Designing and Engineering Interactive Critical Systems Using Contributions from Gaming Research and Practice

Philippe Palanque, Regina Bernhaupt and Célia Martinie
IRIT, Groupe ICS
118, Route de Narbonne
31062 Toulouse, France
+33 561 55 8209
{palanque, regina.bernhaupt, martinie}@irit.fr

ABSTRACT

Integrating automation in systems is a classical approach followed by designers and engineers to support operators in the command and control tasks of more and more complex systems.

However, designing automation and designing interfaces for operating systems that are (partly) automated is a very complex activity altering in depth the development process of these systems. In the early days such automation was rather limited and thus easier to manage even though bad designs have been widely and frequently reported (e.g. [21, 24]).

Entertainment computing and gaming is a large research and industrial domain receiving an ever growing interest that has been facing such issues since the very first games. Early on games were involving autonomous objects and (later) automation-based tools for supporting (and evoking) extremely demanding and challenging cognitive, perceptive and motor activities.

This paper reports on work that has been carried out in the area of games to support the engineering of autonomous systems. It provides a set of design guidelines, processes, and evaluation techniques applicable to the domain of safety-critical interactive command and control systems.

Keywords
Games, automation, user interaction, command and control.

INTRODUCTION

Automation is one of the main means for handling increasing complexity in systems. Indeed, automation makes it possible for designers to transfer the burden of operators to a system. Thus automation is having a strong impact on workload, team size and human error. However, together with this high potential, automation raises also complex issues at all the levels of the development process, from requirements to the very final validation and verification phases.

Command and control systems have to handle, standard interactive objects as well as autonomous objects evolving in the same system but that are having potentially a complete independent manner. In addition to that aspect, designers might want to empower the operators of such systems with (partly) automated tools in order to reduce users’ workload and improve performance.

When designing user interfaces that deal with autonomous objects there are potentially two main problems that might be faced by the designer: (1) How the autonomous objects are represented in the user interface and how in general to design interfaces that enable the control of semi-autonomous objects. (2) How to design, develop, specify and evaluate user interfaces for command and control systems in which functions originally operated by a user are migrated into a partly autonomous system.

Automation has already been studied in a number of (sub-) disciplines and application fields: design, human factors, psychology, (software) engineering, aviation, health care, games [5] but the horizon for embedding them systematically into operational systems is not more than 10 years ahead. However, earlier adoptions of automation have not always been entirely successful as demonstrated by many studies in various application domains [21] or [24].

One distinguishing feature of the area of safety-critical systems is that system properties such as fault-tolerance, dependability or usability all have to be treated on an equal basis.

This paper focuses on two problem areas for the engineering of autonomous systems: first the engineering of the user interaction with autonomous systems (e.g. how to integrate autonomous behavior), especially in cases were autonomous objects are represented on the user interface. For example, the representation (on an air traffic controller screen) of an unmanned aerial vehicle (UAV) with which no direct interaction is possible, together with aircrafts with which interaction is possible via an interactive screen (and that information is sent to the pilots). Second the design and engineering of user interfaces and interaction techniques in general for (partly)-autonomous systems in charge of performing in an autonomous way tasks which were previously performed manually by operators. For instance, designing the user interface for an auto pilot in a plane or a cruise control must allow operators to set and/or control the auto pilot behavior.
PROBLEM STATEMENT
The current challenges and goals for autonomous behavior are related to the analysis, planning, decision and action cycle as described in [22]. When interacting with a (partly)-autonomous system the user interface and its underlying system should provide ways to support the operator in these activities. More precisely the operator should be supported:

• In analyzing the current context of the operations,
• in identifying one or several plans in order to carry on the current tasks or to handle unexpected events,
• in deciding amongst the various plans which one is the most appropriate,
• in inputting the plan into the system, taking into account aspects such as reusing a previous entered plan or mending an existing one that could significantly improve performance,
• in triggering the supervisory system to execute the plan which might include some degrees of autonomy (i.e. that the supervisory system has some delegated authority), and
• in following the execution of the plan allowing the operator to know what has already been performed, what is the currently executed and what will be executed in future steps.

Work has been done and is still in progress on authority sharing [7]. Of course the operator being in charge and responsible for the operations should always have the possibility to interrupt the current plan. One solution to that problem could be to reduce the operator role to the one of automation overseer and thus only acting at a high (and abstract) strategic level. Such solution would make it very difficult (and nearly impossible) for the operator to come back to a more low (and concrete) tactical level especially in case of automation degradation or system failure. Thus, other solutions have to be indentified and designed requiring scientific means to assess:

• How the operator will be able to identify (from the currently available information about the autonomous system) new plans or modification to current potential plans (or potential configurations)?
• How the operator will be able to build new plans or configurations?
• How the operator will be able to assess beforehand the impact of a potential new plan or configuration?
• How the operator will be able to interact (both monitor and possibly interrupt) with the current configuration under “execution”? This interaction aspect can be particularly complex if, in a proactive system, the configurations are executed in an autonomous way by the supervision system. Additionally such interaction should be conformant to the ones used for normal interactions.

