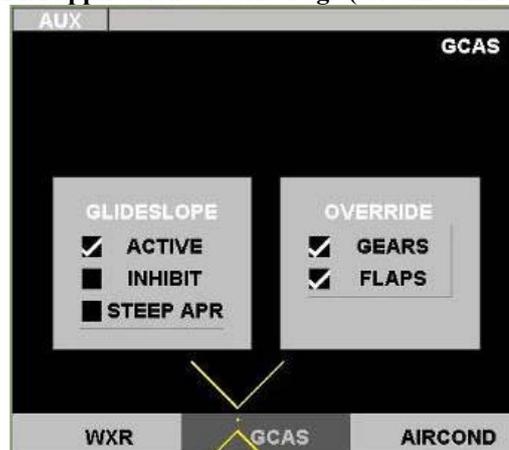


Figure 9-6 Snapshot of the page 1 of the MPIA UA (GCAS)

The GCAS page has fewer features than the WXR page, as can be seen in Figure 9-6. There are two windows providing options for Glidescope and Override. The Glidescope menu behave a radio button widget (i.e. only one option can be selected at a time) while the Override menu behaves as a checkbox allowing the crew to select 1 or more options at the same time.

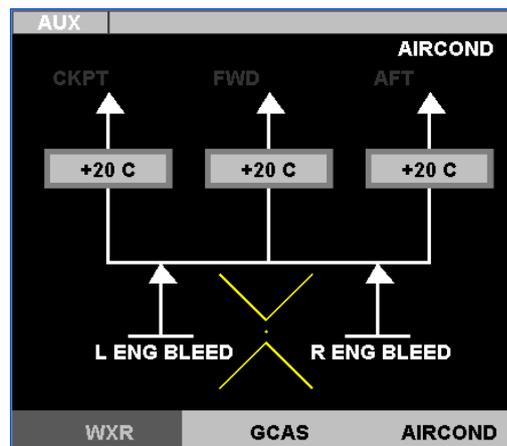
Informal Description of the MPIA application: AIRCOND Page (Air Conditioning)

Figure 9-7 Snapshot of the page 1 of the MPIA UA (AIRCOND)

On the AIRCOND page, the crew can use the mouse to select either the cockpit (CKPT), forward (FWD) or aircraft (AFT) area to enter numerical values for the temperature settings within a range of range [18°; 28°].

4 ARINC 661

Here we introduce the ARINC 661 specification in an informal way including the purpose and scope of this standard.

The aim of the ARINC 661 specification is to define interfaces of a Cockpit Display System (CDS) currently used in interactive cockpits of several aircraft manufacturers including Airbus, Boeing and Dassault.

4.1 Informal overview of ARINC 661

The Airlines Electronic Engineering Committee (AEEC) (an international body of airline representatives leading the development of avionics architectures) formed the ARINC 661 Working Group to define the software interfaces to the Cockpit Display System (CDS) used in all types of aircraft installations. The standard is called *ARINC 661 - Cockpit Display System Interfaces to User Systems (ARINC 661 2002; ARINC 661-2 2005)*.

The CDS (the software system embedded in an aircraft) provides graphical and interactive services to user applications within the flight deck environment. When combined with data from user applications, it displays graphical images and interactive components to the flight deck crew. It also manages user-system interactions by integrating input devices for entering text (via keyboard) and for interacting with these interactive components (via mouse-like input devices). ARINC 661 emphasizes the need for independence between aircraft systems and the CDS and fully defines software interfaces between the CDS and the aircraft applications. This includes the software interface between the avionics equipment and display system. However, this document does not specify the "look and feel" of any graphical or interaction technique information, and as such does not address human factors issues. These elements are meant to be defined by the aircraft manufacturers and/or by avionics equipment manufacturers.

The primary objective of ARINC 661 specification is to minimize the development costs, directly or indirectly by accomplishing the following (excerpt from *ARINC 661 2002*):

- Minimizing the cost of changing or adding new avionic systems.
- Minimizing the cost of adding new display functions to the cockpit during the life of an aircraft.
- Minimizing the cost of managing hardware obsolescence in an area of rapidly evolving technology.
- Introducing interactivity to the cockpit, thus providing a basis for airframe manufacturers to standardize the Human Machine Interface (HMI) in the cockpit.

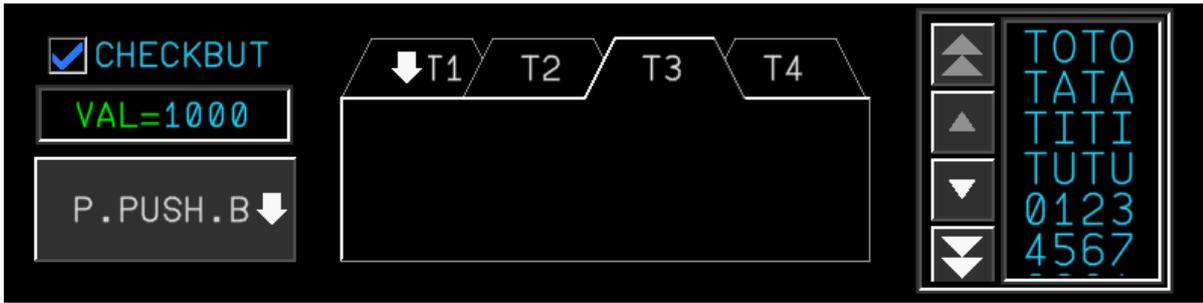


Figure 9-8. Examples of ARINC 661 Widgets (buttons, check box, edit boxes, labels, scroll lists)

ARINC 661 was adopted in October 2001 and first published in April 2002 and met critical requirements by Airbus. The first supplement to ARINC 661 introduced vertical display capabilities, eight new widgets, the Airbus A380 CDS versus needs for future CDSs. This supplement was published in June 2003. The first draft of the second supplement was published in September 2004 and introduced changes to ARINC 661 necessary for the Airbus A380 (NextFocusedWidget) and Boeing 7E7 cockpit display system development. It also included seven new widgets and the addition of state diagrams for interactive objects (p.209). Figure 9-8 and Figure 9-9 provide examples of ARINC 661 widgets

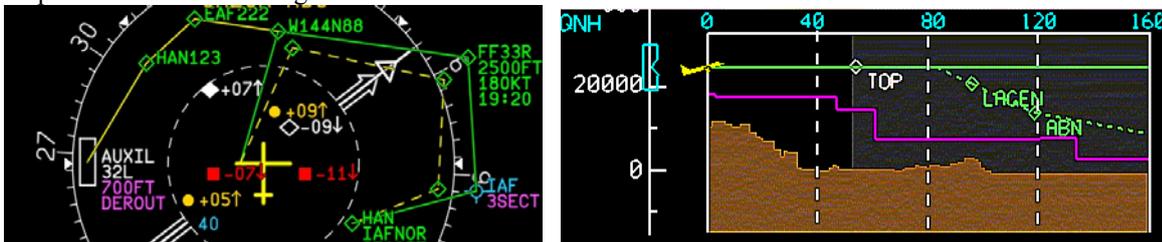


Figure 9-9. Examples of ARINC 661 widgets specific to cockpits (Map management, masks, AESS bitmap)

In (Barboni et al. 2006b), we defined an architecture that clearly identifies each part of this architecture and their communication, as shown on Figure 9-10. The aim of this architecture is also to clearly identify which components will be taken into account in the modelling activities (via the complete description of their behaviour using ICOs) and which ones are taken into account in a different way (via the entire description of their graphical representation exploiting Scalable Vector Graphic (SVG) facilities).

Such architecture presents two main advantages:

1. Every component that has an inner behaviour (server, widgets, UA, and the connection between UA and widgets, e.g. the rendering and activation functions) is fully modelled using the ICO formal description technique (see icons with a Petri net in Figure 9-10).
2. The rendering part is delegated to a dedicated language and tool (such as SVG or Java 2D) represented by the box closed to the users on the left-hand side of Figure 9-10.

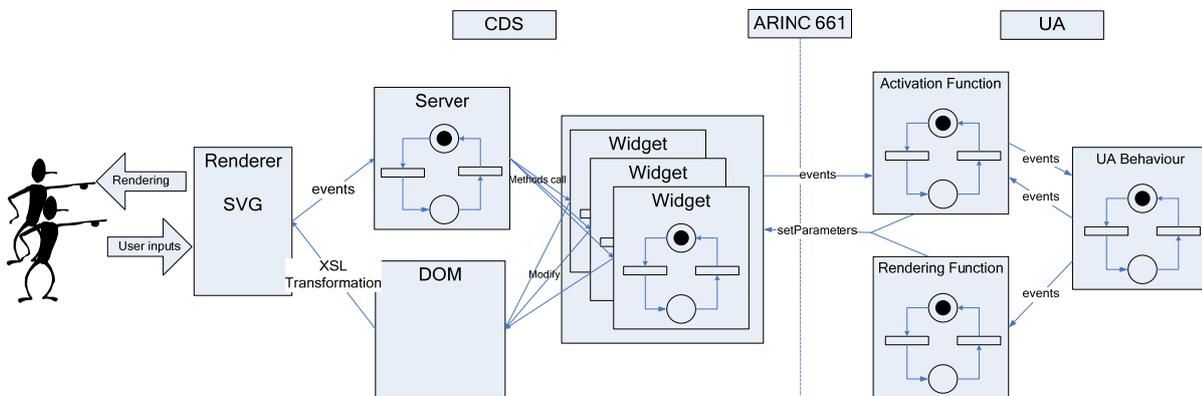


Figure 9-10 Detailed architecture to support ARINC 661 specification

Figure 9-10 is split in 3 parts. The right hand side corresponds to the user applications and the left-hand side to the widgets and the interface server. ARINC 661 mainly addresses the communication protocol between UAs and CDS (dotted line between the two parts).

In opposition of “classical” work on interfaces for cockpits that target at extending interaction in the cockpit (as for instance in (Faerber R et al. 2000) (Harel 1987), the aim of the ARINC 661 specification is to provide a systematic and standardised way to engineer interactive application in the cockpit.

As ARINC 661 specification is devoted to aircraft cockpit, certification problems are raised. Therefore our main contribution is to use ICOs to precisely exhibit ambiguities or underspecifications in ARINC 661 specification and furthermore, illustrate how the interaction hazard analysis for identifying and modeling software barriers can be applied to such a real-life safety-critical interactive software application.

5 System Model of the MPIA Application

In order to show the modelling of software barriers, we must first present the system model of the MPIA. Since the modelling of the MPIA is particularly complex, we focus on the three main aspects of the application, the weather radar page, the ground anti-collision system page and the air condition page. User Interface Server modelling is not presented in this chapter as we are focussing on applications rather than underlying software infrastructures on top of which applications are executed.

5.1 Formal Description of the MPIA application: WXR Page (weather radar)

The following sections provide formal descriptions of the behaviour of each of the three MPIA windows. This includes the behaviour, the activation function and rendering function. We do not repeat here the presentation parts since these have already been presented above (see screenshots of MPIA application in Figure 9-5, Figure 9-6, Figure 9-7).

5.1.1 Behaviour

Figure 9-11 presents the ICO model describing the behaviour of the WXR page. The behaviour is split into two distinct (unconnected) parts. The upper part, concerning the 5 check buttons and the lower part concerning the 2 picture push buttons and the numeric edit box. The principle behaviour of this model is as follows. For the upper part, while the operator clicks on one of the 5 check boxes, the corresponding transition in the model is fired which modifies the value of `MODE_SELECTION` (`OFF`, `STDBY`, `TST`, `WXON`, `WXA`). For the lower part of the model, each time the operator presses the upper `CTRL` button (relating to the tilt selection), a token is moved between place `AUTO` and place `NOT-AUTO`. If a token is in place `NOT-AUTO` the second `CTRL` button (relating to the stabilization) becomes clickable and passes a token from place `STABILIZATION_ON` and `STABILIZATION_OFF` while the transition `changeAngle_TI` becomes friable making the numerical edit box (for the tilt angle) active. If a value is entered in this edit box, the value contained in place `TILT_ANGLE` is modified.

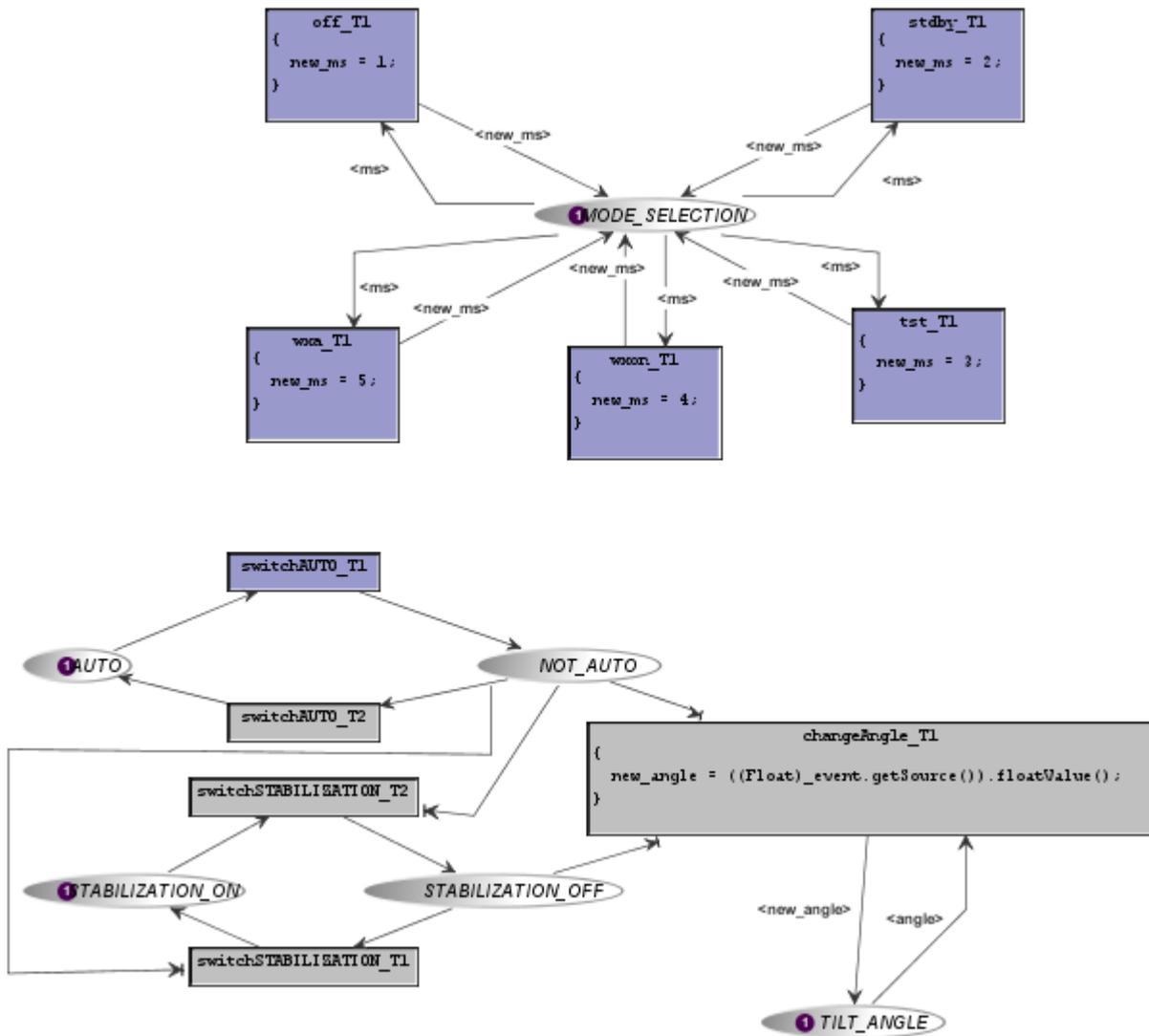


Figure 9-11. ICO Model of the WXR page of the MPIA presented in Figure 9-5

5.1.2 Activation Function

As presented in chapter 1, section 5 the activation function is dedicated to connecting the user interfaces widgets to the behaviour of the application described using a Petri net. Table 9-1 presents the activation function of the WXR page. First column of the tables contains the entire set of widgets available in the page. The second column describes the ARINC 661 user event that triggers a state change in the application. Third column contains a set of methods that have to be called when the corresponding widget becomes enabled on the user interface. The last column makes explicit the transition (in the Petri net) that has to be fired when the corresponding event is triggered by the user on the corresponding widget.

For instance line six must be read as follows:

The widget of type PushButton named AUTO (the first one with the caption CTRL on the user interface see Figure 9-5) can be clicked by the user (using any means like a graphical input device like KCCU or a keyboard short cut) and in such case a A661_EVT-SELECTION event will be received. Upon the reception of this event the transition named switchAUTO in the Petri net model of Figure 9-11 will be fired. The last column is not called “transition” as in fact several transitions in the Petri net model can be connected to a same user-event. Such set of transitions is called a user service. Indeed, the AUTOPicturePushButton is connected to 2 transitions called switchAUTO_T1 and switchAUTO_T2. Only one of these two transitions must be fireable at a time (otherwise this means there is indeterminism in the Petri net model. Other widgets in the WXR page are connected to user services featuring only one transition (like OFFCheckBox connected only to transition Off_T1).

Table 9-1. Activation function of WXR page

Widget	Event	Rendering Method	User Service
OFFCheckBoxButton	A661_EVT_STATE_CHANGE	setEnabled	Off
STDBYCheckBoxButton	A661_EVT_STATE_CHANGE	setEnabled	Stdby
TSTCheckBoxButton	A661_EVT_STATE_CHANGE	setEnabled	Tst
WXONCheckBoxButton	A661_EVT_STATE_CHANGE	setEnabled	Wxon
WXACheckButton	A661_EVT_STATE_CHANGE	setEnabled	Wxa
AUTOPushButton	A661_EVT_SELECTION	setEnabled	switchAUTO
StabPushButton	A661_EVT_SELECTION	setEnabled	switchSTABLIZATION
AngleTextField	A661_EVT_STATE_CHANGE	setEnabled	changeAngle

5.1.3 Rendering Function

Table 9-2 presents the rendering function of the WXR page. That function is in charge of changing the presentation of the user interface when a state change occurs in the Petri net model. As states are represented by tokens in the Petri net model, the rendering function connects graphical methods places (that store tokens).

Only two places in the Petri net model are connected to a graphical rendering namely places MODE_SELECTION and TILT_ANGLE. The first line of Table 9-2 shows that when a token enters in place MODE_SELECTION the method showModeSelection is executed and that this method takes as a parameter the value of the token that has entered (ms).

Table 9-2. Rendering function of WXR page

ObCS	State change	Rendering method
Place MODE_SELECTION	Token <ms> enters	showModeSelection (ms)
Plaе TILT_ANGLE	Token <angle> enters	showAngle(angle)

Now that we have presented the entire system models of the WXR page of MPIA, we show, using our analysis of software barriers, how this model can be improved with respect to safety issues related to hazardous interactions.

6 Hazardous Interaction Analysis

We remind the reader of the approach applied for this hazardous interaction analysis in Figure 7-5.

