A Comparative Study of RELAX and SysML/KAOS

Manzoor AHMAD, Jean-Michel BRUEL

Abstract:

This report explains two methods used for defining Non Functional Requirements (NFRs). Of the two methods one is RELAX which is a requirements engineering language; used to divide requirements into invariant and RELAX-ed. RELAX-ed requirements can be functional or non functional and is an enhanced form of the original requirement. SysML/KAOS has introduced the concept of goals in SysML; infect the meta model of SysML is extended with the concept of goals.

Existing approaches for defining NFRs**:**

In order to define non functional requirements, different questions need to be considered. Following are some of them:

- Which process to use for defining NFRs?
- Which formalism to use to specify NFRs and the links between them?
- How to treat the impact of NFRs on functional requirements?

Approach by Chung [1]:

It is based on i* approach, used to define NFRs in the form of goal graphs. Put differently a goal should be either satisfied or unsatisfied and nothing else, in every possible imaginable situation. In this context a set of "sub goals" is introduced to satisfy a given goal, where the relationship between sub goals and its parent goal is etiher *AND* or *OR*. When the relationship is *AND,* the goal is satisfied if all of its sub goals are; when the relationship is *OR*, the goal is satisfied if any of its sub goals is.

The i* model is defined by the concept of *softgoal* which is a NFR that must be satisfied [2]. The authors in [1] are of the view that it is not always possible to satisfy a goal so they have introduced the notion of softgoal for an NFR. A softgoal does not have a clear cut criteria of satisfaction. One *softgoal* is decomposed into other *softgoals* and they are linked by the contribution type *AND* or *OR*. The decomposition ends when operational *softgoals* are attained.

Approach by Cysneiros [3]:

This approach is also based on i* model, it defines how to discover NFRs and study their impact on the conceptual models. It defines a set of rules to realize the traceability between requirement analysis and the UML conceptual models.

An NFR graph is created for each NFR, where this NFR is the root of the graph. This graph is further decomposed to express all the operationalizations that are necessary to satisfy the NFR. After having these graphs, one can search for the possible interdependencies among them which can contribute positively or negatively to another NFR. For example an NFR pointing out that the software might need a high level of security may have a negative impact (a negative interdependency) on another NFR like usability.

The Two Approaches:

The next sections explain the two approaches that we have studied with the help of examples.

1. **RELAX [4]:**

RELAX is a new requirements engineering language for Dynamic Adaptive Systems (DASs), where explicit constructs are included to handle uncertainty. The need for DASs is typically due to two key sources of uncertainty: First is the uncertainty due to changing environmental conditions, such as sensor failures, noisy networks, malicious threats, and unexpected (human) input; the term environmental uncertainty is used to capture this class of uncertainty. A second form of uncertainty is behavioral uncertainty, whereas environmental uncertainty refers to maintaining the same requirements in unknown contexts, behavioral uncertainty refers to situations where the requirements themselves need to change. It is difficult to know all requirements changes at design time and, in particular, it may not be possible to enumerate all possible alternatives.

Typically textual requirements prescribe behavior using a modal verb such as *SHALL* that defines functionality that a software system must always provide. For self-adaptive systems however, environmental uncertainty may mean that it is not always possible to achieve all of those *SHALL* statements; or behavioral uncertainty may allow for trade-offs between *SHALL* statements to relax noncritical statements in favor of other, more critical ones. Therefore RELAX identifies two types of requirements: one that can be relaxed in favor of other ones called *variant* or *relaxed* and other that should never change called *invariant.*
The following requirements are taken from an AAL (Ambient Assisted Living) case study [13].

1.1. RELAXED Requirements:

1.1.1. Requirement 1:

Marry should have minimum liquid intake.

After RELAX:

Mary *SHALL* maintain liquid intake *AS CLOSE AS POSSIBLE To* ideal.