As for solutions, one of the areas heavily using autonomous behavior is the area of games. Recent work in the area of games has shown that beyond simply doing a task, users want to engage with an interactive system, allowing them to have a playful and joyful experience [4]. User experience in games considers that the users should enjoy and engage while interacting with the computer and that engagement and fun can be one of the main goals they want to achieve.

THE GAMING PERSPECTIVE
In games the interaction is not longer only based on direct manipulation and input (i.e. the entire set of input is produced by the user having the entire initiative on the interaction), but gamers interact with a number of autonomous agents (e.g. non-player characters) that might (or sometimes not) be indirectly influenced and of which behavior is usually predefined but can be parameterized. Other interaction and usages include autonomous behaviors of game entities (e.g. for collecting resources in a game). There is a broad variety of ways to interact with these autonomous entities ranging from simple dialogue-based interaction to more complex interaction patterns, like position and movement of the (played) avatar, to influence the behavior of the non-player character (NPC).

Additionally, in order to support players in their activities, repetitive, cumbersome or boring tasks can be usually automated allowing the player to focus and invest in more rewarding, fun ones.

While the application of gaming elements to the area of safety-critical systems might sound completely contradictory, there is a long tradition of applying simulation and game approaches by military commanders in their everyday planning activities [8] Researchers in that area understood that it is of immense importance to use games in education or simulations [23].

AUTOMATION IN GAMES
We believe that the experiences in game design might help to find interaction solutions that allow displaying the critical information necessary for the operator to analyze, plan, decide, and act. Additionally, games themselves could be used as a research platform to investigate usage of the systems, identifying potential for errors and usability problems in the area of safety-critical systems.

Games offer a broad variety of interaction possibilities with autonomous objects. In the game community the development and representation of these autonomous objects is often based on what is called (by language misuse) “games artificial intelligence”. The term games artificial intelligence is abbreviated AI, but typically refers to the usage of scripts, multi-agent systems, agents or other algorithmic solutions that allow the programming of gaming elements that appear “intelligent” to the gamer. It must not be confused with the term artificial intelligence (considered as a computer science sub-discipline), we thus use in the following the term game-AI.
In the following sections we will exemplify three different types of autonomous entities in games. These three levels would be good candidates for identifying desirable levels of automation and for identifying possible user interface and interaction techniques for interacting with objects of this type in a safety-critical system.

**Full Initiative (Entity as Enemy - Fight)**

One of the game-AI approaches is a pattern-based technique that allows the entity to switch between a set of states: alert, patrol, and idle. These states are triggered by the current state of the environment and, in order to support the player, are displayed using dedicated rendering techniques.

As an example of such an approach we use the game "Metal Gear Solid" [15] embedding guards as an example of autonomous entity. The behavior of a guard is rather simple, he is walking along a predetermined path, pausing sometimes to look around by sweeping the head and then continue his patrol. This type of autonomous behavior is called patrol-based games-AI [14]. The guard has sensory capabilities like eyesight and hearing.

To support playability, the visual detection capability is presented to the player as a line of sight (LOS). On the mini-map the LOS is a cone extending from the guard's eyes (see upper right corner of Figure 1). Similarly to the LOS, the hearing of a guard is represented by a circle surrounding his location.

![Figure 1. A security guard in Metal Gear Solid](image)

The security guard in Metal Gear Solid shows one of the possible solutions for how information about an autonomous entity can be presented to the player (or the user of the system). It corresponds to the case in which *no direct interaction with the autonomous entity* is allowed. In the gaming context this form of information presentation has been shown to work efficiently and effectively, allowing the user to understand the abilities and possibilities of the guard (i.e. the autonomous entity). It also supports the players’ activity of forecasting the future behavior of the entity.

**Mixed Initiative (Entity as Partner - Collaboration)**

A second example (see Figure 2) is the usage of direct interaction with the autonomous entity. To support better understanding of the player about the capabilities of the entity and how to interact with it, such interaction is usually based on a metaphor-driven interaction.