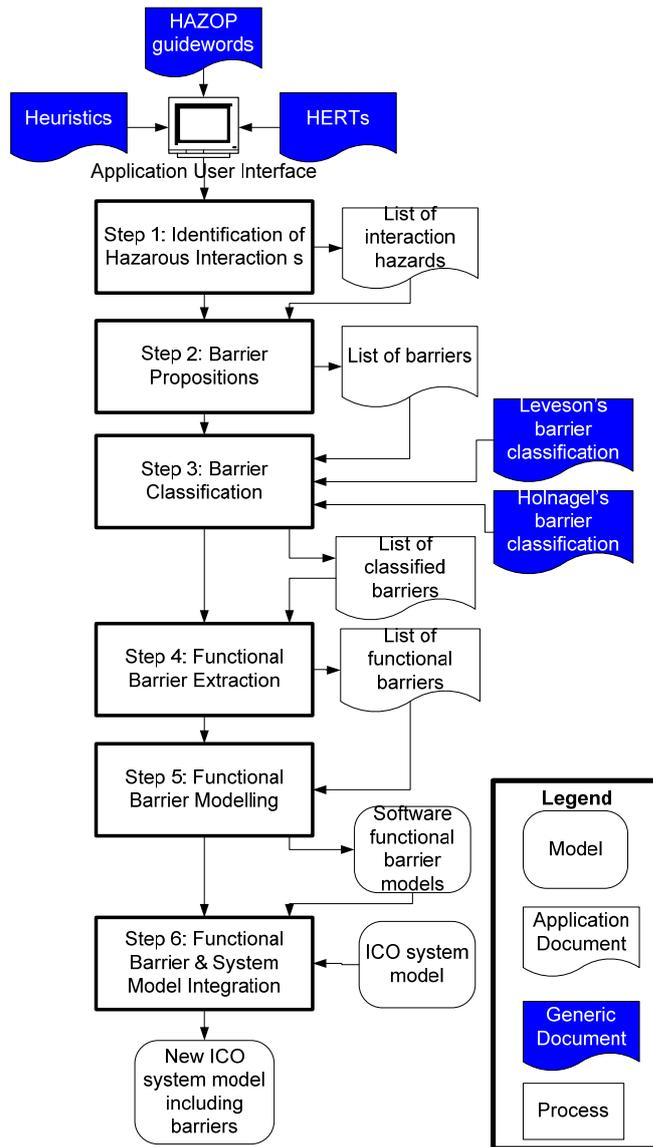


Figure 9-12. Software Barrier Identification, Analysis and Modelling Approach

We have applied the HERTs (see Chapter 4) directly to the Weather Radar (WXR) part of the application without using any information from pilot’s tasks. Using HERTs we have been able, for each type of human error, to deduce possible erroneous interactions with the system and what the impact of the identified ‘error’ will be on the task in hand. The aim of these reference tables is not to guarantee a comprehensive and exhaustive identification of every possible eventuality but to provide investigators with systematic ways of exploring likely and reoccurring user deviations. For example, Table 9-3 documents the outcome of this process focussing on the skill-based errors. The key benefit of this human ‘error’ analysis is that its style not only helps to identify potential interaction problems that led to a particular incident (if analysing after an incident), it can also be used to identify other alternate problems that might jeopardise the future safety of an application but which did not arise in this specific accident.

Table 9-3. Skill based Human Error Analysis Results on Weather Radar (WXR) page

Error Type	Example	System Impact
Strong-habit intrusion	Select mode used more frequently Enter a common tilt angle	Unknown real life impact, mode will change Tilt angle will change to possible undesired angle

Strong-habit capture	Intend to change mode but don't	Unknown real life impact, mode remains the same
	Intend to change tilt selection to manual/automatic but don't	Unknown real life impact, mode remains the same
	Intend to change stabilization to off/on but don't	Unknown real life impact, mode remains the same
	Intend to change tilt angle but don't	Unknown real life impact, mode remains the same
Strong-habit exclusion	Click on tilt angle to change but then don't	Unknown real life impact, tilt angle remains same
	Change tilt selection to manual but then don't change tilt angle	Unknown real life impact, tilt selection remains auto
Branching errors	Set tilt selection to manual, change tilt angle, but don't change tilt selection back to auto	Unknown real life impact, tilt angle changed and tilt selection remains set to manual
Overshooting a stop rule	Change to WXR tab to change mode selection, but change tilt angle instead	Mode remains same, tilt angle is changed
Program Counter Failures	Change tilt selection mode to manual, get distracted, then omit the tilt angle change	No impact
Trips	Interruption while changing tilt angle, therefore tilt angle change incomplete.	Unknown – probable that current tilt angle remains active
Detached Intentions	Intend to change tilt selection mode but click on stabilization mode instead because of external discussion	If tilt selection was in manual, then stabilization mode is available and becomes on or off depending on its current state If tilt selection was in auto, then stabilization mode is unmodifiable
Environmental Capture	Go to WXR tab, change mode selection, go to another tab without changing tilt angle	Unknown real life impact, mode changes, tilt angle remains the same
I-should-be-doing-something-but-I-can't-remember-what	Go to WXR tab, stare at interface without remembering what should be doing	No impact
Fumbles	?????	
Post-completion error	Change tilt selection to manual, change tilt angle, forget to change tilt selection back to auto afterwards	Unknown real life impact, tilt angle changed and tilt selection remains set to manual
Under-motivation	Try to change tilt angle without first changing tilt selection mode to manual	Tilt angle is unmodifiable
Description error	Click on stabilization button instead of tilt selection button	If tilt selection was in manual, then stabilization mode is available and becomes on or off depending on its current state If tilt selection was in auto, then stabilization mode is unmodifiable
	Change tilt angle in a temperature character box	Temperature will be set to figure entered or a figure nearest that entered if outside the 18-28 degree range and tilt angle will remain the same
Input or Misperception errors	Not realise that stabilization mode can only be changed when tilt selection is in manual	Will cause delay in interaction
Data driven error	Enter a tilt angle figure of a different figure that is currently being looked at or discussed	Tilt angle will be set to the degree entered or a degree nearest to that entered if outside of the +15 -15 degree range.

Associative-activation error	Enter a tilt angle figure of a different figure that is currently being thought about	Unknown
Omission	Try to change tilt angle without first changing tilt selection mode to manual	Tilt angle is unmodifiable
Repetition	Click on tilt selection button twice Change tilt angle twice Click stabilization mode Click a mode twice	Remains in initial mode setting No impact Remains in initial mode setting No impact
Reversal	Try to change tilt angle and then change tilt selection mode Try to change stabilization mode and then tilt selection	Tilt angle is unmodifiable Stabilization mode is unmodifiable
Insertion	Aiming to change tilt angle, change mode selection in between process	Assuming the new mode does not effect the tilt angle, there will be no impact except being in a new mode with the new tilt angle
Replacement	N/A	Unknown
Premature	??	
Order Errors	Changing tilt angle before changing tilt selection mode	Tilt angle is unmodifiable
Over Motivation	N/A	

We have chosen to use Nieslen’s 10 usability heuristics due to the nature of MPIA application whose interaction and visualization techniques correspond directly to the type of desktop application targeted by this kind of heuristics. It has been argued that such a UIM is not the most effective for identifying potential usability problems (Cockton et al. 2003). We understand that for larger, more complex systems involving complex activities from users, it would be more appropriate to base the evaluation on a user goal and involve analysts to predict usability as we have done while identifying potential scenarios of interaction with a system (Basnyat et al. 2005b).

The results of the heuristic evaluation are provided in Table 9-4. The results have been used to identify potential software barriers. By this we do not mean barriers to accessibility as in (Killam and Holland 2003), instead we mean potential software barriers that could help prevent erroneous user interactions.

Table 9-4. Heuristic Evaluation Results

	WXR Workspace	GCAS Workspace	AIRCOND Workspace	All Tabs	
				Info bar	Menu
Visibility of system status	Mode selection status clear, tick box indicator & selection highlight Tilt selection, Stabilization and Tilt angle status less clear	Glidescope selection status clear, tick box indicator & selection highlight Override selection less clear since selections can be neither, either or both options irrespective of Glidescope selection	Current temperature settings clear	Permanently ‘AUX’	Current selected tab clear
Match between system and the real world	Abbreviations could be misleading to non-experienced pilot	Abbreviations could be misleading to non-experienced pilot	Layout of cockpit, forward cabin and afterward cabin temperature setting displays do not match real world layout of aircraft	Permanently ‘AUX’	Abbreviations could be misleading to non-experienced pilot

User control and freedom	No undo/redo however user can easily undo/redo their actions	No undo/redo however user can easily undo/redo their actions	No undo/redo however user can easily undo/redo their actions	N/A	No undo/redo however user can easily undo/redo their actions
Consistency and standards	N/A	Glidescope selection behaves as a 'standard' radio button design style though represented as checkbox	N/A	Permanently 'AUX'	N/A
Error prevention	No confirmation options	No confirmation options	Temperature confirmation window available with "ok" button only	N/A	N/A
Recognition rather than recall	Recall of abbreviations is necessary	Recall of abbreviations is necessary	Recall of abbreviations is necessary	Permanently 'AUX'	Recall of abbreviations is necessary
Flexibility and efficiency of use	Unavailable	Unavailable	Unavailable	N/A	Unavailable
Aesthetic and minimalist design	Simple minimalist design	Simple minimalist design	Simple minimalist design	Simple minimalist design	Simple minimalist design
Help users recognize, diagnose, and recover from errors	None available	None available	None available	None available	None available
Help and documentation	None available	None available	None available	None available	None available

HAZOP is a systematic technique that attempts to consider events or process in a system exhaustively. Within a particular system domain (or scenario), items (or events) are identified and a list of guidewords is applied to the items. The guidewords prompt consideration of deviations to item behaviour (guideword examples include less, more, none, more than) to elicit the potential for arriving at possible hazardous states (Smith and Harrison 2002). These guidewords provide the structure of the analysis and can help to ensure complete coverage of the possible failure modes (Pumfrey 1999). Although a HAZOP analysis should generally be applied to a Piping and Instrumentation Diagram, we apply the analysis to the WXR UI. The results of this analysis can be seen in Table 9-5.

Table 9-5. HAZOP analysis results on WXR tab

Guideword	Deviation	MPIA WXR Interpretation
NO Or None	No part of the intention is achieved	No tilt angle is entered
More	Quantitative increase in a physical property (rate or total quantity)	Higher tilt angle is entered
Less	Quantitative decrease in a physical property (rate or total quantity)	Lower tilt angle is entered
More Than Or As Well As	All intentions achieved, but with additional effects (qualitative increase)	Change tilt angle, also change stabilization mode to off
Part Of	Only some of the intention is achieved (qualitative decrease)	Change tilt selection to manual, but do not change tilt angle

Other Than	A result other than the intention is achieved	Mistype new tilt angle but do not realize, new state is undesired
Reverse	The exact opposite of the intention is achieved	Tilt angle is not entered/changed
Early		Not analysed, could be applicable to procedures of the pilots
Late		Not analysed, could be applicable to procedures of the pilots
Before		Not analysed, could be applicable to procedures of the pilots
After		Not analysed, could be applicable to procedures of the pilots

7 Software Barrier Identification

This section is dedicated to the discussion of human-computer interaction/software barrier identification. In previous chapters, and in the previous case study chapter, we used incident and accident reports to identify existing (successful and failed) socio-technical barriers. The case study in this chapter differs from our previous work in that it is a software application and there exists no known accident report with reference to this particular application. This section describes the barriers that were identified following the three analyses. The barriers identified following the heuristic analysis relate mainly to the interface design, layout and representation.

The first barrier (B1) (see Table 9-4 visibility of system status) is related to graphical feedback provided by the system and should improve user understanding of the current state of the system. For example, the current status of tilt selection, stabilization, tilt angle and Glidescope selection are not clear. A second barrier (B2) (see Table 9-4 match between system and real world) could be an improvement in the design of the AIRCOND page such that the layout of the modifiable temperatures resembles the layout of the aircraft to reduce possible misinterpretation. In the GCAS page there is a restriction on selecting more than one 'GLIDESCOPE' option at any given time. Options include ACTIE, INHIBIT or STEEP APR. However, in relation to consistency and standards, symbolic coding should be imposed (B3) (see Table 9-4 consistency and standards) such that radio style button interaction is represented as radio buttons and not as checkboxes. Error prevention is non-existent in the application. Symbolic warning (visual or auditory for example) is required (B4) (See Table 9-4 Error prevention) for each mode change, tilt angle changes and temperature changes that could conflict with other modes and settings. However, it is not necessarily useful to have a warning for every modification. Providing reference (B5) (See Table 9-4 Recognition rather than recall) to abbreviations could help reduce possible misinterpretation of labels used. Furthermore, a combination of training (non-technical barrier) and software feedback could help users recognize, diagnose, and recover from errors.

To minimise the likelihood of the issues relating to the categories, detached intentions, under motivation, description error, input or misperception, omission, reversal, insertion and order errors, a barrier relating to the three tabs of the interface should be imposed. In relation to the description error issue, the barrier (B6) (See Table 9-3 description error) should enforce clear distinctions between button behaviour. For example, the Tilt selection mode and stabilization mode buttons are both labelled CTRL however the behaviour of CTRL for Tilt selection is auto/manual and the behaviour of CTRL for stabilization mode is on/off.

The data driven error example could be minimised with a barrier (B7) (See Table 9-3 data driven error) providing a symbolic warning in relation to the difference of the figure entered and the acceptable figure. There is no such barrier implemented in the MPIA. The tilt angle range (lower right hand corner of the right-hand side of Figure 9-5) must be between -15° and $+15^{\circ}$. If a tilt angle outside of the range is entered, no warning is provided, the angle is set to the nearest entered, i.e. if -19 is entered, the tilt angle will be set to -15° , if $+40$ is entered, the tilt angle will be set to $+15^{\circ}$. The same rule applies for the temperature setting however the range is between 18° and 28° degrees. The preventative barrier therefore would be to inform the pilot or co-pilot that their command has not been accepted. Also it was noted that if a character is entered into the tilt angle text box as opposed to a number, the previously set angle disappears and becomes simply "+ °". We do not know the impact this would have on the operation of the application within a cockpit.

The HAZOP analysis highlighted two further barriers in addition to exemplifying barriers 4 and 7 which had already been identified. (B8) (see Table 9-5 part of) calls for a restriction on available interaction until the current task in hand is complete, in this case entry of a new tilt angle for the WXR. (B9) (see Table 9-5 more than/as well as) aids the user to follow a correct sequence of actions by modifying the UI. This would also support potential cognitive "errors" such as a post-completion error.

8 Barrier Classification

In total, 10 barriers (after generalising the findings) were identified using the three complementary techniques. They can be considered a combination of protective and preventative barriers. These barriers are classified (see Table 9-6) according to Hollnagel's classification of system barriers and Leveson's classification of software barriers. This is particularly useful because it is possible to then allocate each type of barrier to a relevant design model for accurate representation. We are interested in functional barriers since this is where we can make most contribution to the field. The functional barriers can be modelled using the ICO notation and integrated with an existing system model of the MPIA using the same notation to induce a safer interaction system. The ICO CASE tool, Petshop supports simulation of models, thus once the barrier has been integrated, it is possible to view the impact it would have on the system and human interactions with the system

8.1.1 Barrier Classification

In total, 10 barriers (after generalising the findings) were identified using the three complementary techniques. They can be considered a combination of protective and preventative barriers. These barriers are classified (see Table 9-6) according to Hollnagel's classification of system barriers and Leveson's classification of software barriers. This is particularly useful because it is possible to then allocate each type of barrier to a relevant design model for accurate representation. We are interested in functional barriers since this is where we can make most contribution to the field. The functional barriers can be modelled using the ICO notation and integrated with an existing system model of the MPIA using the same notation to induce a safer interaction system. The ICO CASE tool, Petshop supports simulation of models, thus once the barrier has been integrated, it is possible to view the impact it would have on the system and human interactions with the system.

Table 9-6. Classification of Suggested Barriers

No	Barrier	Analysis Technique Identifier	Leveson's software barrier classification	Hollnagel's system barrier classification
B1	Improve user understanding of current system state	Heuristic	N/A	Symbolic Indicating
B2	Improve design of AIRCOND workspace such that the layout of the modifiable temperatures resembles the layout of the aircraft	Heuristic	N/A	Symbolic Indicating
B3	Use radio style button interaction for GLIDESCOPE option in GCAS workspace	Heuristic	Interlock	Symbolic Indicating Functional Soft
B4	Symbolic warning is required for all mode changes, tilt angle changes and temperature changes.	Heuristic HAZOP	Lockout	Symbolic Indicating, Functional Countering, Preventing, Hindering
B5	Provide references and ensure correct training regarding abbreviations.	Heuristic	N/A	Symbolic Indicating Immaterial Prescribing
B6	Enforce clear distinctions between button behaviour	HERT	N/A	Symbolic Indicating, Countering
B7	Provide a symbolic warning in relation to the figures entered and acceptable figures.	HERT + HAZOP	Lockout	Symbolic Indicating Countering
B8	If tilt angle is not entered, system should not allow other actions to be available until task in hand is complete i.e. tilt angle entered	HAZOP	Lockin	Functional Soft Preventing, Hindering
B9	When sequences of actions should be performed in a certain order, the UI should guide the user	HAZOP	Interlock	Functional Soft Preventing
B10	Existing barrier: restricts use of two mice at same time for pilot & co-pilot	None general observation	– Lockout	Symbolic Countering Indicating Functional Soft Preventing Hindering

9 Software Barrier Modelling

In this section we present the modelling the some functional barriers (B4, B8, B9). The implications of the identified barriers should be tested on the system in order to justify them. This could be achieved by formal specification of the interactive system and is possible using the Interactive Cooperative Objects (ICO) formalism.

9.1.1 An example of Barrier Modelling

This section presents how the ICO models have been modified in order to integrate the functional barriers that have been identified using the framework presented above. We have chosen to illustrate barrier B9 as it is both representative and still has a limited impact on the models. Figure 9-13 is a screenshot of the tilt angle edition ICO model produced using Petshop environment (the same model is presented in Figure 9-11) The image has been numbered 1-4 for explanatory purposes.