ENV: Fluid intake that is necessary

MON: Sensor enabled cups, fluid monitoring cups and faucet sensors

REL: Sensor enabled cups, fluid monitoring cups and faucet sensors all interact to collaboratively determine Mary's daily liquid intake.

1.1.2. Requirement 2:

The Fridge shall read, store and communicate RFID information on food packages.

After RELAX:

The fridge *SHALL* detect and communicate information with *AS MANY* food packages *AS POSSIBLE*.

Another variant of this requirement can be:

The fridge shall detect AS MANY information AS POSSIBLE for each food package.

ENV: Food locations, foot item information (type, calories), food state (spoiled and unspoiled)

MON: RFID readers, Cameras, Weight sensors

REL: RFID tags provide food locations and food information; Cameras provide food locations (Cameras provide images that can be analyzed to estimate food locations), Weight sensors provide food information (whether eaten or not)

DEP: R2 negatively impact R3

Negatively signifies that it will be harder to satisfy R3 while relaxing R2.

This requirement is considered RELAX-ed as the RE must ask itself if the system will not work if this requirement in not fulfilled or is it possible for the system to continue to operate at a reduced capacity e.g. if we are unable to find complete food information. Less than full capacity might help in the functioning of the system to handle an emergency situation. By declaring this requirement as RELAX-ed, the system will be flexible enough to divert some resources in this way. Thereby adapting R2; an adaptive system can balance resources in order to optimize global system parameters. The DEP field is updated as the requirements are RELAX-ed.

1.1.3. Requirement 3:

The fridge *SHALL* suggest a diet plan with total calories AS *CLOSE AS POSSIBLE TO* the daily ideal calories. The fridge *SHALL* adjust the diet plan in line with Mary's actual calorie consumption.

ENV: Mary's daily calorie consumption.

MON: RFID readers and weight sensors in fridge and trash can.

REL: RFID readers and weight sensors provide consumed items; items vanish from fridge and the items (if uneaten) or the packaging (if eaten) appears in trash can.

1.2. Invariant Requirements:

1.2.1. **Requirement 4:**

The System SHALL raises an alarm if no activity by Mary is detected within a predefined time during normal waking hours.

Discussion:

By looking at the Mary case study from the point of view of RELAX and KAOS, we can say that the *ENV, MON* and *REL* uncertainty factors of RELAX are not only giving us the same information as the Responsibility model of KAOS but rather complements it by giving us the means of how to achieve that desired functionality.

Uncertainty factors especially *ENV* and *MON* attributes are particularly important for documenting whether the system has the means for monitoring the important aspects of the environment. By collecting these *ENV* and *MON* attributes, we can build up a model of the environment in which the system will operate, as well as a model of how the system monitor its environment. Sources of uncertainty can include: contention for resources, adverse environmental conditions, timing of events and the duration of conditions.

Once the requirements engineer determines that a certain level of flexibility can be tolerated in requirements then it is up to the downstream developers including designers and developers to incorporate the most suitable adaptive mechanisms to support the required functionality.

2. Goal Oriented Techniques for Requirements Engineering

2.1. Why Use Goal Oriented Techniques for Requirements Engineering [5]:

There are a number of claims of advantages made from GORE (Goal Oriented Requirements Engineering) literature [6], Following is the summary:

2.1.1. Requirement Completeness and Pertinence

Goals enable the sufficient completeness and pertinence of a requirements specification.

2.1.2. Rationale for Requirement

A requirement exists because it satisfies its higher goals. Any requirement which does not contribute to any goal will not be considered at all. For this reason every requirement will have a rationale for it. Explaining requirements to stakeholders is another important issue. Goals provide the rationale for requirements.

2.1.3. Traceability

Goal graphs provide traceability links like from low level requirements to high level objectives and from organizational to business context.

2.1.4. Conflict Management

Contributions among goals (positive or negative) can be modeled and managed. In this way conflicts can be identified and resolved.