As an example of such an approach, the game "Black and White" [6] is played from the perspective of the player being a god. The “god” player is graphically represented as a disembodied hand in the interface (see lower part of Figure 2 close to the white area). The “god” player is allowed to manage a unique creature that will serve as an assistant throughout the game. This creature (looking like a humanoid in the instance of the game presented in Figure 2) has its own behavior but can be allocated to tasks (destroying objects, collecting resources …). However, the creature has to be trained and its training has to be tuned on a regular basis. Training can be done by thanking the creature if it is behaving as expected (by scratching its belly with the “god” hand) or by punishing it (by slapping its face with the “god” hand). Doing so, the current state of the creature (corresponding to its future behavior) is presented to the user in a small cloud. In Figure 2 the creature helped automatically building and to thank it the player scratches its belly with the hand. As shown in the small cloud, the creature status for building has moved to the right (green part) meaning that now the creature is more likely to support the “god” player activity of construction.

![Figure 2. Interaction with creature in Black and White (the creature is now likely to help erecting buildings)](image)

This example presents how it is possible to interact with an automation and how it is possible, using direct manipulation techniques, to tune this autonomous behavior.
Low Initiative (Entity as Slave - Tool)
A third example shows how indirect interaction with autonomous objects can be used. Games typically have ways that allow the player to manipulate entities considered as slaves and used as (partly)-autonomous entities.

An example of such entities is the “Probes” in the game StarCraft [25] that are a kind of collecting resources unit type. These probes have the ability to gather resources automatically, to teleport buildings etc. The player can influence these autonomous entities by selecting certain settings in a menu, which are then taken into account by the autonomous entity. Other possible interaction is the direct manipulation of the drone by, for instance, selecting the drone and then right-clicking on the type of resources the player wants the drone to collect. In Figure 3 a drone is selected and shows such a menu which allows triggering an order to the probe that will perform it as a slave, until the resource is depleted, another order is triggered by the player, or the probe is destroyed.

Figure 3. Probes collecting resources in StarCraft
Such entities exhibiting low initiative could be easily used for performing repetitive operations that only require high-level monitoring from the operator. As they are present in a lot of games (especially the real time strategy ones) there have been many presentations and interaction techniques to allow players both to perceive the current plan of the entities (not in the case of StarCraft) and to interact with this plan.

A FORMAL NOTATION FOR DESCRIBING INTERACTIVE BEHAVIORS
ICO s (Interactive Cooperatives Objects) are a formal description technique dedicated to the specification of interactive systems. 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 [11] to describe their dynamics or behavior. The ICO notation is based on a behavioral description of the interactive system using the Cooperative objects formalism that describes how the object reacts to external stimuli according to its inner state. This behavior, called the Object Control Structure (ObCS) is described by means of Object Petri Net (OPN). An ObCS can have multiple places and transitions that are linked with arcs as with standard Petri nets. As an extension to these standard arcs, ICO allows using test arcs and inhibitor arcs. Each place has an initial marking (represented by one or several tokens in the place) describing the initial state of the system.

As an extension to standard Petri nets, the Cooperative Objects provide mechanisms for instantiation. Indeed, every ObCS can be instantiated and allows multiple executions of the same class as this is the case in object-oriented approaches. These instantiations can be parameterized by constructor arguments that associate a marking to the instantiation. For example, in an interaction technique exploiting several mice (such interaction may be found in interactive cockpits such as the Airbus A380), each mouse driver is a distinct instance of an ObCS class with different Class Parameters (i.e. the number of the mouse). This makes it possible for each driver to handle its own coordinates represented in the values of the tokens defining the marking of the instance.

Another important part of an ICO description is the link with the presentation part of the interactive system. This is done by means of a tabular description defining two functions, namely Rendering and Activation functions. The Rendering function describes how state changes (represented by tokens movements across the ObCS) trigger presentation methods call (which can be graphical, audio, ..). The Activation function describes how availability of particular transitions leads to availability of user events on the interface. As the paper mainly focuses on behavioral aspects, we do not describe them further (more can be found in [3], [18]).

It is important to note that ICOs have been used for many types of interfaces. The notation is supported by a CASE tool called PetShop [1], [16]. How the tool is structured and how it is integrated in a software development (and in particular with prototyping activities) goes beyond the scope of this paper that focuses on descriptive aspects of game-based interactions and interfaces. More information about that aspect is available in [20].

As explained in [10], in a user interface output is state based while input is event based. In the Petri-nets based descriptions done using ICOs, a state change only occurs when tokens of the Petri net are moved. On the opposite, events are associated to transitions making it possible to represent the relationship between user interface events
LEARNING FROM STARCRAFT FOR STATES, EVENTS AND PLANS

This section is dedicated to the description of both states and events in games. In order to make these considerations more concrete, we provide detailed examples from StarCraft II. In addition to states and events (that will be in the end captured by the ICO formal model (as presented above) we also present how plans (i.e. sequences of actions to be performed in an autonomous way by the game engine) can be defined by a set of user events and graphically represented to support their interruptability and their perception by the users.