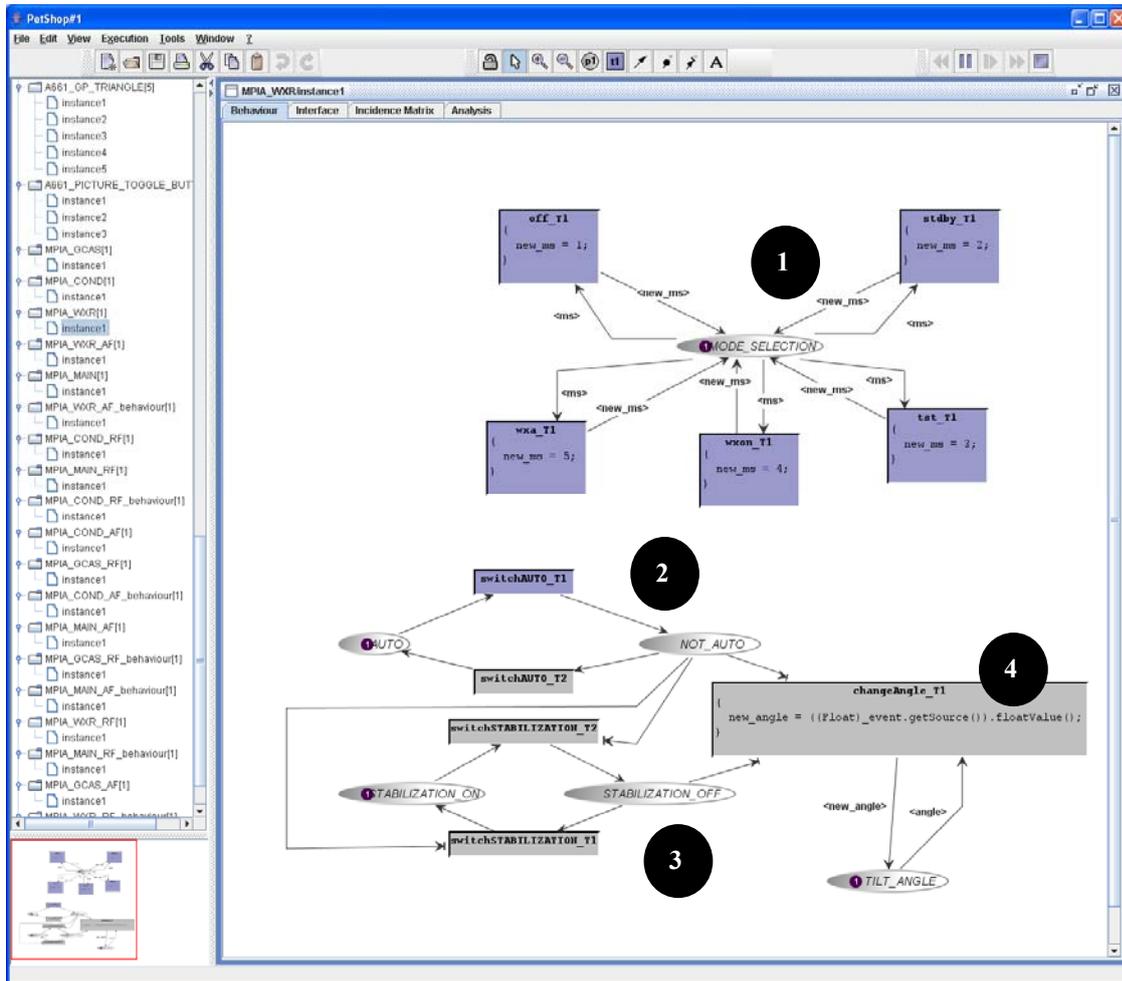


Figure 9-13. Behaviour of the Tilt angle edition using ICOs

The five blue transitions (label number 1) represent the behaviour of the mode selection in the WXR workspace, i.e. off, stdby, tst, wxon and wxa (See upper part of Figure 9-5). The behaviour of the tilt selection button is represented near label 2 in Figure 9-13, auto and manual. The lower part of the Petri net, label 3, represents the stabilization button behaviour, on and off. Finally the transition the behaviour of the tilt angle edit box (label 4). The edit box transition is fireable when tilt selection place *not_auto* and stabilization place *stabilization_off* contain tokens i.e. representing the fact that the systems is in a adequate state for allowing the crew to modify the angle. The system model described above has been modified in order to represent barrier B9.

Figure 9-14 illustrates the modifications made to the model. A new place, Lock has been added to the model (label 1). This place plays the role of a semaphore providing an exclusion mechanism to any other action in the WXR page when it does not contain a token. The additional transition StartEditing removes that token when the

user activates the Tilt-angle edition. Transition FinishEditing puts the token back to place Lock. Meanwhile no other action is available and thus all other interface interactions are unavailable meeting the requirements of the barrier B9.

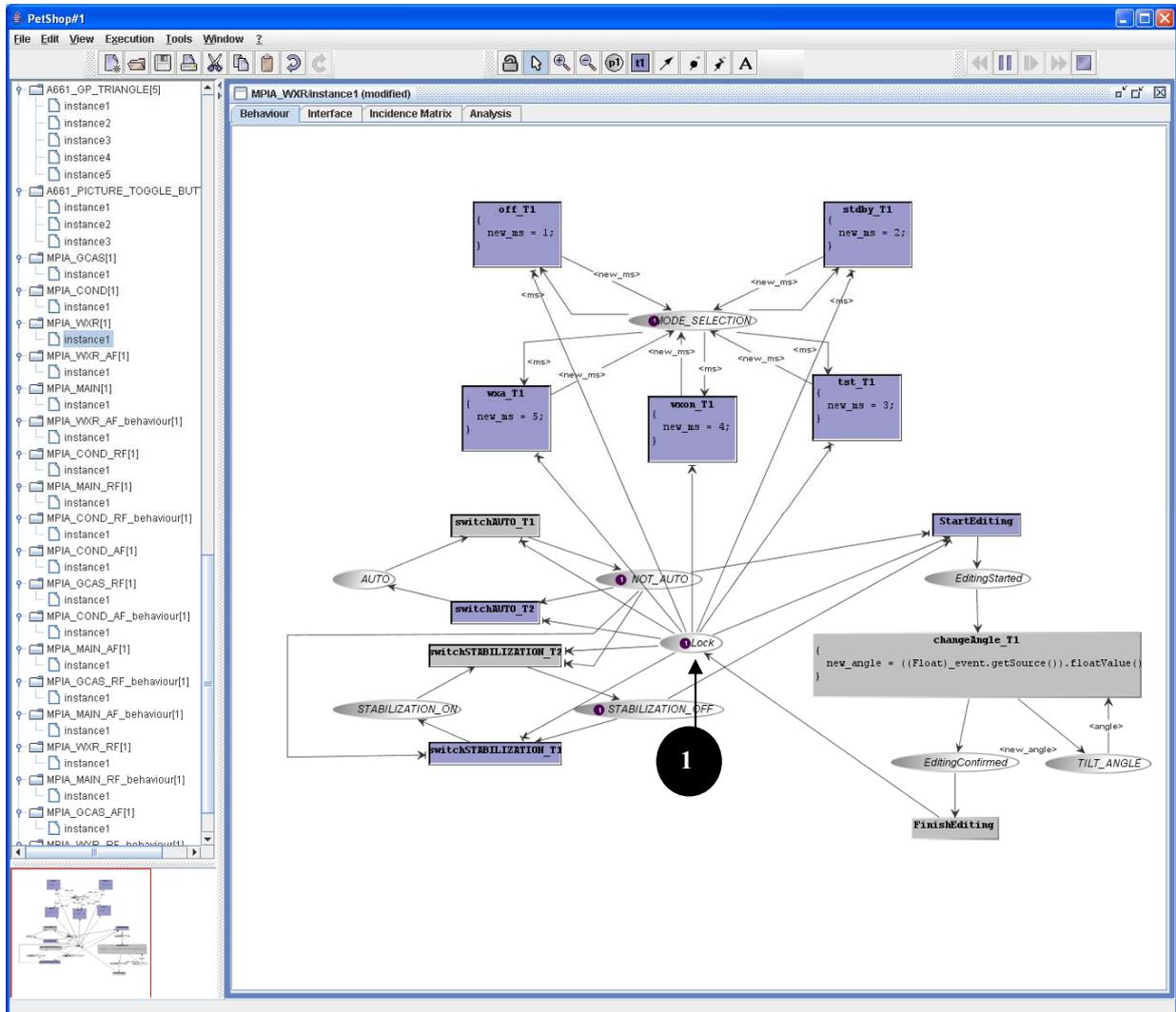


Figure 9-14. B9 Barrier modelling for Tilt angle edition

10 Conclusion

We have presented a systematic approach for identifying user interaction hazards and their related mitigating software barriers based on three complementary analysis techniques: heuristic, human error and HAZOP analysis. We have used the ICO formal description technique to model the extracted functional barriers and integrate then with an existing model of a complete real-life embedded interactive cockpit application. The modelling of such barriers allows us to simulate potential user interactions with the system to ensure that the barrier eliminates the previously identified hazards.

The specification and modelling parts of the study have shown that it is possible to formally model the entire behaviour of a real-life safety-critical interactive application. The interactive cooperative objects (ICO) formal description technique has been adapted to meet the requirements of user applications compliant with ARINC 661. This adaptation has been applied to the MPIA application.

Modelling interactive applications (even relatively simple types as the MPIA) require the creation of numerous models (37 models for 91 instances). Without an adequate tool that allows the entry of data as well as the simulation of models and without an adequate modelling process, this task would not be possible due to industrial resource constraints (human, time and financial).

The approach also aids in identifying further barriers such as symbolic and immaterial barriers that should be represented and taken into account within the development process, however not within the system model.

In addition to the above mentioned results, the study also provided interesting prospects. In an interactive cockpit application like MPIA, we noticed that both the pilot and co-pilot are interacting with the same application. In the current deployed version of MPIA multi-user interaction is not allowed as only the first cursor entering a window is active (it becomes inactive after a period of inactivity). However, taking into account multi-user interaction techniques which allow collaborative interaction between the pilot and co-pilot might become a requirement for future cockpit systems. We have already shown the capability of ICOs and PetShop for dealing with multimodal interactions (Navarre et al. 2006) but we also believe that addressing that aspect within real-size applications might raise additional challenges.

Conclusions and Perspectives

The research presented in this thesis concerns the Human Computer Interaction (HCI) discipline applied to the safety-critical industry. It is clear, from interactions we experience on a daily basis that poor design can lead to frustration and unintentional mistakes when interacting with devices. Such erroneous interactions in the field of safety-critical systems can have a negative impact on the behaviour of the system and potentially jeopardise the safety of the system.

When adding a human to the control of complex safety critical interactive systems, such as aircraft cockpit, Air Traffic Control workstations, power plant workstations etc, the unpredictable nature of humans can bring human unreliability issues in addition to technical unreliability. Even though classical development processes are employed for the development of such systems, it is not simple. The limits of such processes lie in their cost in terms of time and money, in terms of new iterations of the same process due to inadequacies between the produced system and the real requirements.

Summary of the thesis

The hypothesis of the thesis is that safety-critical interactive systems are not reliable. There exists no design method that allows to increase the reliability, usability and safety of these systems. We therefore provide a framework to take into account human errors, make explicit barriers and describe these concepts using formal methods. This allows us to increase the reliability, usability and safety (by making the system more tolerant to fault, usage errors and technical errors).

Addressing Usability

We believe User Centred Design (UCD), an established HCI design approach to be the most appropriate for the design of interactive systems, and more specifically safety-critical interactive systems, because it particularly targets usability of systems. However, existing methods in current state of the art in UCD, such as participatory design, storyboarding, usability testing, brain storming etc, are mostly informal. When the system under design is safety-critical, potentially involving loss of human lives, damage to the environment and loss of large sums of money, informal methods are not sufficient in terms of expressive power and scalability; furthermore, they do not provide a means for complete, concise and unambiguous specifications.

Addressing Reliability

To deal with these issues we advocate the use of formal methods for making explicit specifications with no ambiguities. This is particularly important to avoid things like automation surprise and unknown system states that the operator may find themselves in. While the use of formal descriptions is currently not the mainstream in human factors and UCD, formal descriptions of system behaviour and requirement specifications are used in many software development processes (e.g., the Unified Modelling Language methodology). However, we advocate in this thesis the use of formal modeling to guarantee a certain quality of the system.

We have shown in this thesis, the use of models provides a means to highlight the relationships between different actors in the design process and provides a high-level description of elements necessary for the development of a system. Typical models promoted in UCD practice are task models and user models. However, the notations used for these models are semi-formal and inadequate for specifying safety-critical user tasks. We believe that when designing an interactive system, the task model can greatly influence the system model and vice versa. The models can be cross-checked to ensure that the system model supports operator tasks. For this reason, we focus on the task and system model in this thesis. The task modeling has been performed using the ConcurTaskTrees (CTT) notation, while the system modeling has been performed using the interactive Cooperative Objects (ICOs) formalism, a formal specification technique based on Petri nets, dedicated to the specification of interactive systems.

Addressing Safety

Current UCD approaches and model-based design methods do not explicitly account for potential erroneous human and technical behaviour. Such information is vital for the successful design of a system. It is especially important for a safety-critical interactive system that potential problems are considered in advance. Also, there is currently no UCD technique for accounting for known incidents and accidents on the same or similar system to that under design. We address this issue by providing means for systematically representing safety related issues such as accident scenarios, identified failed and successful barriers and human-related procedural data.

Contribution

In this thesis, we have proposed a multidisciplinary generic integrated modelling framework. The framework provides means (notations, tools, techniques, methods etc) for the systematic inclusion of human factors issues, safety issues and reliability issues throughout the model-based development process of safety critical interactive systems. The framework could also be applied to non-safety critical systems by excluding unrelated phases such as the safety modeling phase. The aim of the framework is to improve reliability, safety and usability in the design of such systems by incorporating human factors issues, incident and accident investigation data and barriers (socio-technical, technical and human) in the design to inform the task and system models.

We do not imply that specialists of different domains (who will have different backgrounds and experience) should use modeling notations and techniques that they are not used to. We believe it is important that domain experts continue using the methods they are used to. We simply provide a means for this data to be represented such that the several domains of expertise can be used to inform one another.

The framework is comprised of four main phases, the task modeling, system modeling, safety modeling and testing phase. The phases are interrelated with various types of information and data propagating between them. For example, data in an incident or accident report represented in the safety modeling phase may feed into the system modeling phases and task modeling phases to make explicit particular erroneous behaviour.

To account for the many types of human error in the model based design of a safety-critical interactive system, we have produced a skill-based human error reference table (HERT), which contains a number of types of cognitive human errors from the literature. These can be systematically applied to interface applications, task models or system models to verify whether particular errors could potentially occur at various phases of interaction. When adding such information to the models, their size significantly increases. For this reason, we have proposed model patterns for error tolerance as a means of dealing with the explosion of models.

We use these different kinds of information (human errors, previous known technical and human-related erroneous behaviour from incident and accident reports) to inform the formal specification of the system modeling using ICOs. This allows us to produce a system model containing hazardous behaviour, showing that such situations have been accounted for. We can then adapt the system model such that a particular hazard is unreachable (i.e a hazardous place cannot contain a token) by human or system events by making modifications to the ICO model.

Furthermore, we improve the system model by informing it with barriers. The barriers considered in this thesis can be technical, human or socio-technical. They are identified using safety cases, incident and accident investigation techniques and human-computer interaction hazard analysis techniques. The barriers are classified, according to Hollnagel (Hollnagel 2004) and Leveson's (Leveson 1991) barrier classification. Those relating directly to the system are modeled, also using the ICO notation and plugged into the system model.

Because we use a dialect of Petri nets to model the system, we are able to perform powerful mathematical analyses to verify system properties. Since we are particularly interested in HCI, we employ marking graph analysis on our system models to identify and reason about scenarios.

Applicability of the framework

We have applied all principle phases of the generic integrated modeling framework to a rudimentary ATM Cash machine example throughout the Chapters 2 to 7 when introducing the concepts of the phases and then to two real life case studies; a fatal mining accident and an interactive cockpit software application for which no accident has yet been directly related (that we know of). So far, only parts of the framework have been applied to the two case studies. It would be interesting to apply the whole framework to an individual case study.

Usability of the framework

To date, we are the only people to have applied the framework on the two case studies presented and cash machine example. To validate the usability of the generic integrated modelling framework, including its proposed notations, tools and methods, would require use by other people. Furthermore, the framework should be applied to additional domains and contexts. We have argued that the framework can also be applied to non-safety critical interactive systems by not applying parts of the framework that concern safety. We believe however, that accounting for erroneous human behaviour can still bring benefits to the design of a non-safety critical system. Other domains of application could include gaming, which requires usability and reliability, and in some cases safety when gamers are addicted to playing for long hours!

Cost Benefits

We have discussed in the general conclusion of part two the issues of costs/benefits of our approach. The fact that we use models and notations to support our approach introduces additional costs to the development process. We argue that though the cost of development is higher than in classical development processes, because of the additional user centred design approaches, error analyses, barrier analyses and incident and accident investigation analyses, the benefits will be visible during actual system use. That is to say, because the system will be more reliable, safe and usable, less problems are likely to occur resulting in savings on training, bug patches, system crashes, loss of service etc. This assumption, that our framework is beneficial in terms of system use has not been proven. Furthermore one cannot guarantee that the system will be perfectly safe for example which is why industrial standards adhere to words such as “acceptable safe”. It would be useful and we should have developed an experiment to validate or invalidate such claims but we believe this is beyond the work of this PhD. Indeed, this would require multiple case studies and a lot of resources unfortunately not compatible with a PhD project.

Finalising the framework

Not all of the results of this project were positive in terms of the specific techniques being proposed. The task and system modelling notations used did not provide adequate means of documenting the more detailed psychological and managerial causes of failures, incidents and accidents, though these issues were out of the scope of this thesis. Such managerial, psychological, contextual factors should be considered in the model-based design of safety-critical interactive systems and we propose this to be a useful path of further work.

The complexity of design in the field of safety critical interactive systems clearly requires tool support for the creation, edition; formalisation; simulation, validation; verification of models and information, ability to check for inconsistencies; means for sharing and embedding data; cross-checking of hybrid models etc. To date, tools exist for the support of individual models, CTTe (Paterno 1999) for supporting various activities around task modelling (edition, simulation, verification ...), Petshop (Navarre et al. 2001b) for supporting various activities around system modelling. Despite some preliminary work about interaction (Navarre et al. 2001a) integration needs are still to be addressed. In this thesis we used these two existing tools for the editing and simulation of task and system models. The development of an integrated tool suite, capable of modelling all aspects presented in this thesis, including model patterns for error tolerance and model verification and validation is certainly an area of future research.

Furthermore, we have only considered skill-based errors in all of our human erroneous behaviour analyses. Incorporating rule-based and knowledge-based errors is a further field of work. Obtaining and understanding of user knowledge is difficult and time consuming, making an analysis of rule-based or knowledge-based errors difficult, though we are aware from the literature they exist.

We believe that our framework, can improve the safety, reliability and usability of systems.