2.1.5. Managing Requirements Evolution

The higher-level a goal is the more stable it is likely to be. Goals are thus essential elements for managing requirements evolution.

2.2. SysML and KAOS:

SysML and KAOS have some advantages and weak points, but these are complementary to each other based on the following points:

- *Requirements description: A* textual description in *SysML* and a description in the form of goals in *KAOS*.
- *Relation between requirements: SysML* has *contain* and *derive* relations; these relations do not have a precise semantics which leads into confusion. *KAOS* has refinement relations AND/OR.
- *Traceability relations: Satisfy* and *verify* relations in *SysML* allow to define traceability. *KAOS* does not have explicit relations.
- *Tools:* A number of tools exist for *SysML*; most of them are open source. *KAOS* propose a tool *Objectiver* which is proprietary.

2.3. Why SysML/KAOS?

In KAOS, non functional properties are taken into account only at the architectural level. Due to the complexity of the systems, non functional properties should be processed much more early; at the same level of abstraction as functional properties which will allow taking into account these properties for the evaluation of alternate options, risk and conflict analysis.

2.4. SysML/KAOS Approach [7]:

Its main objective is to implement the extended meta model and to provide an environment for modeling functional and non functional requirements. Functional requirements are treated in [8], in this work non functional requirements are treated and the impact of these on functional requirements. The main idea is to extend the SysML language with the most relevant concepts of commonly used requirements engineering approaches.

Figure 1 shows the extended meta model of SysML/KAOS. The meta-class ***NON FUNCTIONAL GOAL*** represents the non functional goal, it is specified as a sub-class of the meta-class *GOAL* which itself is a sub-class of the meta-class *REQUIREMENT* of *SysML*. A *Non Functional Goal* represents a quality that the future system must have in order to satisfy a functional requirement.

The ***NFGTYPE*** specify the type of NFR and the attribute ***TOPIC*** that represent the domain concept concerned by this type of requirement. An NFR can thus be represented with the following syntax: ***NFGType [Topic]***. (Mary case study example)

An **NFG** can either be an abstract goal (meta-class ***Abstract NFG***), or an elementary goal (meta-class ***Elementary NFG***). An abstract **NFG** can be refined further into sub-goals. The hierarchies of goals are modeled using ***Refinement*** association which becomes an A*ssociation Class* between the abstract goal and its sub-goals. A goal that cannot be further refined is an *Elementary Goal*. Once the refinement is done, the next step is to identify different solutions to attain the elementary goals, which will allow to choose from these solutions the one that could be implemented in the system to be developed.

A *Contribution Goal* (meta-class ***Contribution Goal***) is a third type of goal that is used to operationalize an elementary goal as shown in the **figure 1**. For example, the elementary goal *Confidentilité* which can be satisfied by using the contribution goals *using a PIN code* or *use an additional identifier*. These two contribution goals are associated to the *Confidentilité* elementary goal.

Refinement of **NFG** can be performed on two basis: ***NFG Type*** and ***Topic***. The **NFG** *Securité* is refined (***AND Refinement***) on the basis of ***NFG Type*** into three sub goals: *Disponibilité, Intégrité and Confidentialité* as shown in the **figure 2**. The ***NFG Disponibilité*** is refined (***OR Refinement***) into two sub goals on the basis of ***Topic.*** The ***NFG Bonne Precision*** is refined (***AND Refinement***) into two sub goals on the basis of ***Topic.*** The sub goal ***Confidentilité*** can be satisfied by a ***positive*** and ***direct*** contribution from one of the ***Contribution Goals*** i.e. using PIN Code, compare the signatures and add an additional identifier.