Rendering States in Games

Among a lot of data that concretize the state of a unit, one of particular importance is the position. This position is selected by the user (providing a usually right-mouse click on the 2D scene). In Starcraft II, resources are collected by semi-autonomous objects (called probes) that are assigned to a task and then perform it iteratively (until the resource is depleted or the worker is assigned to another task). Probes can perform tasks like gathering resources, moving from a point to another or building structures. Its graphical state (represented by cyclic animations) changes when its state evolves (see a static representation in Figure 7). For instance a blue light appears in front of a probe when gathering resources (see Figure 6) while a crystal is attached to it (see Figure 5) when the resource is brought back to the Nexus. Animations allow the gamer to keep track of the activities of the probes and that the correct orders have been issued and are executed.

Figure 5: A probe carrying crystals in Starcraft.

Figure 6: A probe prospecting a crystal cluster in Starcraft.

Figure 7: A probe in its idle mode in Starcraft.

Some state changes in the game take time. Such constraints have to be made visible to the player. From the player's perspective they can be seen as autonomous tasks executions by the entity. For instance the light ray in front of the probe in Figure 6 moves up and down while collecting crystals. This animation lasts for about 10 seconds before the resource is finally collected.

Figure 4. State and Event-based behavior of a Probe
Similarly, a structure can be under construction. In this case when the mouse cursor is moved over it a progress bar is displayed to show the process advancement but can also be cancelled by clicking on the right icon. Here again, this notion of progressive state changes are used for represented autonomous behaviors but in order to favor interactivity they can usually be paused, resumed and cancelled.

Lastly, as it is explained later, recorded plans of actions of autonomous objects managed by the players will also interfere with the other events.

**Objects Behaviors Combining Events and States**

Figure 4 presents an automaton describing the behavior of a probe. Rounded squares represent the states of the probe and while arcs represent events. Such events can be triggered by users (called user-events) or by the system (game) when some conditions are true.

Between the state “Prospect” and the state “Idle” the event “Stop order received” corresponds to a user action on the probe control panel.

![Figure 8. Graphical representation of building construction in StarCraft II](image)

**User Events in Games**

Direct manipulation is the most common interaction technique used in games. As such, the common set of events produced by the user is the one accepted by the input devices i.e. click, move, drag and drop, ... for the mouse and any kind of key press combined with modifiers (CTRL, ALT, SHIFT) for the keyboard. Of course, multimodal interaction (combination of events from both mouse and keyboard) is usually provided making it possible to trigger more commands in a short period of time. Management of such multimodality remains a challenge for developers. A survey of modeling techniques addressing these issues can be found in [12].

User events will be the main triggers of state changes as the user interface must remain directed by the user. However, events triggered by the computing system (both software and hardware) might also interact with the flow of events produced by the user.

Beyond that, events can be used as a communication medium between the object directly managed by the user and the ones managed by the application. Such communication allows to makes the objects appear autonomous to the player: For instance, one probe managed by the player and one probe managed by the so-called AI in the system cannot be positioned at the same coordinates. When getting close enough a collision event is produced which will result in blocking the user from positioning the probe.

![Figure 9. Detailed behavior of resource gathering using the ICO notation](image)

**Handling plans in Games**

The initial element for the construction of a plan is a set of events produced by the player. Such plans can be considered as a set of events entered at a present time but to be executed in the future. One critical key element is the fact that, in the current state of the system such events are licit, but this might not be true when the time has elapsed. In StarCraft II it is the task of the player to be sure that the state of the system remains compatible with the plans that have been previously entered. There is a non-symmetric decision that has been made here by the game developers indeed, it is impossible to enter a plan not compatible with the current state even though this state might be made compatible with the plan by future player actions. It is hard to tell whether this was an intended design but it has been commented as “unfair” by skilled players.
In StarCraft II any command can be queued by holding down the Shift key while triggering it. This is a very simple interaction with powerful effect making it possible to define plans without going to a special editing mode. A plan can be a useful method for constructing several building or to queue orders for a given unit. For instance, a plan can be issued to move a probe along a complex path: Figure 10 present the graphical representation of a plan issued only by selecting the construction of supply depot in the probe interface and shift-clicking several positions on the scene. For example, this plan can be used to quickly order a probe to build a series of Supply Depots instead of having to re-select the unit and re-order a new Supply Depot each time.