Personal Publications

- Bastide, B and Basnyat, S. (2006) Error Patterns: Systematic Investigation of Deviations in Task Models. Task Models Diagrams for User Interface Design. (TAMODIA'2006) Hasselt, Belgium, October 23-24 2006.
- Taneva, S, Palanque, P, Basnyat, S, Winckler, M, Law, E. Clinical Application Design: Task Modeling with Failure in Mind. 11th World Congress on Internet in Medicine. MedNet 2006. Poster Presentation. Toronto, Canada, October 15-18 2006.
- Taneva S., Palanque P., Basnyat S., Winckler M., Law E. Analysis of Communication Breakdowns for eHealth Systems Design. 6th Nordic Conference on eHealth and Telemedicine, Helsinki, Finland, August 31 - September 1 2006.
- Basnyat, S., Schupp, B., Palanque, P., Wright, P. (2006) Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design. Special Issue of Safety Science Journal.
- Basnyat, S (2006) A multi-perspective approach for the design of error-tolerant socio-technical safety-critical interactive systems. Resilience for Survivability in Information Society Technologies (ResIST) European Network of Excellence Student Seminar. 5-7 September 2006, Centro Studi "I Cappuccini", Italy.
- Barboni, E., Navarre, D., Palanque, P., Basnyat, S. (2006) Addressing Issues Raised by the Exploitation of Formal Specification Techniques for Interactive Cockpit Applications HCI-Aero International Conference on Human-Computer Interaction in Aeronautics 20-22 September 2006 Seattle, USA.
- Basnyat, S (2006) Une approche multi perspective pour la conception de systèmes interactifs critiques. Rencontres doctorales IHM 2006 la 18ème Conférence Francophone sur l'Interaction Homme-Machine, 18-21 April 2006, Montreal, Canada
- Basnyat, S., Palanque, P. (2006) A Barrier-based Approach for the Design of Safety Critical Interactive Application. ESREL 2006 Safety and Reliability for Managing Risk. Safety and Reliability Conference. Balkema (Taylor & Francis) 18-22 September 2006, Estoril, Portugal.
- Schupp, B., Basnyat, S., Palanque, P., Wright, P. (2006) A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems 9th International Symposium of the ISSA Research Section Design process and human factors integration: Optimising company performances 1-3 March 2006 Nice, France
- Basnyat, S., Chozos, N., Palanque, P. (2006) Multidisciplinary perspective on accident investigation. special edition of Elsevier's Reliability Engineering and System Safety journal. Complexity in Design and Engineering Workshop Journal Special Edition, Elsevier.
- Basnyat, S., Bastide, R & Palanque, P. (2005) Extending the Boundaries of Model-Based Development to Account for Errors. Model Driven Development of Advanced User Interfaces. Workshop organised at MoDELS 2005, ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences) October 2, 2005
- Basnyat, S., Chozos, N., Johnson, C., Palanque, P. (2005) Redesigning an Interactive Safety-Critical System to Prevent an Accident from Reoccurring. 24th European Annual Conference on Human Decision Making and Manual Control. (EAM) Organised by the Institute of Communication and Computer Systems. 17-19 October 2005. Athens, Greece.
- Basnyat, S., Chozos, N., Johnson, C., Palanque, P. (2005) Incident and Accident Investigation Techniques to Inform Model-Based Design of Safety-Critical Interactive Systems. 12th International workshop on Design, Specification and Verification of Interactive Systems. 13-15 July 2005. Newcastle upon Tyne, England.
- Basnyat, S (2005) Task Patterns For Taking Into Account In An Efficient And Systematic Way Both Standard And Erroneous User Behaviour. LAAS Ecole Doctorales Systems Congress 2005.
- Basnyat, S., Navarre, D., Palanque, P., (2005) Complexity of Design in Safety Critical Interactive Systems: Gathering, Refining, Formalizing Multi-Type and Multi-Source Information while Ensuring

Bibliography

Consistency, Reliability, Efficiency and Error-Tolerance. 2nd Workshop on Complexity in Design and Engineering. Glasgow, Scotland. UK 10th-12th March 2005. Pages 45-58. GIST Technical Report G2005-1.

Basnyat, S, Palanque, P. (2005) A Task Pattern Approach to Incorporate User Deviation in Task Models. Proceedings of the first ADVISES Young Researchers Workshop Hans H.K. Andersen and Asmatullah Nayebkheil (eds.) Risø-R-1516(EN) (2005) 30 p. Electronic format only Liege, Belgium ADVISES meeting. January 4th - 7th 2005. Riso Technical Report.

Basnyat, S. (2004) Erreur humaine, modèles de tâches et description formelle pour la conception et l'évaluation des systèmes critiques et tolérant aux erreurs. Secondes Rencontres Jeunes Chercheurs en Interaction Homme-Machine. Lacanau, France, Octobre, 20-22, 2004.

Palanque, P., Basnyat, S. (2004) Task Patterns for Taking into Account in an Efficient and Systematic Way Both Standard and Abnormal User Behaviour. IFIP 13.5 Working Conference on Human Error, Safety and Systems Development Toulouse (HESSD) Toulouse, France, August 22-27, 2004.

Bibliography

- Accot, J, Chatty, S, and Palanque, P. A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems. Bodard, F and Vanderdonckt, J. The Third Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSVIS 1996). 92-104. 1996. Springer Verlag.
- Adduci, R. J, Hathaway, W. T, and Meadow, L. J. Technical Report: Hazard Analysis Guidelines for Transit Projects. U. S. Department of Transportation. January 2000.
- Albrechtsen, H, Andersen, H. H. K, Břdker, S, and Pejtersen, A. M. Affordances in Activity Theory and Cognitive Systems Engineering. Risř National Laboratory Riso-R-1287(EN). 37.
- Alexander, C., Ishikawa, S., & Silverstein, M. (1977) *A Pattern Language: Towns, buildings, construction* (Center for Environmental Structure Series), Oxford University Press, New York.
- Alexander, H. *Formally-Based tools and techniques for Human-Computer Dialogues*. Ellis Horwood limited, Chichester. 1987.
- Amalberti, R. *La conduite des systřmes ř risques*. (1996).
- Andrews, W. S, Anguiano, J. Y, Palmer, B. B, and Skrabak, R. A. *Exploding Vessels Under Pressure Accident*. United States Department of Labor Mine Safety and Health Administration. Mine I.D. No. 03-00256. 2002.
- Annett, K. & Duncan, J. (1967) *Task Analysis and Training Design*. *Journal of Occupational Psychology*, 41, 211-221.
- ANSI/IEEE. *Standard Glossary of Software Engineering Terminology STD-729-1991*, ANSI/IEEE. 1991.
- ARINC 661 . *Cockpit Display System Interfaces to User Systems*. Arinc Specification 661. 2002.
- ARINC 661-2. *Cockpit Display System Interfaces to User Systems*. Arinc Specification 661-2. 2005.
- Avizienis, A, Laprie, J. C, Randall, B, and Landwehr, C. *Basic Concepts and Taxonomy of Dependable and Secure Computing*. *IEEE Transactions On Dependable And Secure Computing*. 1, 11-33. January - March 2004.
- Avizienis, A., Laprie, J. C, and Randell, B. *Dependability and its Threats: A Taxonomy*. 18th IFIP World Computer Congress Presentation Available from <http://www2.laas.fr/IFIPWG/Top3/02-Laprie.pdf>. 2004.
- Baber, C. & Stanton, N. (2004) *Task Analysis for Error Identification*. IN: *The Handbook of Task Analysis for Human-Computer Interaction* (eds D. Diaper & N. Stanton), pp. 367-379. Lawrence Erlbaum Associates, New Jersey.
- Barboni, E, Conversy, S, Navarre, D, and Palanque, P. *Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification*. *Proceedings of the 13th conference on Design Specification and Verification of Interactive Systems (DSVIS 2006)*. 2006. *Lecture Notes in Computer Science*, Springer Verlag.
- Barboni, E, Navarre, D, Palanque, P, and Basnyat, S. *Addressing Issues Raised by the Exploitation of Formal Specification Techniques for Interactive Cockpit Applications*. *International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero)*. 2006.
- Barnard, P and May, J. *Interactions with Advanced Graphical Interfaces and the Deployment of Latent human Knowledge*. *Interactive Systems: Design, Specification and Verification*. DSVIS 1994 pp15-49. 1994.

- Basnyat, S, Bernhaupt, R, Boring, R, Palanque, P, and Johnson, C. Testing Interactive Software: A Challenge for Usability and Reliability. Special Interest Group CHI 2006 (SIGCHI). Available from: <http://liihs.irit.fr/palanque/SIGchi2006.html>. 2006.
- Basnyat, S, Chozos, N, Johnson, C, and Palanque, P. Incident and Accident Investigation Techniques to Inform Model-Based Design of Safety-Critical Interactive Systems. M. Harrison and S. Gilroy. 12th International Workshop on Design, Specification and Verification of Interactive Systems. 2005. Elsevier.
- Basnyat, S, Chozos, N, Johnson, C, and Palanque, P. Redesigning an Interactive Safety-Critical System to Prevent an Accident from Reoccurring. 24th European Annual Conference on Human Decision Making and Manual Control. (EAM) Organised by the Institute of Communication and Computer Systems. 2005.
- Basnyat, S., Chozos, N., & Palanque, P. (2005c) Multidisciplinary perspective on accident investigation. Special Edition of Elsevier's Reliability Engineering and System Safety Journal.
- Basnyat, S and Palanque, P. A Barrier-based Approach for the Design of Safety Critical Interactive Application. ESREL 2006 Safety and Reliability for Managing Risk. Safety and Reliability Conference. 2006. Balkema (Taylor & Francis) .
- Basnyat, S, Schupp, B, Palanque, P, and Wright, P. Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design. Special Issue of Safety Science Journal. Submitted.
- Bass, L, Little, R, Pellegrino, R, Reed, S, Seacord, R, Sheppard, S, and Szezur, M. R. The Arch Model: Seeheim Revisited. User Interface Developers' Workshop, version 1.0. 1991.
- Bass, L, Little, R, Pellegrino, R, Reed, S, Seacord, R, Sheppard, S, and Szezur, M. R. The Arch Model: Seeheim Revisited. User Interface Developers' Workshop. Version 1.0 . 1991.
- Bastide, R and Basnyat, S. Task Models Diagrams for User Interface Design. (TAMODIA'2006) . 2006.
- Bastide, R, Navarre, D, Palanque, P, and Schyn, A. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04). 2004.
- Bastide, R and Palanque, P. UML for Interactive Systems: What is Missing. Workshop on Software Engineering in Human-Computer Interaction INTERACT '03: IFIP TC13 International Conference on Human-Computer Interaction. 2003. IOS Press.
- Baumeister, L. K, John, B. E, and Byrne, M D. A Comparison of Tools for Building GOMS Models Tools for Design. ACM Conference on human Factors in Computing Systems CHI 2000. 502-509. 2000. ACM Press.
- Beaudouin-Lafon, M. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. Computer-Human Interaction 2000 (CHI 2000). 446-453 . 2000.
- Blamire, J. Genotype and Phenotype Definition. Science at a Distance available from <http://www.brooklyn.cuny.edu/bc/ahp/BioInfo/GP/Definition.html>. 2000.
- Blandford, A, Butterworth, R, and Curzon, P. PUMA Footprints: linking theory and craft skill in usability evaluation. PUMA working paper WP26. See the world wide web pages at <http://www.cs.mdx.ac.uk/puma/>. 2000.
- Blandford, A and Connell, I. Designing to avoid post-completion errors. PUMA Working Paper WP33. 2000.
- Blandford, A and Connell, I. Ontological Sketch Modelling (OSM): Concept-based Usability Analysis. 9th IFIP TC13 International Conference on Human-Computer Interaction INTERACT 2003. 1021-1022. 2003. IOS Press.
- Blandford, A and Good, J. Programmable User Models - Exploring Knowledge Transfer between Devices.

PUMA Working Paper. 1997. WP5.

Bloomfield, R., Bishop, P, Jones, C, and Froome, P. ASCAD- Adelard Safety Case Development Manual, Adelard. 1998.
 . Available online at: <http://www.adelard.co.uk/resources/papers/pdf/dsn2004v10.PDF> (Accessed 12 December 2004). 2004.

Boehm, B.W. (1988) A spiral model of software development and enhancement.

Booch, G, Rumbaugh, J, and Jacobson, I. The Unified Modelling Language User Guide. 1999. Addison-Wesley.

Botting, R and Johnson, C. W. A Formal and Structured Approach to the Use of Task Analysis in Accident Modelling. *International Journal of Human Computer Systems*. 49, 223-244. 1998.

Bowen, J. P. Formal Methods in Safety-Critical Standards. In *Proc. Software Engineering Standards Symposium (SESS'93)*. 168-177. 1993. IEEE Computer Society Press.

Bowen, J. P, Butler, R. W., Dill, D. L., Glass, R. L., Gries, D., Hall, A, Hinchey, M. G., Holloway, C. M., Jackson, D, Jones, C. B., Lutz, M. J., Parnas, D. L., Rushby, J, Wing, J, Zave, P, and Saiedian, H. An invitation to formal methods. *IEEE Computer*, 29(4):1630, April 1996. 1996.

Bowen, J. P and Hinchey, M. G. Ten Commandments of Formal Methods. *IEEE Computer*. 28(4):56--63. 1995. University of Cambridge Computer Laboratory.

Bredereke, J and Lankenau, A. A Rigorous View of Mode Confusion. *Proceedings of the 21st International Conference on Computer Safety, Reliability and Security (SafeComp 2002)*. 2002. Springer Verlag .

Breedvelt, I, Paterno, F, and Sereriins, C. Reusable Structures in Task Models. *Proceedings Design, Specification, Verification of Interactive Systems*. 251-265. 1997. Springer Verlag.

Burns, C. P. Analysing Accident Reports Using Structured and Formal Methods. PhD Thesis. University of Glasgow. Department of Computing Science. <http://www.dcs.gla.ac.uk/~burnsc/thesis.pdf>. 2000.

Buschmann, R, Meunier, H, Rohnert, P, Sommerlad, and M. Stal. *Pattern Oriented Software Architecture*. 1996. John Wiley & Sons, Inc.

Busse, D. Linking Human Error Analysis and the Validation of Safety Recommendations . *Proceedings of the 18th European Annual Conference on Human Decision Making and Manual Control*. 1999. Loughborough University, UK Jim Alty (ed.).

Buxton, W and Myers, B. A study of two-handed input. *Proceedings of Computer Human Interaction (CHI '86)*, 321-326. 1986.

Buys, J. R and Clark, J. L. Events and Causal Factors Charting Technical Report. available at www.osti.gov/energycitations/product.biblio.jsp?osti_id=5651580. 2001.

Byrne, M. D and Bovair, S. A working memory model of a common procedural error. *Cognitive Science* 21, 31-61. 1997.

Campos, J and Harrison, M. D. Formally verifying interactive systems: A review. Harrison, M. D and Torres, J. C. 4th Eurographics workshop on Design, Specification and Verification of Interactive Systems DSVIS '97. 109-124. 1997. Springer Computer Science, Springer Verlag.

Card, S. K, Moran, T. P, and Newell, A. *The Psychology of Human-Comuter Interaction*. 1983. New Jersey, Lawrence Erlbaum Associates.

Carlsen, N, Christensen, N, and Tucker, H. An extended event model for specifying user interfaces. In *proceeding of the 4th IFIP Working conference on user interfaces*. 1989.

- Cebulla, M. Modeling concepts for safety-related requirements in sociotechnical systems. Proc. of SAFECOMP'04. Lecture Notes in Computer Science. Volume 3219, Berlin: Springer. 2004.
- Center for Resilience. Concepts Centre for Resilience, What is Resilience? Available from: <http://www.resilience.osu.edu/concepts.html> (last accessed August 2006). 2006.
- Cho, S M, Hong, H. S, and Cha, S. D. Safety Analysis Using Coloured Petri Nets. Third Asia-Pacific Software Engineering Conference (APSEC'96). p. 176. 1996.
- Clarkson, M. B. E. A stakeholder framework for analyzing and evaluating corporate social performance. *Academy of Management Review*, 20: 39-48. 1995.
- Cockton, G, Woolrych, A, Hall, L, and Hindmarch, M. Changing Analysts' Tunes: The Surprising Impact of a New Instrument for Usability Inspection Method Assessment. Palanque, P., Johnson, P, and O'Neill, E. *People and Computers XVII: Designing for Society (Proceedings of HCI 2003)*. 145-162. 2003. Springer-Verlag.
- Cook, R. I and Nemeth, C. Taking Things in One's Stride: Cognitive Features of Two Resilient Performances (Chapter 13). Hollnagel, E, Woods, D, and Leveson, N. *Resilience Engineering: Concepts and Precepts*. 2006.
- Cukic, B, Ammar, H. H, and Lateef, K. Identifying high-risk scenarios of complex systems using input domain partitioning. In: *Proc. 9th Int. Symp. on Software Reliability Engineering*. pages 164-173. 1998.
- Curzon, P and Blandford, A. Formally Justifying User-centred Design Rules: a Case Study on Post-completion Errors. *Proceedings of Integrated Formal Methods, Lecture Notes in Computer Science*, Springer. 2004.
- Czarnecki, K and Eisenecker U. W. *Generative Programming Methods, Tools, and Applications*. 2000. Addison-Wesley, 2000.
- D. Hazel, P. Strooper, and O. Traynor. Possum: An animator for the SUM specification language. W. Wong and K. Leung. *Asia Pacific Software Engineering Conference (APSEC '97)*. 51, pages 42-51. 1997. IEEE Computer Society, 1997.
- Dammag, H and Nissanke, N. Safecharts for specifying and designing safety critical systems. *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems* October 1999. pages 78-87. 1999.
- Dammag, H and Nissanke, N. Safecharts for Specifying and Designing Safety Critical Systems. *Proc. 18th IEEE Symp. Reliable Distributed Systems*. 1999.
- Dammag, H and Nissanke, N. A Mathematical Framework for Safecharts. *Proceedings of the 5th International Conference of Formal Engineering Methods*. pp: 620-640. 2003.
- Daouk, M and Leveson, N G. An Approach to Human-Centered Design. *Workshop on human error and system development*. <http://sunnyday.mit.edu/papers.html> . 2001.
- de Haan, G. MUSE - a Successful Case-Study of Engineering Human Factors Design Methods? Report on a SigCHI.nl presentation by John Long. SigCHI.nl website: <http://www.sigchi.nl/page.asp?folder=102&page=102>. 1998.
- Dearden, A. M and Harrison, M. D. Formalising human error resistance and human error tolerance. *Proceedings of the Fifth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace*. 1995. EURISCO.
- Degani, A. *Modeling Human-Machine Systems: On Modes, Error And Patterns of Interaction*. PhD Thesis. Georgia Institute of Technology. December 1996.
- Degani, A. *Taming HAL: Designing user interfaces beyond 2001*. 312 pages. 2004. Palgrave Macmillan.

- Degani, A and Kirlik, A. Modes in Human-Automation Interaction: Initial Observations about a Modeling Approach. Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC). 1995.
- Degani, A, Shafto, M, and Kirlik, A. Modes in Human-Machine Systems, Review, Classification and Application. *International Journal of Aviation Psychology*. 9(2), 125-138. 1999.
- Dix, A, Finlay, J, Abowd, G, and Beale, R. *Human-computer Interaction*. Prentice Hall, Second Edition, Prentice Hall Europe. 1998.
- Dix, A and Runciman, C. Abstract models of interactive systems. Ed. P. J. & S. Cook. *People and Computers: Designing the Interface*, Cambridge University Press. pp. 13-22. 1985.
- Dix, A. J. *Formal methods for interactive systems*. London: Academic Press. 1991.
- Dryden Handbook. Code S System Safety Handbook. NASA Dryden Flight Research Center Handbook DHB-S-001 Baseline 03/02/99 . 1999.
- Earthy, J. V. Full Hazops of Programmable Electronic Systems. Contesse Project. Report No. 7266-9-0-0.3. 1995. Lloyds Register, Croydon, U.K.
- Ericson, C. A. *Software Safety in a Nutshell*. The Boeing Company; Seattle, Washington. Available from http://www.dcs.gla.ac.uk/~johnson/teaching/safety/reports/Clif_Ericson1.htm. 1999.
- Ernst & Young. *Global Information Security Survey 2003*. Assurance And Advisory Business Services. Score Retrieval File No.Ff0224 . 2003.
- Ernst & Young. *Global Information Security Survey 2004*. Assurance And Advisory Business Services. Score Retrieval File No.FF0231 . 2003.
- Esteban, O, Chatty, S, and Palanque, P. Whizz'Ed: a visual environment for building highly interactive interfaces. Proceedings of the Interact'95 conference, 121-126. 1995.
- F. Paternò, C. Santoro, and S. Tahmassebi. Formal Models for Cooperative Tasks: Concepts and an Application for En-Route Air Traffic Control. Proceedings DSV-IS'98. pp.71-86. 1998. Springer Verlag.
- Faerber R, Vogl T, and Hartley D. Advanced Graphical User Interface for Next Generation Flight Management Systems. In proceedings of HCI Aero 2000, pp. 107-112. In proceedings of HCI Aero 2000, pp. 107-112. 2000.
- Falla, M. Advances in safety-critical systems. Results and achievements from the DTI/EPSRC R&D programme in safety critical systems. June 1997.
- Fenton, N. E and Neil, M. Software Metrics and Risk. Proceedings of the 2nd European Software Measurement Conference (FESMA'99), TI-KVIV. 1999.
- Fields, B, Paterno, F, and Santoro, C. Analysing User Deviations in Interactive Safety-Critical Applications. Design, Specification and Verification of Interactive Systems (DSVIS). 189-204. 1999. Springer-Verlag.
- Fiksel, J. Designing Resilient, Sustainable Systems. *Environmental Science & Technology*. December 2003.
- Fitts, P. M. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology* 47 pp. 381-91. 1954.
- Fujita, Y. Nature of Changes in Systems. Hollnagel, E, Woods, D. D, and Leveson, N. *Resilience Engineering: Concepts and Precepts*. 2006. Ashgate.
- Galliers, J, Sutcliffe, A, and Minocha, S. An impact analysis method for safety-critical user interface design. *ACM Transactions on Computer-Human Interaction (TOCHI)* Special issue on interface design for safety-

critical interactive systems: when there is no room for user error. Volume 6 , Issue 4 , Pages: 341-369 . 1999.