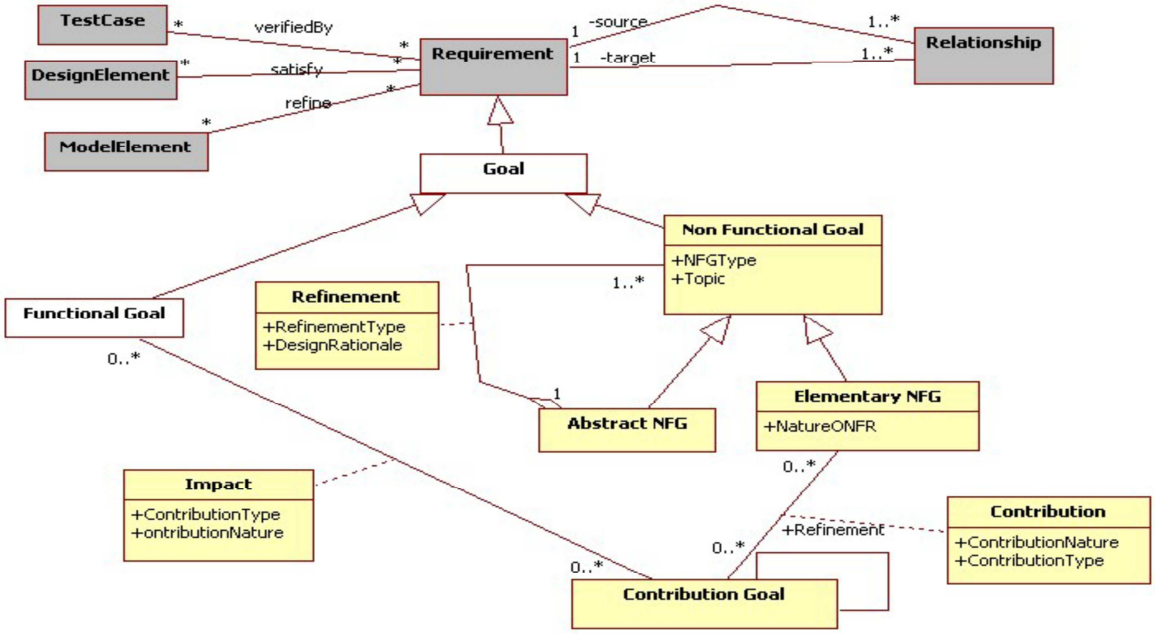

Fig. 1 Extended SysML Meta Model

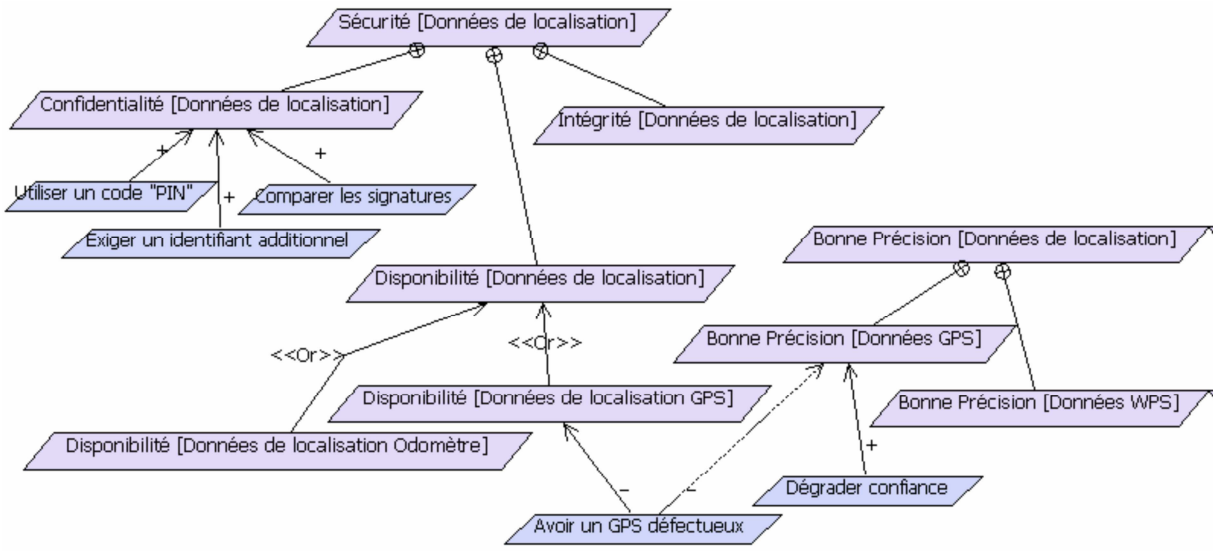

Fig. 2 Model of the Localisation Non Functional Goals

3. Our Hypothesis:

From the discussion above we have deduced three hypothesis.

3.1. Hypothesis 1:

In order to correlate the two approaches we know that a *<<Block>>* satisfy *<<Requirement>>* and an *<<Agent>>* satisfy *<<Goal>>*, so we can deduce a hypothesis from this correlation that a *<<Requirement>>* corresponds to a *<<Goal>>* and a *<<Block>>* corresponds to an *<<Agent>>.*

This hypothesis can be verified by the work of Chung [1] and Cysneiros [2] which shows that non functional requirements can be formulated in the form of goals.

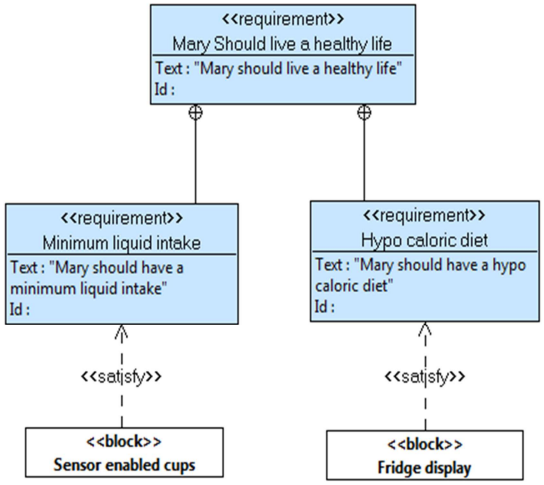

Fig. 3 SysML Requirement Diagram