However the plan entered can only be cancelled and not be modified. This is clearly an interaction trade-off between the powerfulness of the plan definition/modification and complexity of the user interaction. As complex interfaces will not be used by gamers who are always interacting under strong temporal constraints, the designers have opted for simple and efficient interactions with the plans.

While representing and defining plans can be seen as a challenge their actual execution by the game engine is also an issue that has to be addressed. Indeed, the plan must be executed by a dedicated plan engine in charge of intertwining the events of the plan with the ones produced by the system and by the player. This adds another flow in the game and of course another set of tests that have to be performed in order to assess the reliability of the game.

**A SAFETY-CRITICAL CASE STUDY**

This section applies the concepts presented above to a case study from the safety critical domain. We have been applying ICOs and its modeling framework PetShop to several application domains including satellite control rooms [13], commercial aircraft interactive cockpits [27] and military cockpits [2].

**Informal Presentation of the Case Study**

The case study is based on a command and control system for drones as proposed in [9]. A screenshot of such an interface (the paparazzi application) is given in Figure 11 where a group of three drones is managed at the same time. According to the classification above the drones in this application are similar to the probes in StarCraft and can be managed as slaves. In a broader context, and if considering the inclusion of such drones in an airspace, the other two types of entities have to be considered too. A completely independent entity could be a civil aircraft managed by a civil air traffic controller not reachable by the operator in charge of the drone. Only the “callsign” of the aircrafts would appear on the radar interface with no possible interaction. Military aircrafts that might be involved in the mission might not be reachable directly by the operator but the operator might be able to interact with the pilots through the military Air Traffic Controller (ATCO) in charge of the airspace. In such a case, the operator can use the user interface to send requests to the ATCO that, in turn, will send them to the pilots.

**LESSONS LEARNED AND MAIN ISSUES**

**Issues raised by the gamification of safety critical interactions**

We identified the following set of issues that must be considered if the system is (partly)-autonomous:

- what is usability in a critical context and how to evaluate it;
- how to guarantee the safety and dependability of the possible interactions;
- how to analyze and prevent operators’ errors;
- how to design and specify interaction techniques where autonomous behavior from the system interfere with operator input (including the question on how to model that formally [17], [19]);
- how to design interaction so that the operators can foresee the systems’ future steps and states;
• how to design interactions when the automation can fail and how to notify the operators;
• and how to enhance and evaluate aspects of user experience, while fulfilling the constraints of a safety-critical system which has to be secure, safe, reliable and usable.

We have applied the game approaches above to support the design and evaluation of user interfaces and interactions for safety critical (partly)-autonomous systems. This research work has started by establishing a classification of interaction possibilities that allow to support the interaction of users with (partly) autonomous objects and to establish new forms of user interaction that allow the control of autonomous objects. We do not present here such results due to space constraints but the final paper will present:

1. How to apply game elements and aspects in safety-critical systems and how to apply them on the selected case study.
2. Classification of interaction techniques and interaction concepts in games: Based on an extensive review of the games literatures as well as the topics on automation and interaction technologies in safety-critical systems a set of possible interaction concepts was identified.
3. Development of a first conceptual prototype: In the requirements specification phase we investigate how the conceptual prototype can be formally modeled including the interaction technology, answering the following questions:
   • What gaming elements have to be considered which seem to be useful in that case study?
   • How the conceptual prototype was specified (taking into account the safety-critical system)?
   • How such a case study can be modeled using formal methods (including a brief presentation on how we model interaction technologies using formal methods in the area of safety-critical systems)?

CONCLUSION AND PERSPECTIVES
The research work presented here has been trying to bring together multiple disciplines for the design and evaluation of interactive systems with entities featuring heterogeneous levels of automation.

The contributions presented here have been focusing on the current practice in the games community and have presented how some elements of this practice could be integrated in the user interface design and modeling.

An important aspect that has not been taken into account in the current paper is the possibility of occurrence of natural faults i.e. faults from the outside the system (e.g. the environment condition producing bit flips in an embedded calculator). Indeed, the current approach based on formal modeling support the detection and correction of faults that are made at design time (by the developer). First, natural faults must then be detected at execution time and mitigation or correction actions have to be undertaken in order to ensure dependability of the command and control system (see [26] for further details). Second, architectures supporting better dependability might have a significant impact on the usability of interactive application. A possible solution of this critical issues is presented in [27].

ACKNOWLEDGMENTS
This work is partly funded by ruwido research grant on user experience, R&T CNES (National Space Studies Center) Tortuga R-S08/BS-0003-029 and Eurocontrol research network HALA!

References