Gamma, E, Helm, R, Johnson, R, and Vlissides, J. Design Patterns: Elements of Object-oriented Software. 1995. Addison-Wesley.

Garrett, C and Apostolakis, G. Context and Software Safety Assessment. HESSD 1998. 1998.

Genrich, H. J. Predicate/Transitions Nets, High-Levels Petri-Nets: Theory and Application. K. Jensen & G. Rozenberg (eds.). High-Level Petri: Theory and Applications. 3-43. 1991. Springer Verlag.

Gerhardt-Powals , J. (1996) Cognitive engineering principles for enhancing human - computer performance. International Journal of Human-Computer Interaction, 8(2), 189-211.

Gould, J and Lewis, C. Designing for usability: Key Principles and What Designers Think. Communications of the ACM 28, no. 3 (March), ACM, 360 - 411. 1975.

Green, M. Report on Dialogue Specification Tools. G. Pfaff. User Interface Management Systems. 9-20. 1985. New York: Springer-Verlag.

Green, M. Human Error in Medicine. Available from: http://www.expertlaw.com/library/malpractice/medical_error.html (last accessed 16/11/2006). 2003.

Green, M. A. A survey of three dialogue models. ACM Transaction on graphics. Vol. 5, n° 3, July 1986. pp. 244-275. 1986.

Greenwell, W. S, Strunck, E. A, and Knight, J. C . Failure Analysis and the Safety- Case Lifecycle. 6th International Working Conference on Human Error, Safety and System Development (HESSD 2004), IFIP. p. 163-176. 2004.

Gunderson, L and Protchard, L. Jr. Resilience and the Behavior of Large-Scale Systems. 2002. Washington, DC, Island Press.

Haiko, S, Lehto, E, and Virvaldo, T. Modelling of Water Hammer Phenomenon- Based Pressure Intensifier. [http://www.callisto.si.usherb.ca/~fluo2000/PDF/F1_078.pdf] Last accessed 23/2/05 . 2000.

Hale, A and Heijer, T. Defining Resilience. Hollnagel, E, Woods, D, and Leveson, N. Chapter 3: Resilience Engineering Concepts and Precepts. 2006.

Harel, D. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming. 8, pp.231-274. 1987. North-Holland.

Harrison, M and Dix, A. State Model of Direct Manipulation in Interactive Systems. In Formal Methods in Human-Computer Interaction. Edited by M. Harrison and H. Thimbleby. Cambridge Series on HCI. P.129. 1990.

Harrison, M. D and Thimbleby H.W. Formal methods in Human-Computer Interaction. 1990. Cambridge University Press.

Hartson, R. & Gray, P. (1992) Temporal Aspects of Tasks in the User Action Through Product and Process, John Wiley, New York.

Heiner, M and Heisel, M. Modeling safety-critical systems with Z and Petri nets. Lecture Notes in Computer Science, Vol. 1698: Computer Safety, Reliability and Security, 1999. 1999. Springer-Verlag.

Heinrich, H. W, Petersen, D, and Roos, N. Industrial accident prevention. 1980. McGraw-Hill: New York.

Hendrick, K and Benner, L. Jr. Investigating accidents with STEP. 1987. Marcel Dekker.

- Heymann, M and Degani, A. Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm. Paper accepted for publication; expected date of publication, Spring 2007. 2007.
- Hill, J. C. Resolving Complexity in Accident Texts Through Graphical Notations and Hypertext . PhD Thesis. University of York Department of Computer Science. March 2001.
- Hinchey, M. G and Bowen, J. P. Applications of formal methods FAQ. In M.G. Hinchey and J.P. Bowen, editors. Applications of Formal Methods, International Series in Computer Science. chapter 1, pages 1–15. 1995. Prentice-Hall.
- Hix, D. & Hartson, H.R. (1993) Developing User Interfaces: ensuring usability through product and process. John Wiley and Sons, New York.
- Hoare, C. A. R. Communicating Sequential Processes. International Series in Computer Science. Prentice-Hall International. 1985.
- Holling, C. S. Resilience and stability of ecological systems. Annual Review of Ecology and Systematics. 4: 1-23. 1973.
- Hollnagel, E. (1991) The Phenotype of Erroneous Actions: Implications for HCI Design. IN: Human-Computer Interaction in Complex Systems (eds G. Weir & J. Alty), pp. 1-32. Academic Press.
- Hollnagel, E. Human Reliability Analysis Context and Control. Academic Press, Inc., New York NY. 1993.
- Hollnagel, E. The phenotype of erroneous actions. International Journal of Man-Machine Studies. 39. 1993.
- Hollnagel, E. Accident Analysis and Barrier Functions. Project TRAIN Report Version 1.0. 1999.
- Hollnagel, E. (1999b) Accidents and barriers. IN: Proceedings of Lex Valenciennes (eds J.M. Hoc, P. Millot, E. Hollnagel, & P.C. Cacciabue), pp. 175-182. Presses Universitaires de Valenciennes.
- Hollnagel, E. The Elusiveness of "Human Error". Report based on E. & Amalberti, R. The Emperor's New Clothes, or whatever happened to "human error"? Invited keynote presentation at 4th International Workshop on Human Error, Safety and System Development. Linköping, June 11-12, 2001. Available from: http://www.ida.liu.se/~eriho/HumanError_M.htm. 2001.
- Hollnagel, E. Barriers and Accident Prevention. 2004. Ashgate.
- Hollnagel, E. Achieving System Safety by Resilience Engineering. Keynote presentation. The 1st Institute of Engineering and Technology International Conference on Systems Safety. p184-195. 2006.
- Hollnagel, E. Resilience - the Challenge of the Unstable (Chapter 1). Hollnagel, E, Woods, D. D, and Leveson, N. Resilience Engineering: Concepts and Precepts. 2006. Ashgate.
- Hollnagel, E, Woods, D, and Leveson, N. About Resilience Engineering International Symposium on Resilience Engineering, From Symposium Material. 2004.
- Hollnagel, E, Woods, D, and Leveson, N. Resilience Engineering Concepts and Precepts. 2006. Ashgate.
- Hollnagel, R and Woods, D. Epilogue: Resilience Engineering Precepts from of Resilience Engineering: Concepts and Precepts, Hollnagel, E., Woods D and Leveson, N (editors) 0 7546 4641 6. Ashgate. 2006.
- Hollnagel, R and Woods, D. Prologue: Resilience Engineering Precepts from of Resilience Engineering: Concepts and Precepts, Hollnagel, E., Woods D and Leveson, N (editors) 0 7546 4641 6. Ashgate. 2006.
- Holloway, C. M. Johnson C. W. Why System Safety Professionals Should Read Accident Reports. The 1st IET International Conference on System Safety. 2006.

Hourizi, R. and Johnson P. Unmasking Mode Error: A New Application of Task Knowledge Principles to The Knowledge Gaps in Cockpit Design. *Interact'01, The Eighth IFIP Conference on Human Computer Interaction*. 2001.

Hsiung, P. A and Lin, Y. H. Modeling and Verification of Safety-Critical Systems Using Safecharts. *International Conference on Formal Techniques for Networked and Distributed Systems FORTE 2005*: 290-304. 2005.

Hughes, D and Dornheim, M. Automated Cockpits: Who's in Charge? *Aviation Week & Space Technology*. 1995.

Hussey, A. HAZOP Analysis of Formal Models of Safety-Critical Interactive Systems. *Computer Safety, Reliability and Security, 19th International Conference, SAFECOMP 2000*. pp. 250 -- 263. 2000. *Lecture Notes in Computer Science 1943 Springer*.

International Electrotechnical Commission IEC 61508. International Standard for functional safety of electrical/electronic/programmable electronic safety-related systems. Part 4. Available via <http://www.iec.ch/functionalsafety>. 2003.

International Ergonomics Association (IEA). Definition of Ergonomics. <http://www.iea.cc/about/fundamentals.cfm>. 2000.

Isaksen, U, Bowen, J. P, and Nissanke, N. System and Software Safety in Critical Systems. Technical Report RUCS/97/TR/062/A, Department of Computer Science, The University of Reading, UK, 1997. 1996.

ISO 13407. Human-centered design processes for interactive system. International Organization for Standardization, Geneva. 1999.

ISO/IS 8807. ISO - Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on temporal Ordering of Observational Behaviour. . ISO/IS 8807, ISO Central Secretariat. 1988.

Jacob, R. J. K. A Specification Language for Direct-Manipulation User Interfaces. *ACM Trans. Graphics*283-317. 1986.

Jeffcott, M and Johnson, C. W. The Use of a Formalised Risk Model in NHS Information System Development. *Cognition Technology and Work*. 4, 120-136. 2002.

Jeffcott, M. A and Johnson, C. W. The Use of a Formalised Risk Model in the UK National Health Service Information System Development. *Cognition Technology and Work Journal*. 4, 120-136. 2002.

Johnson, C. On the over emphasis of human error as a cause of aviation accidents: systemic failures and human error in US NTSB and Canadian TSB aviation reports 1996-2003. Submitted to *Ergonomics*. 2006 (in review) .

Johnson C.W. A probabilistic logic for the development of safety critical interactive systems. *International Journal of Man-Machine studies*, 38, 2, pp. 333-351. 1992.

Johnson, C. W. Social Aspects of Computing Lecture Notes. Available from <http://www.dcs.gla.ac.uk/~johnson/teaching/social/course.html>. 1997.

Johnson, C. W. Lecture 1: Definitions of safety and the Ariane 5 Case Study. <http://www.dcs.gla.ac.uk/~johnson/teaching/safety/default.html>. 1999.

Johnson, C. W. Principles for the Use of Formal Methods in Accident Investigations. Summary Report EPSRC Grant GR/K55042. Available from http://www.dcs.gla.ac.uk/~johnson/papers/GR_K55042_summary.htm. 1999.

Johnson, C. W. Visualizing the Relationship between Human Error and Organizational Failure. J. Dixon.

- Proceedings of the 17th International Systems Safety Conference. 101-110. 1999. The Systems Safety Society.
- Johnson, C. W. Failure in Safety-Critical Systems. A Handbook of Accident and Incident Reporting. Available on-line at: <http://www.dcs.gla.ac.uk/johnson/book> October 2003. 2003. Glasgow, Scotland, University of Glasgow Press.
- Johnson, C. W. What are emergent properties and how do they affect the engineering of complex systems? Reliability Engineering and System Safety Journal. 91, 12, 1475-1481. 2006.
- Johnson, C. W and Holloway, C. M. The ESA/NASA SOHO Mission Interruption: Using the STAMP Accident Analysis Technique for a Software Related 'Mishap'. Software: Practice and Experience Journal. 33, 1177-1198. 2003.
- Johnson, C. W and Holloway, C. M. 'Systemic Failures' and 'Human Error' in Canadian TSB Aviation Accident Reports between 1996 and 2002. HCI in Aerospace 2004. 25-32. 2004. EURISCO, Toulouse, France.
- Johnson, C. W and Holloway, M. Questioning the Role of Requirements Engineering in the Causes of Safety-Critical Software Failures. Kelly, T. The First IET International Conference on System Safety. 352-360. 2006. Institute of Engineering and Technology.
- Johnson, C. W, McCarthy, J. C, and Wright, P. C. Using a Formal Language to Support Natural Language in Accident Report. Ergonomics Journal 36(6). 1995.
- Johnson, P. Human-Computer Interaction: Psychology, Task Analysis and Software Engineering. London: McGraw-Hill. 1992.
- Johnson, P and Johnson, H. Knowledge Analysis of Task: Task Analysis and Specification for Human-Computer Systems. Downton, A. Engineering the Human Computer Interface. 119144. 1989. Maidenhead, McGraw-Hill.
- Johnston, I. H. A, Butler, J. J, and Johnson, M. H. Success Rather than Failures as a Basis for Good Practice Guidance on Functional Safety. The 1st IET International Conference on System Safety. 2006.
- Johnston, W and Wildman, L. The Sum Reference Manual. Software Verification. Research Centre TR99-21, The University of Queensland, November. 1999.
- Jordan, B. Ethnographic Workplace Studies and CSCW. D. Shapiro, M.J. Tauber, and R. Traunmueller (eds). The design of computer supported cooperative work and groupware systems. 17-42. 1996.
- Karat, J. User Centered Design: Quality or Quackery? ACM/SIGCHI magazine, Interactions July+August. 1996.
- Karat, J. Evolving the Scope of User-Centered Design. Communications of the ACM, Vol. 40, No 7, pp. 33-38. 1997.
- Kelly, T and Weaver, R. The Goal Structuring Notation – A Safety Argument Notation. Proc. of Dependable Systems and Networks 2004. Workshop on Assurance Cases. 2004.
- Kelly, T. P. Arguing Safety - A Systematic Approach to Safety Case Management. DPhil Thesis. Department of Computer Science, University of York. 1998.
- Kelly, T. P. Managing Complex Safety Cases. Proceedings of 11th Safety Critical System Symposium (SSS'03). 2003. Springer-Verlag.
- Kieras, D. E and Polson, P. G. An approach to the formal analysis of user complexity. International Journal of Man-Machine Studies, 22, 365-394. 1985.
- Killam, B and Holland, B. Position Paper on the Suitability to Task of Automated Utilities for Testing Web

Accessibility Compliance [Electronic version]. The Usability SIG Newsletter: Usability Interface Accessibility and Usability: Partners in EffectiveDesign. 9[4]. April 2003.

Kjellén, U. Prevention of accidents through experience feedback. 2000. Taylor and Francis.

Koester, T. Human error in the maritime work domain. Proceedings (pp. 149-158): EAM 2001, 20th European Annual Conference on Human Decision Making and Manual Control. 2001.

Kontogiannis, T, Leopoulos, V, and Marmaras, N. A comparison of accident analysis techniques for safety-critical man-machine systems. *International Journal of Industrial Ergonomics*. 25, 327-347. 2000. Elsevier.

Kruchten, P. *The Rational Unified Process: An Introduction*. Second Edition ed. Boston, USA. 2000. Addison Wesley Longman Publishing.

Kumamoto, H and Henley, E. J. *Probabilistic Risk Assessment and Management for Engineers and Scientists*. Second Edition. 1996. New York, IEEE Press.

Lacaze, X. *La conception rationalisée pour les systemes interactifs*. PhD Thesis, University of Toulouse 1. 2005.

Lacaze, X, Palanque, P, Navarre, D, and Bastide, R. Performance Evaluation as a Tool for Quantitative Assessment of Complexity of Interactive Systems. DSV-IS'02 9th workshop on Design Specification and Verification of Interactive Systems. 2002. Springer, Lecture notes in Computer Science 2545, p. 208-222.

Ladkin, P and Loer, K. Why Because Analysis: Formal Reasoning About Incidents. Technical Report RVS-BK-98-01. 1998.

Laprie, J. C. Dependability of Computer Systems: from Concepts to Limits. in Proc. IFIP International Workshop on Dependable Computing and its Applications. pp. 108-126. 1998.

Laprie, J. C. Sûreté de fonctionnement informatique : un nécessaire changement d'échelle. Invited Talk at IHM 2005. <http://www.irit.fr/ihm2005/download/IHM2005-JCL.pdf>. 2005.

Laprie, J. C, Arlat, J, Blanquart, J P, Costes, A, Crouzet, Y, Deswarte, Y, Fabre, J. C, Guillermain, H, Kaâniche, M, Kanoun, K, Mazet, C, Powell, D, Rabéjac, C, and Thévenod, P. *Guide De La Sûreté De Fonctionnement*. 1995. Cépaduès-Editions, Toulouse, 1995.

Leathley, B. A. HAZOP approach to allocation of function in safety critical systems. ALLFN'97 Revisiting the Allocation of Function Issue. pp. 331-343. 1997. IEA Press.

Lee, A. Y, Polson, P. G, and Bailey, W. A. Learning and transfer of measurement tasks. 20, 115-120 . 1989. New York, NY, USA, ACM Press .

Lekberg, A. Different Approaches to Accident Investigation: How the Analyst Makes the Difference. Proceedings of the 15th International Systems Safety Conference. p173-193. 1997. Sterling, VA: International Systems Safety Society.

Leveson, N. Software Safety: Why, What, and How'. 18(2), 125 -- 163. 1986. ACM Computing Surveys .

Leveson, N. Evaluation of software safety. Proceedings of the 12th international conference on Software engineering. 223-224 . 1990.

Leveson, N. A Systems Model of Accidents . In J.H. Wiggins and S.Thomason (eds) . Proceedings of the 20th International Safety Conference. 476-486. 2002. International Systems Safety Society, Unionville, USA.

Leveson, N. Hazard Analysis . White Paper. 2003.

Leveson, N. Software Hazard Analysis . White Paper. 2003.