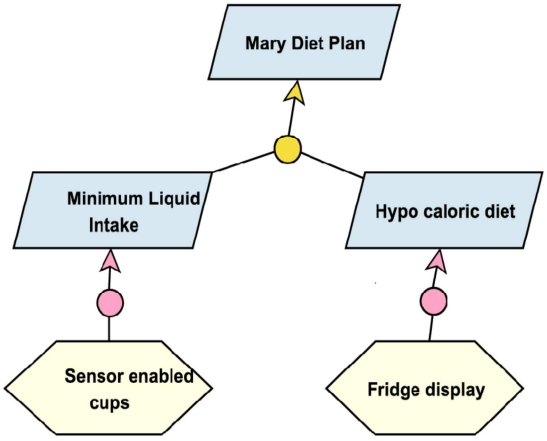

Fig. 4 KAOS Responsibility Model

This correlation can be found in figures 3 and 4. **Figure 3** shows the SysML requirement diagram. The main requirement is to keep mary healthy; this requirement is composed of two requirements i.e. <<*Minimum liquid intake*>> which is satisfied by the block <<*Sensor enabled cups*>> and <<*Hypo caloric diet*>> which is satisfied by the block <<*Fridge display*>>. **Figure 4** shows the KAOS responsibility model: here the main goal is to look after the mary diet plan which is decomposed into two sub goals. The first sub goal is the *Minimum liquid intake* which is *achieved* by the *agent Sensor enabled cups* and the other sub goal is *Hypo caloric diet* which is achieved by the agent *Fridge display*.

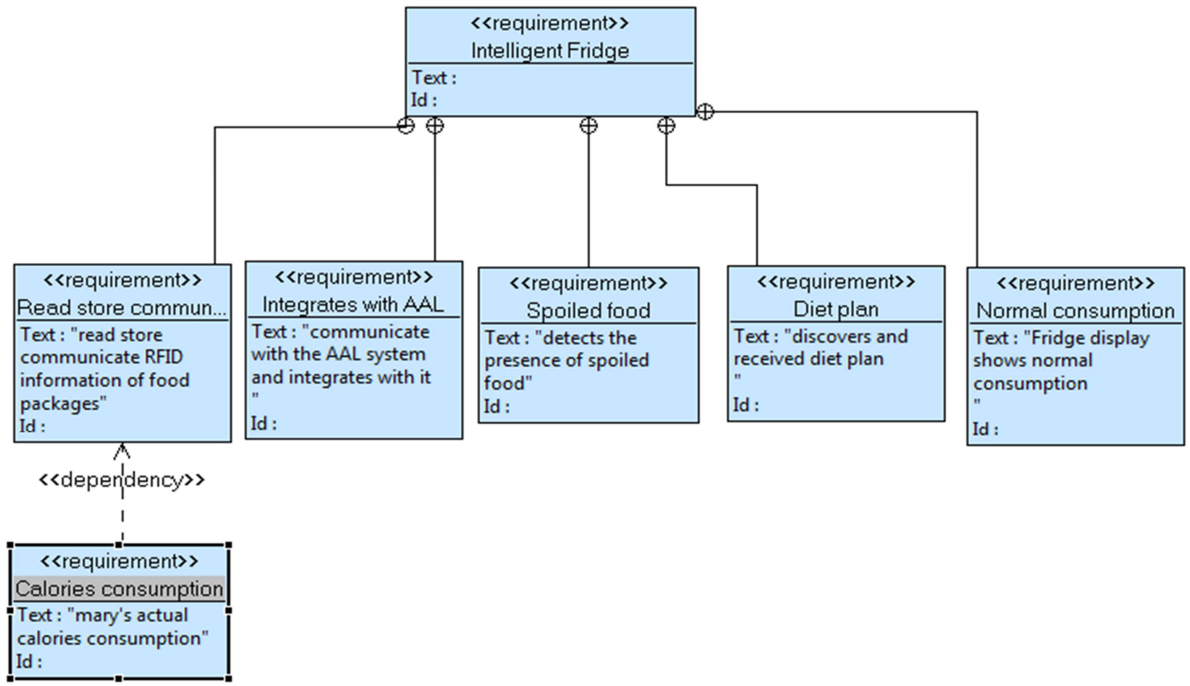

Fig. 5 Dependency b/w Requirements

Figure 5 shows the concept of dependency. Requirement <<Calories consumption>> is dependent on requirement <<Read store communicate RFID info>>, this requirement is treated as RELAXE-ed. By relaxing this requirement it negatively impacts the other requirement; as to calculate the actual

calories consumption we need to have all the information of food packages, which can be achieved only by having every means to read, store and communicate RFID information on food packages.

3.2. Hypothesis 2:

If a top level goal is RELAX-ed then its sub goals can be RELAX-ed or Invariant and if a top level goal is Invariant then its sub goals are invariant.

The example of Confidentiality in SysML/KAOS approach, it is considered as a RELAX-ed goal as we can achieve this goal by many ways:

2. Using PIN Code
3. Compare the signatures
4. Add an additional identifier

Based on the second case, confidentiality is treated as a RELAX-ed requirement as we need some kind of relaxation in order to identify the signature.

3.3. Hypothesis 3:

Another important aspect of RELAX is that the ENV, MON and REL attributes will be particularly interesting in building the SysML parametric diagrams so we can for example use mathematical equations to implement these attributes in the parametric diagram.

4. **Relationship b/w SysML/KAOS, SysML and RELAX:**The following table shows relationship between different concepts dealt by SysML, SysML/KAOS and RELAX.