- Leveson, N. A New Accident Model for Engineering Safer Systems. *Safety Science*, 42: 237-270. 2004.
- Leveson, N, Alfaro, L, Alvarado, C, Brown, M., Hunt, E. B, Jaffe, M, Joslyn, S, Pinnel, D, Reese, J, Samarziya, J, Sandys, S, Shaw, A, and Zabinsky, Z. Demonstration of a Safety Analysis on a Complex System. Software Engineering Laboratory Workshop, NASA Goddard. December 1997.
- Leveson, N, Hunt, E. B, Jaffe, M, Joslyn, S, Rees, J, Shaw, A, Zabinsky, Z, Alfaro, L, Alvarado, C, Brown, M, Pinnel, D, Samarziya, J, Sandys, S, and Shafer, M. A demonstration of safety analysis of Air Traffic Control software: Final Report. September 1997. Available from <http://sunnyday.mit.edu/papers/dfw2.pdf>.
- Leveson, N.G. (1991) Software safety in embedded computer systems. *Communications of the ACM* archive. ACM Press, 34, 34-46.
- Leveson, N. G. *SAFWARE: System Safety and Computers*. 1995. Addison-Wesley.
- Leveson, N. G. A new accident model for engineering safer systems. *Safety Science Journal*. 42. April 2004.
- Leveson N.G, Cha S.S, and Shimeall T.J. Safety Verification of Ada Programs using Software Fault Trees. *IEEE Software*, 8(7). p. 48-59. 1991.
- Leveson, N. G, Pinnell, L. D, Sandys, S. D, Koga, S, and Reese, J. D. Analyzing Software Specifications for Mode Confusion Potential. Workshop on Human Error and System Development. 1997.
- Leveson, N. G and Stolzy, J. L. Analyzing software safety. *IEEE Transactions on Software Engineering*. 13, 92-103. 1987.
- Li, B. Integrating software into PRA (Probabilistic Risk Assessment). PhD Thesis. University of Maryland, College Park. 2004.
- Lim K.Y and Long J. *The MUSE Method for Usability Engineering*. Cambridge University Press. 1995.
- Lu, S, Paris, C, and Vander Linden, K. Toward the automatic construction of task models from object-oriented diagrams. Proceedings of the IFIP TC2/TC13 WG2.7/WG13.4 Seventh Working Conference on Engineering for Human-Computer Interaction. 169-189. 1999. Kluwer Academic Publishers.
- MacLean, A, Young, R. M, Bellotti, V. M. E, and Moran, T. P. Questions, Options, and Criteria: Elements of Design Space Analysis. In Moran, T. P, Carroll, J. M, and eds. *Design Rationale: Concepts, Techniques, and Use*. LEA. 1996.
- Mahemoff, M. J and Johnston, L. J. Principles for a Usability-Oriented Pattern Language. Calder, P and Thomas, B. *OZCHI '98 Proceedings*. 1998. 132-139. Los Alamitos, CA.
- Marrenbach J and Kraiss K-F. Advanced Flight Management System: A New Design and Evaluation Results. In proceedings of HCI Aero 2000, pp. 101-106. 2000.
- Massie, T. H and Salisbury, J. K. The PHANTOM Haptic Interface: A Device for Probing Virtual Objects. Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. 1994.
- Mayhew, D and Bias, R. *Cost-justifying usability*. Pages Academic Press. 1994.
- Mc Dermid, John and Ripkin, Knut. *Life cycle support in the ADA environment*. Cambridge University Press. 1984.
- Miller, C. A, Funk, H. B, Goldman, R. P, Meisner, J, and Wu, P. Implications of Adaptive vs. Adaptable UIs on Decision Making: Why "Automated Adaptiveness" is Not Always the Right Answer. In Proceedings of the 1st International Conference on Augmented Cognition. 2005.
- MOD. HAZOP Studies on Systems Containing Programmable Electronics. UK Ministry of Defence. Interim

Def Stan 00-58. 1996.

Mori, G, Paterno, F, and Santoro, C. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transaction on Software Engineering*. 28(8):797--813. September 2002. 2002.

Mori, G, Paterno, F, and Santoro, C. Design and development of multidevice user interface through multiple logical descriptions. *Transactions on Software Engineering*, 30(8), August 2004. 2004.

Mostia, B. Mistakes are inevitable. Understanding how and where they occur can minimize their frequency and impact. *Human Error in Instrumentation Systems (Part 1 of 2)*. *Control Magazine*. 2002.

Nardi, B. (1995) *Context and Consciousness: Activity Theory and Human Computer Interaction*, MIT Press, Cambridge MS..

Navarre, D. Contribution à l'ingénierie en Interaction Homme Machine - Une technique de description formelle et un environnement pour une modélisation et une exploitation synergiques des tâches et du système. PhD Thesis. University of Toulouse I. Defended July 2nd 2001. 2001.

Navarre, D, Dragicevic, P, Palanque, P, Schyne, A, and Bastide, R. Very-High-Fidelity Prototyping for both Presentation and Dialogue Parts of Multimodal Interactive Systems. *DSVIS/EHCI 2004 joint conference 11th workshop on Design Specification and Verification of Interactive Systems and Engineering for HCI*. 2004.

Navarre, D., Palanque, P., & Bastide, R. (2003) A Tool-Supported Design Framework for Safety Critical Interactive Systems. *Interacting with computers*, 15/3, 309-328.

Navarre, D, Palanque, P, and Bastide, R. A Formal Description Technique for the Behavioural Description of Interactive Applications Compliant with ARINC 661 Specification. *International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero'04)*. 2004.

Navarre, D, Palanque, P, Bastide, R, Paterno, F, and Santoro, C. A Tool Suite for Integrating Task and System Models Through Scenarios. *8th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2001*. 2001. *Lecture Notes in Computer Science*, Glasgow, Scotland, Springer.

Navarre, D, Palanque, P, Bastide, R, and Sy, O. A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications. *12th IEEE, International Workshop on Rapid System Prototyping*. 2001.

Navarre, D, Palanque, P, Dragicevic, P, and Bastide, R. An Approach Integrating two Complementary Model-based Environments for the Construction of Multimodal Interactive Applications. *Interacting with Computers*, vol. 18, n°5, 2006, pp. 910-941. 2006.

Nielsen, J and Molich, R. Heuristic evaluation of user interfaces. Chew, J. C and Whiteside, J. *Proceedings of Human Factors in Computing Systems, CHI' 90*. 249-256. 1990. ACM.

Norman, D. Categorization of action slips. *Psychological Review*. 88, 1-15. 1981.

Norman, D. *The Design of everyday things*. 1990. Doubleday Books.

Norman, D. A. *The Psychology of Everyday Things*. 1988. New York, NY, Basic Books.

NRC. Regulatory Guide 1.177 - An Approach for Plant-Specific, Risk-Informed Decisionmaking: Technical Specifications. (Draft was issued as DG-1065). Available from: <http://www.nrc.gov/reading-rm/doc-collections/reg-guides/power-reactors/active/01-177/>. 1998.

Ortmeier, F, Thums, A, Schellhorn, G, and Reif, W. Combining Formal Methods and Safety Analysis - the ForMoSA Approach. *Integration of Software Specification Techniques for Applications in Engineering*, LNCS 3147. 2003. Verlag: Springer.

- Ould, M, Bastide, R, Navarre, D, Palanque, P, Rubio, F, and Schyn, A. Multimodal and 3D Graphic Man-Machine Interfaces to Improve Operations. Eighth International Conference on Space Operations, Montréal, Canada, May 17 - 21, 2004. <http://www.spaceops2004.org/papers>. 2004.
- Palanque, P and Basnyat, S. Task Patterns for taking into account in an efficient and systematic way both standard and erroneous user behaviours. HESSD 2004 6th International Working Conference on Human Error, Safety and System Development. 2004.
- Palanque, P and Bastide, R. Petri nets with objects for specification, design and validation of user-driven interfaces. Proceedings of the third IFIP conference on Human-Computer Interaction, Interact'90. 1990 .
- Palanque, P. and Bastide, R. Synergistic modelling of tasks, system and users using formal specification techniques. *Interacting With Computers*, Academic Press. 9, 12, 129-153. 1997.
- Palanque, P, Navarre, D., and Gaspard-Bouline, H . MEFISTO Method version 1. September 2000. The Mefisto Project ESPIRIT Reactive LTR 24963 Project WP2-7. 2000.
- Palanque, P. Bastide R. Paterno F. Formal Specification as a Tool for Objective Assessment of Safety-Critical Interactive Systems. *INTERACT 1997*: 323-330. 1997.
- Palmer, E. "Oops It Didn't Arm", A Case Study Of Two Automation Surprises . Proceedings Of The Eighth International Symposium On Aviation Psychology. 1995.
- Pangoli, S and Paterno, F. Automatic generation of task-oriented help. In *ACM Symposium on User Interface Software and Technology*, pages 181--187, 1995. 1995.
- Parnas, D. L. On the use of transition diagrams in the design of a user interface for an interactive computer system. In *Proceedings 24th National ACM Conference*. pp. 379-385. 1969.
- Parnas, D. L. On the Use of Transition Diagrams in the Design of a User Interface for an Interactive System. *Proceedings of the National ACM Conference*, 379-385. 1969.
- Paterno, F. *Model Based Design and Evaluation of Interactive Applications*. 1999. Berlin, Springer Verlag.
- Paterno, F, Breedvelt-Schouten, I, and de Koning, N. Deriving presentations from task models. In *Engineering for Human-Computer Interaction*. 1999. Kluwer Academic Pub. pp. 319-338.
- Paterno, F and Faconti, G. On the Use of LOTOS to Describe Graphical Interaction . Monk, Diaper & Harrison eds. *People and Computers VII: Proceedings of the HCI'92 Conference*. pp.155-173. 1992. Cambridge University Press.
- Paterno F and Mancini, C. Developing Task Models from Informal Scenarios. *Proceedings ACM CHI'99, Late Breaking Results*. 1999. ACM Press, Pittsburgh, May 1999.
- Paterno, F and Santoro, C. Preventing user errors by systematic analysis of deviations from the system task model. *International Journal Human-Computer Studies*. 56, N.2, 225-245. 2002. Elsevier Science.
- Peters, G. A and Peters, B. J. *Human Error Causes and Control*. 2006. Taylor and Francis.
- Petoud, I. Génération automatique d'Interface Homme-Machine d'une application de gestion hautement interactive. Thèse de l'Ecole des Hautes Etudes Commerciale de l'Université de Lausanne. 1990.
- Petri, C. A. *Kommunikation mit automaten*. PhD Thesis. 1962. Technical University Darmsteadt .
- Pidgeon, N. The Limits to Safety? Culture, Politics, Learning and Man-Made Disasters. *Journal of Contingencies and Crisis Management*, 5 (1). 1-14. 1997.
- Pierotti, D. Heuristic Evaluation - A System Checklist, Xerox Corporation Available online at <http://www.stcsig.org/usability/topics/articles/he-checklist.html>. 1995.

Bibliography

- Pocock, S., Fields, B, Harrison, M, and Wright, P. THEA A Reference Guide. 2001. University of York Computer Science Technical Report 336.
- Polet, P, Vanderhaegen, F, and Amalberti, R. Modelling border-line tolerated conditions of use (BTCU) and associated risks. *Safety science (Saf. sci.)* ISSN 0925-7535. vol. 41, no 2-3 (1 p.1/4), pp. 111-136. 2003.
- Polet, P, Vanderhaegen, F, and Wieringa, P. Theory of safety related violation of system barriers. *Cognition Technology & Work*, 4, 3, 171-179. 2002.
- Potts, C, Finkelstein, A. C. W, Aslett, M., and Booth, J. "Structured Common Sense": A requirements elicitation and formalization method for modal action logic. Technical Report Alvey Initiative FOREST Report R2. Department of Computing, Imperial College of Science, Technology and Medicine, London. 1986.
- Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994) *Human-Computer Interaction*. Addison-Wesley, Wokingham, UK.
- Puerta, A. R. The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development. 4th International Conference on Computer-Aided Design of User Interfaces (CADUI'02). pp. 19-35. 2002.
- Pumfrey, D.J. (1999) *The Principled Design of Computer System Safety Analyses*. PhD Thesis. University of York.
- Rasmussen, J. Skills, rules, knowledge: Signals, signs, and symbols and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*. 13(3), 257-267. 1983.
- Rasmussen, J. Mental models and the control of action in complex environments. In D. Ackermann, D. & M. J. Tauber Eds. *Mental Models and Human-Computer Interaction*. 1, 41-46. 1990. North-Holland: Elsevier Science Publishers.
- Reason, J. *Human Error*. 1990. Cambridge University Press.
- Reason, J. (1997) *Managing the Risks of Organizational Accidents.*, Aldershot, UK: Ashgate.
- Rouvroye, J. L and Bliet, E. G. van den. Comparing safety analysis techniques. *Reliability Engineering and System Safety*. 75-3, 289-294. 2002.
- Royce, Winston. *Managing the development of large software systems: Concepts and techniques*. WESCON technical papers. 1970.
- RTCA. RTCA/DO-178B. *Software Considerations in Airborne Systems and Equipment Certification*. December 1. <http://www.rtca.org/>. 1992.
- Rushby, J. Using Model Checking to Help Discover Mode Confusions and Other Automation Surprises. 3rd Workshop on Human Error, Safety, and System Development. 1999.
- Rushby, J. Analyzing Cockpit Interfaces using Formal Methods. *Electronic Notes in Theoretical Computer Science*. <http://www.elsevier.nl/locate/entcs/volume43.html>. 23. 2001.
- Rushby, J, Crow, J, and Palmer, E. An automated method to detect mode confusions. 18th AIAA/IEEE Digital Systems Avionics Conference (DASC). 1999.
- Sammarco, J. J, Fisher, T. J, Welsh, J. H, and Pazuchanics, M. J. *System Safety Evaluation Program. Programmable Electronic Mining Systems: Best Practice Recommendations (In Nine Parts)*. Mine Safety and Health Administration, Approval and Certification Center & Department of Health and Human Services, National Institute for Occupational Safety and Health. May, 2001. Triadelphia, West Virginia.
- Sarter, N, Woods D. & Billings C. Automation surprises. In G. Salvendy, (Ed.) *Handbook of human factors*

and ergonomics, 2nd ed. p. 1926-1943. 1997. New York:Wiley.

Satchwell, R. E. Using Functional Flow Diagrams to Enhance Technical Systems Understanding. *Journal of Industrial Teacher Education*. Volume 34, Number 2. Winter 1997. 1997.

Scapin, D and Pierret-Golbreich, C. Towards a Method for Task Description: MAD. Berlinguet, L and Berthelette, D. eds. *Proceedings of the Work with display units Conference WWU'89*. (189) 2734. 1989. Amsterdam, Elsevier Science Publishers.

Schupp, B, Basnyat, S, Palanque, P, and Wright, P. A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems. 9th International Symposium of the ISSA Research Section Design process and human factors integration: Optimising company performances. 2006.

Schupp, B, Smith, S, Wright, P, and Goossens, L. Integrating human factors in the design of safety critical systems - A barrier based approach. W. Johnson and P. Palanque. *Human Error, Safety and Systems Development (HESSD 2004)*. Vol. 152, 285-300. 2004. Springer.

Shappell, S. A and Wiegmann D.A. A human error approach to accident investigation: The taxonomy of unsafe operations. *The International Journal of Aviation Psychology*, 7, 269-91. 1997.

Shappell, S. A and Wiegmann D.A. The Human Factors Analysis and Classification System–HFACS. U.S. Department of Transportation Federal Aviation Administration Report Number: DOT/FAA/AM-00/7. February 2000.

Shiffman, S, Degani, A, and Heymann, M. UIVerify - a web-based tool for verification and automatic generation of user interfaces. *Proceedings of the 8th Annual Applied Ergonomics Conference*. New Orleans, LA. 2005.

Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction* 3rd Edition. 1998. Addison Wesley Longman.

Sinnig, D, Forbrig, P, and Seffah, A. Patterns in Model-Based Development. Position Paper in INTERACT 03 Workshop entitled: Software and Usability Cross-Pollination: The Role of Usability Patterns. 2003.

Sklet, S. Methods for accident investigation. Norwegian University of Science and Technology Technical Report. ROSS (NTNU) 200208. 2002.

Smith, D. J. Study Unit HE2 - Disasters (Related Theory). Available from: <http://www.smithsrisca.demon.co.uk/unitHE2.html> (last accessed 16/11/2006). 2002.

Smith, S. P and Harrison, M. D. Improving Hazard Classification through the Reuse of Descriptive Arguments. Gacek, C. *Proceedings of the 7th international Conference on Software Reuse: Methods, Techniques, and Tools*. 2319, 255-268. 2002. Springer-Verlag, London. *Lecture Notes In Computer Science*.

Smith, S. P, Harrison, M. D, and Schupp, B. A. How explicit are the barriers to failure in safety arguments? M. Heisel, P. Liggesmeyer and S. Wittmann Eds. *Computer Safety, Reliability, and Security (SAFECOMP'04) Lecture Notes in Computer Science*. 3219, pp. 325-337. 2004. Springer.

Sommerville, I. Human Fallibility. Case study : The London Ambulance Service Despatching System. Available from: <http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/CaseStudies/LondonAmbulance/index.html> . 2003 .

Storey, N. *Safety-Critical Computer Systems*. 1996. Addison Wesley.

Suchman, L. A. *Plans and situated actions: The problem of human-machine communications*. Cambridge, UK: Cambridge University Press. 1987.

Sutcliffe, A. G and Rugg, G. A taxonomy of error types for failure analysis and risk assessment.

- International Journal of Human Computer Interaction 10, 4, 381–406. 1998.
- Svenson, O. The Accident Evolution and Barrier Function (Aeb) Model Applied to Incident Analysis in the Processing Industries. Risk Analysis . 11(3): 499-507 . 1991.
- Tarby, J. C and Barthet, M. F. The Diane+ method. Second international workshop on computer-aided design of user interfaces. 1996.
- Thorley, A. R. D. Fluid Transients in Pipeline Systems. Co-published by Professional Engineering Publishing, UK, and ASME Press. 2004.
- Tsai, W Y. Formal Verification of Safety Critical Systems with SafeCharts. MSc Thesis. 2004. Taiwan, Republic of China, Institute of Computer Science and Information Engineering, National Chung Cheng University.
- van Biljon, W. R. Extending Petri Nets for Specifying Man-Machine Dialogues. . International Journal of Man-Machine Studies . 1988. 28(4): 437-455.
- van Dam, A. Post-WIMP User Interfaces. Communications of the ACM 40, 2, 63-67. 1997.
- van der Veer, G, van der Lenting, B. F, and Bergevoet, B. A. J. GTA: Groupware Task Analysis - Modeling Complexity. Acta Psychologica . 91(3), 297–322. 1996.
- van Welie M, van der Veer G.C, and Eliëns A. Patterns as Tools for User Interface Design. International Workshop on Tools for Working with Guidelines. pp. 313-324. 2000.
- van Zijl, L and Mitton, D. Using statecharts to design and specify a directmanipulation user interface. In Proceedings of the Southern Africa Computer Symposium pages 51–68. 1991.
- Vanderdonckt, J and Puerta, A. Introduction to Computer-Aided Design of User Interfaces. Proceedings of the CADUI'99. 1999. Louvain-la-Neuve, Kluwer Academic Publishers.
- Vanderhaegen, F. Analyse et Contrôle de l'Erreur Humaine dans les Systèmes Homme-Machine. Pour l'obtention de l'Habilitation à Diriger des Recherches. Université de Valenciennes et du Hainaut-Cambrésis. 2003.
- von der Beeck, M. A Comparison of Statecharts variants. LNCS. 863, pp. 128-148. 1996. Springer-Verlag.
- Walldius, A. Patterns can bring in knowledgeable users into the software development cycle. 2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns Official workshop of IFIP working group 13.2 INTERACT 2003. 2003.
- Weill-Fassina, A, De La Garza, C, and Kaplan, M. Integrating human factors in freight interoperability safety design. Design Process and Human Factors Integration: Optimising company performance. International Symposium. ISSA Research Section. 2006.
- Wesson, J and Cowley, L. Designing with Patterns: Possibilities and Pitfalls. 2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns Official workshop of IFIP working group 13.2 INTERACT 2003. 2003.
- Winckler, M. StateWebCharts : une notation formelle pour la modélisation de la navigation des applications Web. PhD Thesis. University of Toulouse I. 2004.
- Wirstad, J. On knowledge structures for process operators. In L.P. Goodstein, H. B. Andersen & S. E. Olsen Eds. Tasks, Errors, and Mental Models . pp.50-69. 1988. London, Taylor and Francis.
- Woods, D. D. Creating foresight: Lessons for resilience from Columbia. Farjoun, M and Starbuck, W. H. Organization at the limit: NASA and the Columbia disaster. 2005. Blackwell.
- Woods, W. A. Transition network grammars for natural language analysis'. Communications of the ACM,

Vol. 13 No. 10, pp. 591-606. 1970.