Concepts	**SysML/KAOS**	**SysML**	**RELAX**
Requirements Description	Goals	Textual Requirements	Enhanced Version of Textual Requirements
Relationship	AND, OR	<<verify>> <<refine>>	REL
Dependency/Impact	Contribution Nature: Positive Negative Contribution Type: Direct (Explicit) Indirect (Implicit) **b/w NFG and FG**	<<derive>> <<contain>>	DEP: Positive Negative
Monitoring	<<contribution goal>>	<<satisfy>>	MON
Tools	Eclipse based SysML/KAOS Editor	Eclipse/Papyrus/Topcased/ …..	Eclipse based COOL RELAX editor

Table 1. Relationship between different Concepts

The table above shows different concepts handled by each approach. In SysML/KAOS; requirements are described in the form of goals, SysML describes requirements in textual form while RELAX requirements are also in textual form with an enhanced version i.e. requirements divided into invariant and RELAX-ed requirements with uncertainty factors added to it. SysML/KAOS has no AND/OR refinement relationships, SysML has verify and refine relationships while for RELAX we have REL variable which identifies the relationship between *ENV* and *MON*. For dependency/Impact SysML/KAOS deals with it from the point of view of the impact of non-functional goal on functional goal; this impact can be positive or negative and direct or indirect while for SysML we have the concept of derive which shows the dependency between requirements, RELAX has the positive and negative dependency. To deal with monitoring SysML/KAOS has the contribution goal concept which is used to satisfy a non-functional goal, SysML has satisfy; used when a block satisfies a requirement while for RELAX we have the concept of *MON* which is used to measure the environment i.e. *ENV*. SysML/KAOS has a tool called SysML/KAOS editor, SysML has a number of tools e.g. eclipse [9], papyrus [10], topcased [11] etc and for RELAX we have eclipse based COOL RELAX editor [12].

Conclusion:

After studying the two approaches, we can conclude that these two approaches are complementary for each other. We can take benefits from goal oriented approaches in defining requirements for self adaptive systems. The worth of goal oriented techniques for requirements engineering in general can be found in literature [5] [6]. RELAX can complement goal oriented approaches by providing more details in the form of *ENV, MON* and *REL* while in the same way goal oriented approaches can complement RELAX in defining requirements for self adaptive systems with the help of precise definition of positive/negative and direct/indirect impacts.

REFERENCES:

1. Chung et al., *Non-functional Requirements in Software Engineering*. Kluwer, 1999.

2. Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos (festschrift), LNCS volume 5600. Springer, 2009. 530 pp. ISBN 978-3-642-02462-7.

3. Cysneiros et al., Nonfunctional Requirements: From Elicitation to Conceptual Models. *IEEE TSE*, May 2004.

4. Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng and Jean-Michel Bruel RELAX: a language to address uncertainty in self-adaptive systems requirement, RE'09 Special Issue

5. Axel Van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", Fifth IEEE International Symposium on Requirements Engineering (RE'01) Toronto, 249-263.

6. Shahzad Anwer, Naveed Ikram, "Goal Oriented Requirement Engineering: A Critical Study of Techniques," 13th Asia Pacific Software Engineering Conference (APSEC'06), pp.121-130.

7. Christophe Gnaho, Farida Semmak: Une extension SysML pour l'ingénierie des exigences non fonctionnelles orientée but. Ingénierie des Systèmes d'Information 16(1): 9-32 (2011)

8. Laleau R., Semmak F., Matoussi A., Petit D., Hammad A., Tatibouet B., A First Attempt to Combine SysML Requirements Diagrams and B. *In Innovations Systems and Software Engineering journal*. Springer-Verlag 2010.

9. http://www.eclipse.org/

10. http://www.papyrusuml.org

11. http://www.topcased.org/

12. S. Gatti, J. Geisel, S. Labondance, J. Pages, Internal Report M2ICE, UTM 2011.

13. http://www.iese.fraunhofer.de/fhg/iese/projects/med_projects/aal--lab/index.jsp