Wright, P, Fields, B, and Harrison, M. Deriving Human-Error Tolerance Requirements from Tasks . BAe Dependable Computing Systems Centre. IEEE International Conference on Requirements Engineering. 1994.

List of figures

Chapter 1

<i>Figure 1-1. Representation of an Interactive System</i>	7
<i>Figure 1-2. Intersection of Chapter Theories</i>	8
<i>Figure 1-3. The RUP Lifecycle (from (Kruchten 2000))</i>	10
<i>Figure 1-4. a) Choice Operator, b) Enabling Operator with varied task type</i>	17
<i>Figure 1-5. Simplified example of a cooperative task (taken from (Mori et al. 2002))</i>	18
<i>Figure 1-6. Extract of the interactive activity pattern, “iterate” (taken from (Mahemoff and Johnston 1998))</i>	21
<i>Figure 1-7. Find pattern and its sub-patterns modelled in UML (taken from (Sinnig et al. 2003))</i>	23
<i>Figure 1-8. Model-based approaches and notations for Interactive Systems engineering.</i>	25
<i>Figure 1-9. Example of an Entity Relationship Model</i>	26
<i>Figure 1-10. Example of a (a) finite state machine and a (b) state transition table</i>	28
<i>Figure 1-11. Example of a statechart</i>	29
<i>Figure 1-12. Example of a Petri net with two states</i>	31
<i>Figure 1-13. Basic Petri net with an inhibitor arc, a) before firing of transition t1, b) after firing of transition t1</i>	32
<i>Figure 1-14. Basic Petri net with a test arc, a) before firing of transition t1, b) after firing of transition t1</i>	32
<i>Figure 1-15. Simple Petri net to explain marking graph analysis technique</i>	33
<i>Figure 1-16. Graphical Marking Graph for Petri net in Figure 1-15</i>	34
<i>Figure 1-17. Screenshot of the JARP Petri net editor tool</i>	34
<i>Figure 1-18. Screenshot of the ARP Petri net analysis tool</i>	35
<i>Figure 1-19. Petri net of Figure 1-17 opened in the ARP tool</i>	35
<i>Figure 1-20. Results of reachable states for Petri net in Figure 1-17</i>	35
<i>Figure 1-21. Marking Tree for Petri net in Figure 1-17</i>	36
<i>Figure 1-22. Marking graph</i>	36
<i>Figure 1-23. Architecture of the PetShop Environment</i>	38
<i>Figure 1-24. The ObCS editor in Petshop</i>	39
<i>Figure 1-25. Interactive Prototyping in Petshop</i>	39
<i>Figure 1-26 Partial view of the MPIA Server with displayed arc labels (taken from (Barboni et al. 2006b))</i>	40
<i>Figure 1-27 Partial view of the MPIA Server with arc labels masked (taken from (Barboni et al. 2006b))</i>	41
<i>Figure 1-28. Real view (left) versus Semantic view (right)</i>	41
<i>Figure 1-29. An example of an OFAN model (taken and adapted from (Degani and Kirlik 1995))</i>	46
<i>Figure 1-30. Schematic view of the Process cycle</i>	47
<i>Figure 1-31. Detailed iterative design process (adaptation from (Palanque et al. 2000))</i>	48
<i>Figure 1-32. Content and structure of the Abstraction cycle (from (Palanque et al. 2000))</i>	48

Chapter 2

Figure 2-1. Required Qualities of a Resilient System(Hollnagel and Woods 2006a)	53
Figure 2-2. Definition of a system (taken from (Leveson 1995))	53
Figure 2-3. Safety as a non-event (Hollnagel 2006a)	55
Figure 2-4. Reactive Organisation (Hollnagel 2006a)	56
Figure 2-5. Reactive Organisation (Hollnagel 2006a)	56
Figure 2-6. Reactive Organisation (Hollnagel 2006a)	57
Figure 2-7. Framework for the Formalisation for Accident Reports (from (Burns 2000)	65
Figure 2-8. Results of Question 36 of the Ernst and Young Global Information Security Survey 2003 (from (Avizienis et al. 2004b)) based on original graph P.13 (Ernst & Young 2003a).	66
Figure 2-9. Fault Tolerance Techniques (from (Avizienis et al. 2004a))	67
Figure 2-10. The classes of combined faults, tree representation (from (Avizienis et al. 2004a))	71
Figure 2-11. Principle Elements of the Goal Structuring Notation (taken from (Kelly 2003))	75
Figure 2-12. Location of the three performance levels within an ‘activity space’ defined by the dominant mode of action control and the nature of the local situation (from (1997))	78
Figure 2-13. The Generic Error Modelling System (GEMS) (Reason 1990)	79
Figure 2-14. Key Features in GEMS ((Reason 1990)p62)	80
Figure 2-15. Hollnagel's taxonomy of phenotypes (from (Hollnagel 1993a) p.76(Hollnagel 1991))	81
Figure 2-16. Summary of the principle error types (from (1997) p.72)	81
Figure 2-17. Peters & Peters’ Countermeasures (Peters and Peters 2006) mapped to development process	84
Figure 2-18. The Domino Theory (based on (Heinrich et al. 1980) taken from (Hollnagel 2004))	86
Figure 2-19. “Swiss Cheese” Model in an Ideal World	86
Figure 2-20. James Reason’s “Swiss Cheese” Model	87
Figure 2-21. Categories of unsafe acts committed by aircrews (from (Shappell and Wiegmann D.A 2000))	88
Figure 2-22. Categories of preconditions of unsafe acts (from (Shappell and Wiegmann D.A 2000))	88
Figure 2-23. Categories of unsafe supervision (from (Shappell and Wiegmann D.A 2000))	88
Figure 2-24. Organisational factors influencing accidents (from (Shappell and Wiegmann D.A 2000))	88
Figure 2-25. HFACS Extension to Reason’s Swiss Cheese Model	89

Chapter 3

Figure 3-1. Multi-Type Data Cube	100
Figure 3-2. Formal low level and complete data modelling using Petri-nets	100
Figure 3-3. High-level data, communication sequence diagram from UML	101
Figure 3-4. Design process centred on formal models (Navarre 2001)	104
Figure 3-5. Ingredients of the system model	107
Figure 3-6. Generic Integrated Modelling Framework	108
Figure 3-7. System modelling phase of the generic integrated modelling framework	109
Figure 3-8. Task modelling phase of the generic integrated modelling framework	110
Figure 3-9. Safety modelling phase of the generic integrated modelling framework	110
Figure 3-10. Testing phase of the generic integrated modelling framework	111

Chapter 4

List of Figures

Figure 4-1. Task modelling phase of the generic integrated modelling framework highlighting predicted user behaviour	115
Figure 4-2. CTT ATM example provided within CTTe	116
Figure 4-3. Task modelling phase of the generic integrated modelling framework highlighting safety-related user behaviour	118
Figure 4-4. Screenshot of Skill-based Human Error Reference Table	120
Figure 4-5. Sub-section of CTT Task Analysis for Withdrawing Cash at an ATM highlighting the “Insert PIN” activity for analysis	122
Figure 4-6. Results of the systematic analysis of possible deviations while inserting a pin at an ATM based on a CTT task analysis model and the skill-based HERT	123
Figure 4-7. Four possibilities of interaction when entering PIN	124
Figure 4-8. a) p1: PIN OK, b) p2: PIN not ok	124
Figure 4-9. a) p3 Too long entering PIN, b) p4: Simple Timeout	124
Figure 4-10. p5: Timeout on PIN confirmation	125
Figure 4-11. Model pattern 1, showing abstract tasks only	125
Figure 4-12. Model pattern 1, showing all leaves	125
Figure 4-13. Model pattern 2, showing abstract tasks only	126
Figure 4-14. Model pattern 3, showing abstract tasks only	126
Figure 4-15. Model pattern 4, showing abstract tasks only	126
Figure 4-16. a) model pattern 2, showing all leaves, b) model pattern 3, showing all leaves	127
Figure 4-17. Illustration of “plugging in” model pattern for error tolerance of “insert PIN” subtask	129
Figure 4-18. Task modelling phase of the generic integrated modelling framework	130
Chapter 5	
Figure 5-1. System modelling phase of the generic integrated modelling framework highlighting the functional behaviour	134
Figure 5-2. Basic ATM ICO System Model	136
Figure 5-3. ATM Interface	137
Figure 5-4. System modelling phase of the generic integrated modelling framework highlighting part of the safety-related behaviour	141
Figure 5-5. ATM System Model Including Data Entry Errors, Card Errors and Cancel Features	142
Figure 5-6. PIN Entry Part of the Extended System Model (label 1 in Figure 5-5)	143
Figure 5-7. PIN Verification Part of the Extended System model (labels 2a and 2b in Figure 5-5)	144
Figure 5-8. ErrorPin and TooManyTries Part of the Extended System model (label 3 in Figure 5-5)	144
Figure 5-9. Cancel Feature Part of the Extended System model (label 5 in Figure 5-5)	145
Figure 5-10. a) Unlimited PIN Entry Model, b) One Attempt PIN Entry Model	145
Figure 5-11. ATM System Model with Full Interleaving	146
Figure 5-12. . ATM System Model with One Interleaving Option	147
Figure 5-13. Timing Feature of the Extended System model (label 7 in Figure 5-5)	148
Figure 5-14. Numbering the Places of the Simple ATM System Model	150
Figure 5-15. Marking Graph for Simple ATM presented in Figure 5-2	151
Figure 5-16. System modelling phase of the generic integrated modelling framework	152

Chapter 6

Figure 6-1. Safety modelling phase and testing phase of the generic integrated modelling framework	155
Figure 6-2. Key Phases of the Approach	158
Figure 6-3. High Level Safety Case of ATM in Airlock Using GSN	160
Figure 6-4. Goal 1 Safety Case Using GSN	160
Figure 6-5. Goal 2 Safety Case Using GSN	160
Figure 6-6. Goal 3 Safety Case Using GSN	161
Figure 6-7. Goal 4 Safety Case Using GSN	161
Figure 6-8. Safety Case Diagram Key	161
Figure 6-9. Simple ECFA for ATM Airlock “incident”	162
Figure 6-10. System Model Including Hazardous State “User Trapped”	164
Figure 6-11. Airlock Feature of Informed System Model (label 1 of Figure 6-10)	165
Figure 6-12. Fire and Alarm Feature of Informed System Model (label 2 of Figure 6-10)	166
Figure 6-13. Card in Machine Feature of Informed System Model (label 3 of Figure 6-10)	167
Figure 6-14. Improving the System Model to Avoid Hazardous State	169
Figure 6-15. Modification Made to System Model to avoid Hazardous State	170
Figure 6-16. Safety Modelling and Testing Phases of the Generic Integrated Modelling Framework	171

Chapter 7

Figure 7-1. Barrier Analysis and Modelling With Respect to the Generic Integrated Modelling Framework	174
Figure 7-2. Approach Diagram	178
Figure 7-3. A Typical H-B-T Diagram for the ATM Airlock Example Including SML Key	179
Figure 7-4. SML representation of (a) fire hazard, (b) a human error hazard, (c) recursion	179
Figure 7-5. Software Barrier Identification, Analysis and Modelling Approach	182
Figure 7-6. ATM Interface (Welcome Screen)	183
Figure 7-7. ATM Interface (Enter PIN Screen)	183
Figure 7-8. ATM Interface (Select Option Screen)	183
Figure 7-9. ATM Interface (Select Amount Screen)	184
Figure 7-10. ATM Interface (Amount Selection Screen)	184
Figure 7-11. ATM Interface (Card Ejection Screen)	184
Figure 7-12. New arcs between existing components of the system model	188
Figure 7-13. New components of the system model	188
Figure 7-14. New arcs between existing components of the system model the new components	189
Figure 7-15. Receipt Feature of System Model Before Barrier Integration	189
Figure 7-16. Receipt Feature of System Model With Barrier B1	189
Figure 7-17. Barrier Model and System Model Integration Concept	190
Figure 7-18. Barrier Analysis and Modelling With Respect to the Generic Integrated Modelling Framework	191
Figure 7-19. Comparing the Costs of Classical Development Processes with UCD Processes	195
Figure 7-20. Comparing the Costs of Classical Development Processes with	

<i>Safety Centred Development</i>	196
<i>Figure 7-21. Comparing the Costs of Classical Development Processes with Reliable Centred Development</i>	197
Chapter 8	
<i>Figure 8-1. Phases of the Generic Integrated Modelling Framework not illustrated on this case study</i>	205
<i>Figure 8-2. Surface quarry and cement plant location (Photo from the National Institute of Standards and Technology NIST http://www.cstl.nist.gov/acd/ashgrove.htm)</i>	206
<i>Figure 8-3. Illustration of a) Grinder, b) Pump-G and c) Pump-S</i>	207
<i>Figure 8-4. Location of the accident within the plant (from the MSHA http://www.msha.gov/FATALS/2002/FTLO2M34.HTM)</i>	207
<i>Figure 8-5. Simplified Layout Diagram of Waste Fuel Delivery Plant</i>	209
<i>Figure 8-6. Fuel Hammer Effect</i>	210
<i>Figure 8-7. Overview of multidisciplinary perspectives and information propagation into models</i>	211
<i>Figure 8-8. Kiln Control Operator Task Model</i>	212
<i>Figure 8-9. Supervisor Task Model</i>	212
<i>Figure 8-10. Worker task model</i>	213
<i>Figure 8-11. Cooperative Task Model for Switching from the North to the South Waste Fuel Delivery System</i>	214
<i>Figure 8-12. Pattern 1 for bleed valve subtask</i>	216
<i>Figure 8-13. Pattern 2 for bleed valve subtask</i>	216
<i>Figure 8-14. Pattern 3 for bleed valve subtask</i>	217
<i>Figure 8-15. Pattern 4 for bleed valve subtask</i>	217
<i>Figure 8-16. Plugging in patterns to “bleed air in SWFS” subtask of worker task model</i>	218
<i>Figure 8-17. Safety Case Arguing the Safety of the Waste Management Piping System</i>	219
<i>Figure 8-18. Considering all hazards that can arise from High-Pressure</i>	220
<i>Figure 8-19. Safety Case for the Monitoring of Pressure function</i>	221
<i>Figure 8-20. Safety Case for the ‘Monitoring of Pressure in the South Delivery System’ Function</i>	221
<i>Figure 8-21. Safety of Manual activities</i>	222
<i>Figure 8-22. Arguing the safety of the PLC</i>	223
<i>Figure 8-23 High-level Sequence Diagram</i>	225
<i>Figure 8-24. ECFA Chart of the Accident</i>	226
<i>Figure 8-25. Connections among components in the Monitoring System</i>	227
<i>Figure 8-26. a) Fuel storage tanks (Applied to the north and south) b) North Pump S Valve</i>	228
<i>Figure 8-27. a) North Pump S Motor b)The North Grinder</i>	229
<i>Figure 8-28. Fuel Plant Kilns</i>	229
<i>Figure 8-29. Petri net model of the PLC</i>	230
<i>Figure 8-30. Complete System Model</i>	231
<i>Figure 8-31. North pump S motor & valve including hazards</i>	234
<i>Figure 8-32. Complete System Model Including Hazards</i>	235
<i>Figure 8-33. Petri net modelling a north pump-s and its motor</i>	236
<i>Figure 8-34. Reduced Petri net for SWFD</i>	238

<i>Figure 8-35. Extract of Marking Graph Showing Non-Explosive States Leading to an Explosive State</i>	239
<i>Figure 8-36: Piping and Instrumentation Diagram of fuel line. See text for explanation.</i>	240
<i>Figure 8-37. SML representation of hazards, barriers and targets discussed in this text (PB1 and PB2 modelled in this chapter)</i>	242
<i>Figure 8-38. PB1 Pipe Priming Procedure Barrier</i>	244
<i>Figure 8-39. PB2: Auto shut down of pumps after 3 minutes barrier</i>	245
<i>Figure 8-40. ICO System Model + PB1</i>	247
<i>Figure 8-41. ICO System Model + PB2</i>	250

Chapter 9

<i>Figure 9-1. Interactive Cockpits</i>	253
<i>Figure 9-2. Phases of the Generic Integrated Modelling Framework not illustrated on this case study</i>	256
<i>Figure 9-3. MPIA Application in THALES Test Bench for interactive cockpits</i>	257
<i>Figure 9-4: Multipurpose Interactive Application (MPIA)</i>	257
<i>Figure 9-5 Snapshot of the page 1 of the MPIA UA (WXR) - a simple highly interactive application based on the look & feel of THALES</i>	258
<i>Figure 9-6 Snapshot of the page 1 of the MPIA UA (GCAS)</i>	258
<i>Figure 9-7 Snapshot of the page 1 of the MPIA UA (AIRCOND)</i>	259
<i>Figure 9-8. Examples of ARINC 661 Widgets (buttons, check box, edit boxes, labels, scroll lists)</i>	260
<i>Figure 9-9. Examples of ARINC 661 widgets specific to cockpits (Map management, masks, AESS bitmap)</i>	260
<i>Figure 9-10 Detailed architecture to support ARINC 661 specification</i>	260
<i>Figure 9-11. ICO Model of the WXR page of the MPIA presented in Figure 9-5</i>	262
<i>Figure 9-12. Software Barrier Identification, Analysis and Modelling Approach</i>	264
<i>Figure 9-13. Behaviour of the Tilt angle edition using ICOs</i>	270
<i>Figure 9-14. B9 Barrier modelling for Tilt angle edition</i>	271

List of tables

Table 1-1. CTT Task Types	16
Table 1-2. Operators used in the CTT notation	16
Table 1-3. Summary of Task Analysis Notations State of the Art with respect to Languages	19
Table 1-4. Summary of Task Analysis Notations State of the Art with respect to Tool Support	20
Table 1-5. Requirements Types and Stakeholders	25
Table 1-6. Table describing effective time for each action or transition (adapted and taken from (Lacaze et al. 2002))	42
Table 2-1. Barrier systems and barrier functions (taken from (Hollnagel 1999a))	61
Table 2-2. A comparative assessment of the three accident analysis techniques (from (Kontogiannis et al. 2000))	64
Table 2-3. Top ten loss of availability incidents. Results of the Ernst and Young Global Information Security Survey 2004 (from (Ernst & Young 2003b) p.14).	66
Table 2-4. The Human Reliability Design Triptych (from (Basnyat et al. 2006))	85
Table 3-1. Generic Integrated Modelling Framework Approach With Respect to Design Column of The Human Reliability Design Triptych	98
Table 3-2. Generic Integrated Modelling Framework Approach With Respect to Modelling Column of The Human Reliability Design Triptych	98
Table 3-3. Positioning our generic integrated modelling framework with respect to the state of the art	106
Table 4-1. Grouping ATM Human Error Analysis Results	123
Table 5-1. User and System behaviour	138
Table 5-2. Activation Function for Figure 5-2	138
Table 5-3. Rendering Function for Figure 5-2	138
Table 5-4. Modelling Impact of Skill-based PIN Entry Errors	140
Table 6-1. Relating ECFA with System Model including hazard	167
Table 7-1. Hazards, barriers and targets	176
Table 7-2. Detailed barrier analysis and correspondence to Safety Cases	176
Table 7-3. Extract of the HERTs Analysis on ATM UI	185
Table 7-4. Extract of Heuristic Evaluation Results on ATM UI	186
Table 7-5. Extract of HAZOP analysis results on ATM UI	186
Table 7-6. Barrier Suggestion and Classification	187
Table 8-1. Hazards, targets and barriers	223
Table 8-2. Detailed barrier analysis and correspondence to Safety Cases	224
Table 8-3. Relating ECFA with System Model	233
Table 8-4. Summary of Hazardous States Identified from Incident and Accident Investigation	234
Table 9-1. Activation function of WXR page	263
Table 9-2. Rendering function of WXR page	263
Table 9-3. Skill based Human Error Analysis Results on Weather Radar (WXR) page	264
Table 9-4. Heuristic Evaluation Results	266
Table 9-5. HAZOP analysis results on WXR tab	267
Table 9-6. Classification of Suggested Barriers	269

PART 4 – ANNEX

Skill-Based Human Error Reference Table

Reason's GEMS & Rasmussen's SRK Classification	Reason's Main Failure Modes	Reason's Common Mechanisms	Error Name & Definition	Examples of errors	Error Type Ref
Skill-based level	Inattention (Omitted checks)	Double-capture slips <i>Involve two distinct, though casually related, kinds of capture.</i> (Reason 1990) (Norman's (Norman 1990) 1st category of slips)	Strong-habit intrusion <i>The unintended activation of the strongest action scheme beyond the choice point.</i> (Reason 1990) Hollnagel (Hollnagel 1993a), simple phenotype, Intrusion. Error mode: action not included in current plans. Corresponding complex phenotype, Hollnagel, 1993 capture, branching and overshoot.	Unintended activation of the strongest action schema beyond the choice point (Reason 1990)	(Reason 1990)
			Strong-habit capture	On starting a letter to a friend, I headed the paper with my previous home address instead of my new one. (Reason 1990)	(Reason 1990)
			Strong-habit exclusion	I intended to stop on the way to work to buy some shoes, but 'woke up' to find that I had driven right past. (Reason 1990)	(Reason 1990)
			Branching errors <i>An initial common action sequence leads to different outcomes and the attentional check at the choice-point is omitted.</i> (Reason 1990)	I brought the milk in to make myself a cup of tea. I had put the cup and saucer out previously. But instead of putting the milk into the cup, I put the bottle straight into the fridge. (Reason 1990)	(Reason 1990)
				I meant to get my car out, but as I passed through the back porch on the way to the garage, I stopped to put on my Wellington boots and gardening jacket as if to work in garden. (Reason 1990)	(Reason 1990)

		Overshooting a stop rule	I went to my bedroom to change into something more comfortable for the evening, and the next thing I knew I was getting into my pyjama trousers, as if to go to bed. (Reason 1990)	(Reason 1990)
		Program Counter Failures Counting <i>An extraneous event as a part of the intended sequence - causing an omission or failing to count a relevant action causing repetition which is less common (Reason, 1984)</i>	I picked up my coat to go out when the phone rang. I answered it and then went out of the front door without my coat (Reason 1990)	(Reason 1990)
	Omissions following interruptions <i>Failure to make an attentional check is compounded by some external event. (Reason 1990)</i>	Trips	A trip is a forcible interruption to an ongoing motor program, as in a trip while walking (Smith 2002)	(Reason 1990)
	Reduced Intentionality <i>Some delay intervenes between the formulation of an intention to do something and the time for this activity to be executed. (Reason 1990)</i>	Detached Intentions <i>The action element of the sequence of events either is neglected or performed with another object or trigger from an external distraction becomes incorporated into the desired sequence.</i>	I intended to close the window as it was cold. I closed the cupboard door instead. (Reason, 1990)	(Reason 1990)
		Environmental Capture	I went into my bedroom intending to fetch a book. I took off my rings, looked in the mirror and came out again - without the book. (Reason, 1990) (Norman's (Norman 1990) 5th category of slip Loss of activation error)	(Reason 1990)
		I-should-be-doing-something-but-I-can't-remember-what	(Norman's (Norman 1990) 5th category of slip Loss of activation error)	(Reason 1990)
		Fumbles	A fumble is a poorly executed motor program, as in a fumbled catch at cricket. (Smith 2002)	(1997)

		Post-completion error <i>There is a precondition to the conceptual operation that achieves the main goal, but satisfying the precondition perturbs the state, and a clean-up action is needed after achievement of the main goal (Blandford, 2002)</i>	Making copies on a photocopier but forgetting to retrieve the original and forgetting to take change from a vending machine ((Curzon and Blandford 2004)	(Byrne and Bovair 1997)
		Under-motivation <i>Stemming from boredom or lack of morale also can lead to shortcuts and risk-taking. However, it is more likely to cause latent errors, discovered later by someone else (Bill Mostia, 2003)</i>		(Mostia 2002)
	Perceptual Confusion <i>Occur because the recognition schemata accept as a match for the proper object some thing that looks like it, is in the expected location or does a similar job. (Reason 1990)</i>	Description error <i>When the correct action is carried out on the wrong object. It tends to occur when one is distracted. (1994))</i>	Putting the salad in the oven and the casserole in the fridge (1994) (Norman's (Norman 1990) 2nd category of slip Description error)	(Norman 1990)
		Input or Misperception errors <i>With errors of this type, the information needed to make a decision or perform a task might be misunderstood, perhaps because it has been presented in an overly complex or misleading way. (Mostia 2002)</i>	The input data are incorrectly perceived, an incorrect intention is formed, and the wrong action is performed; that is, an action other than what would have been intended, had the input been correctly perceived (Green 2003)	(Mostia 2002)
	Interference Errors <i>Two currently active plans or, within a single plan, two action elements, can become entangled in the struggle to gain control of the effectors. (Reason 1990)</i>	Data driven error <i>When unconscious processing of external data interferes with what you had intended to do. (1994))</i>	Saving a file with the name of the file that is displayed in the adjacent window instead of the name you intended. (Preece, 1994) (Norman's (Norman 1990) 3rd category of slip Data driven error)	(Norman 1990)
		Associative-activation error <i>When internal thoughts interfere with what you were supposed to be doing. (1994))</i>	Saving a file with the name of the person you were thinking about rather than the name you had intended. (1994)(Norman's (Norman 1990) 4th category of slip associative-activation error)	(Norman 1990)

Over attention	Mistimed Checks <i>When an attentional check is omitted, the reins of action or perception are likely to be snatched by some contextually appropriate strong habit or expected pattern</i> (Reason 1990)	<p>Omission <i>Occurs if an action has been omitted from the sequence.</i> (Hollnagel, 1993) (Hollnagel's (Hollnagel 1993a) simple phenotype. Error mode: action at the wrong place. Corresponding complex phenotypes, jumping and undershoot</p>	Omit a necessary step like putting tea in pot, or switching on kettle. (Reason, 1990) Part 1a of HAZOP causes of deviation	(Hollnagel 1993a)
		<p>Repetition <i>Means that an action has been carried out twice</i> (Hollnagel, 1993) (Hollnagel's (Hollnagel 1993a) simple phenotype. Error mode: Action in the wrong place. Corresponding complex phenotype, Restart</p>	Setting the kettle to boil for a second time. (Reason 1990)	(Hollnagel 1993a)
		<p>Reversal <i>Means that two actions have been reversed in their sequence.</i> (Hollnagel, 1993) Hollnagel's (Hollnagel 1993a) simple phenotype. Error mode: Action in the wrong place. Corresponding complex phenotype, Jumping.</p>	I intended to take off my shoes and put on my slippers. I took my shoes off and then noticed that a coat had fallen off a hanger. I hung the coat up and then instead of putting on my slippers, I put my shoes back on again. (Reason 1990)	(Hollnagel 1993a)
		<p>Insertion <i>Occurs if the action does not belong to the action sequence and if it does not disrupt it.</i> (Hollnagel, 1993) Hollnagel's (Hollnagel 1993a) simple phenotype. Error mode: Action not included in current plans. Corresponding complex phenotype, side-tracking.</p>	Part 1b of HAZOP causes of deviation	(Hollnagel 1993a)
		<p>Replacement <i>The action is a proper substitute for the expected action.</i> (Hollnagel, 1993) Hollnagel's (Hollnagel 1993a) simple phenotype. Error mode: Action not included in current plans. Corresponding complex phenotype, side-tracking.</p>		(Hollnagel 1993a)

		<p>Premature Action <i>An action that occurs when no action was expected. (Hollnagel, 1993) Hollnagel's (Hollnagel 1993a) simple phenotype. Error mode: Action at wrong time.</i></p>		(Hollnagel 1993a)
		<p>Order Errors There are circumstances in which doing A then B has a different effect from doing B then A, that the actions can be performed in either order, and that the user may not be aware of the order constraint (Blandford et al. 2000)</p>		Blandford, 2001
		<p>Over-Motivation Can come from being too zealous, eg: completing a job too quickly just to please a supervisor. Working fast can lead to shortcuts and risk-taking. (Mostia 2002)</p>		(Mostia 2002)

Safety Analysis Methods

Preliminary Hazard Analysis (PHA) Form

Sample Preliminary Hazard Analysis (PHA) Form

System: <u>Pumping System</u> Subsystem: <u>Pump</u> Drawing No: 13-R1 PHA No.: 1 Rev. No. 1			PRELIMINARY HAZARD ANALYSIS (PHA) Rev. No: 1		Sheet <u>One</u> Of <u>One</u> Prepared by: Joe Safety Date: 12/8/99 Reviewed by: Supervisor Safety Date: 12/8/99 Approved by: Director Safety Date: 12/8/99		
General Description			Hazard Cause/Effect		Hazard Risk Index	Corrective Action	
No.	Hazard Description	Failure Rate	Potential Cause	Effect on Subsystem/System		Possible Controlling Measures and Remarks	Resolution
1	Pump, which removes water from tunnel, fails – causing a loss of water removal capability	1 x 10 ⁻⁶	Mechanical Failure; Maintenance Failure	Water Floods Tunnel	1C	<u>Design Change</u> : Add back-up pump <u>Procedure Change</u> : Provide scheduled maintenance check and testing	1D – Back-up pump added; maintenance and testing procedures developed and implemented, 12/9/99

from (Adduci et al. 2000)

Failure Modes and Effect Analysis (FMEA) Form

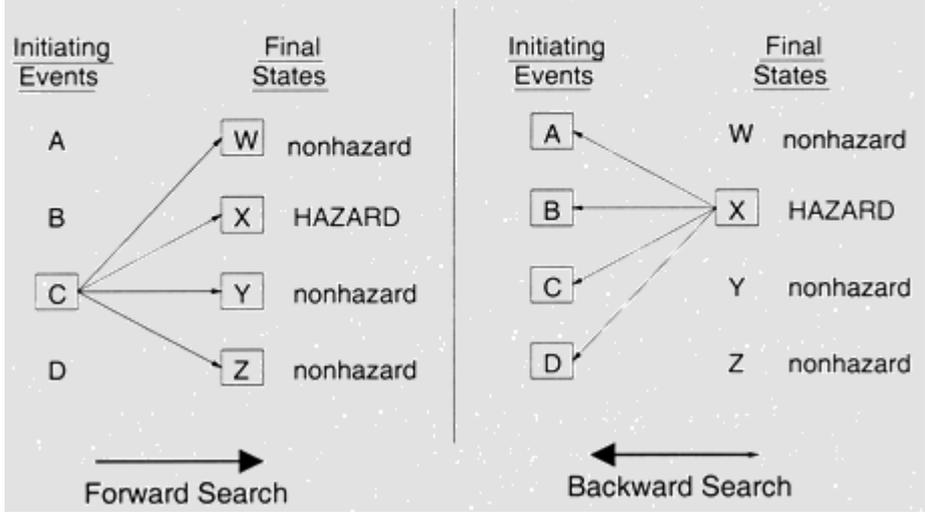
Sample Failure Modes and Effect Analysis (FMEA) Form

System: Pumping System Subsystem: Pump Controller, Power Drawing No: 1 FMEA No.: 1 Rev. No: 1			FAILURE MODES AND EFFECT ANALYSIS (FMEA)			Sheet : <u>One</u> Of: <u>One</u> Prepared by: Joe Engineer Date: 12/9/99 Reviewed by: Safety Supervisor Date: 12/9/99 Approved by: Safety Director Date: 12/9/99	
General Description			Hazard Cause/Effect			Corrective Action	
LRU No. & Description	Failure Mode	Cause of Failure	Effect of Failure on Subsystem/System	Probability of Occurrence	Severity of Occurrence	Possible Controlling Measures and Remarks	Resolution
1 – Pump controller	Shut-off indication not recognized	Electrical malfunction	Pump remains on continuously, burns out and water floods the tunnel	1 x 10 ⁻⁶	Critical	Provide low water out-off for pump Provide alarm to trigger when pump is on for more than "X" minutes	Low water out-off on pump and alarm for "pump on" time in design – 12/7/99

from (Adduci et al. 2000)

SSHA Forward vs Backward Search

Forward vs. Backward Search



from (Leveson 2003b)

Index

A

Abstract · 17
Abstraction · 50, 51
Accidents · 2, 157, 219
Action · 16, 20, 21, 67, 140, 215, 309, 310
Activation function · 40, 263
Activity · 9, 27
Analyse · i, 71
Analysis · i, 12, 15, 16, 20, 21, 25, 26, 29, 35, 46, 47, 48, 50, 60, 61, 62, 64, 65, 71, 72, 75, 84, 87, 89, 92, 124, 125, 141, 150, 155, 156, 157, 158, 159, 164, 173, 176, 177, 179, 182, 184, 187, 188, 189, 193, 204, 211, 212, 214, 219, 220, 223, 225, 237, 241, 251, 253, 263, 264, 269, 277, 311
Application · 17, 71, 74, 111, 161, 253, 255, 256, 257, 261, 277

B

Barrier analysis · 62, 175, 177, 178
Barriers · 175, 177, 178, 183, 190, 224, 255, 269

C

Cognitive · 16, 20, 45, 46, 48
Collaboration · 9, 20
Command · 203, 222
Complexity · 48, 277, 278
Concept · 192
Control · 18, 26, 40, 75, 203, 208, 210, 211, 212, 221, 222, 224, 242, 273, 277
Cooperative · 29, 39, 40, 43, 45, 47, 111, 135, 158, 193, 214, 228, 254, 255, 270, 273

D

Description · 16, 17, 20, 21, 142, 215, 235, 258, 259, 261, 265, 308
Design · i, 2, 7, 8, 12, 14, 23, 26, 46, 48, 49, 50, 58, 78, 87, 91, 99, 100, 106, 109, 115, 116, 132, 196, 273, 277, 278
Development · 46, 92, 196, 197, 198, 277, 278
Dialogue · 14
Domain · 14, 125

E

Engineering · 8, 27, 37, 49, 53, 59, 91, 110, 116, 123, 208, 256, 259, 277, 278
Environment · 16, 41, 48
Evaluation · 24, 25, 71, 187, 188, 266
Events · 61, 65, 155, 157, 158, 159, 164, 173, 174, 204, 210, 224, 225, 233, 251

F

Formal · 8, 20, 28, 29, 30, 35, 47, 64, 65, 96, 102, 106, 108, 156, 158, 203, 261, 277
Formal specification · 29, 156, 158, 203
Formal specification techniques · 29

G

Goal · 77, 78, 112, 156, 161, 162, 163, 219

H

HCI · 1, 7, 14, 21, 23, 25, 26, 31, 68, 74, 76, 123, 136, 137, 139, 148, 167, 187, 196, 273, 274, 277
HERTs · 115, 120, 123, 124, 133, 135, 141, 142, 183, 186, 187, 193, 264
Human Computer Interaction · 7, 68, 273
Human error · 78, 118
Human Error · 20, 23, 78, 79, 92, 115, 116, 119, 120, 122, 123, 125, 154, 183, 214, 255, 264, 278, 306
Human failure · 68

I

ICO · 30, 40, 41, 43, 44, 45, 47, 65, 111, 112, 133, 135, 136, 137, 138, 139, 141, 150, 154, 158, 159, 167, 171, 174, 180, 182, 187, 190, 193, 199, 228, 244, 246, 247, 248, 249, 250, 251, 253, 254, 255, 260, 261, 262, 269, 270, 271, 274
ICOs · 29, 39, 40, 41, 43, 44, 45, 46, 47, 52, 65, 135, 136, 149, 156, 158, 159, 183, 199, 254, 260, 261, 270, 272, 273, 274
Interactive System · 1, 7, 27, 46, 50, 59, 78, 91, 99, 135, 136, 277, 278
Interface · 16, 26, 50, 74, 104, 139, 184, 185, 186, 187, 259, 261, 277

M

Method · 25, 49, 101, 140, 187, 263
Model · 1, 3, 8, 13, 14, 16, 27, 28, 32, 46, 48, 50, 62, 87, 88, 89, 91, 104, 115, 123, 127, 128, 133, 138, 144, 145, 147, 148, 149, 150, 152, 164, 165, 166, 167, 168, 169, 171, 172, 187, 188, 191, 192, 212, 214, 216, 228, 232, 233, 234, 236, 237, 248, 250, 261, 262, 277
Modelling · i, 16, 18, 20, 21, 23, 27, 28, 30, 47, 48, 66, 80, 81, 87, 100, 106, 110, 111, 112, 115, 132, 135, 136, 141, 142, 148, 155, 173, 176, 179, 180, 181, 182, 184, 190, 193, 204, 205, 233, 255, 256, 264, 270, 271, 273, 277
Models · 12, 14, 16, 47, 87, 106, 210, 211, 277, 278

N

Notation · 16, 17, 20, 21, 77, 78, 112, 156, 161, 219

O

Objects · 29, 39, 40, 43, 45, 50, 111, 135, 158, 193, 228, 254, 255, 270, 273

P

Petri net · 28, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 45, 46, 47, 64, 65, 66, 67, 94, 95, 133, 135, 136, 138, 148, 150, 157, 158, 159, 164, 165, 169, 170, 171, 172, 174, 182, 228, 229, 230, 231, 233, 234, 237, 238, 239, 246, 255, 260, 262, 263, 270, 273, 274

Petri nets · 28, 32, 33, 34, 35, 37, 39, 40, 41, 42, 45, 46, 47, 64, 65, 66, 67, 94, 95, 133, 136, 148, 150, 157, 158, 159, 164, 165, 171, 174, 182, 228, 230, 237, 246, 255, 273, 274

Places · 66, 150, 152

R

Rendering function · 40, 263

Resilience · 53, 54, 57, 58, 59, 84, 91, 277

Resilience engineering · 54, 57, 59

S

Specification · 8, 27, 50, 277

States · 10, 32, 33, 60, 62, 137, 169, 228, 235, 240

U

UML · 9

User · 2, 8, 11, 12, 13, 14, 16, 17, 20, 21, 25, 26, 50, 52, 91, 93, 96, 108, 109, 125, 132, 139, 140, 142, 165, 166, 168, 169, 170, 171, 184, 187, 188, 195, 234, 255, 256, 257, 259, 261, 263, 267, 273, 277, 278

Users · 16, 25

V

Validation · i, 30, 87, 140, 148, 164

Verification · 30, 37, 146, 150, 278

W

WIMP · 7, 13, 14, 40