Modeling and verification of functional and non functional requirements of ambient, self adaptative systems

Manzoor Ahmad

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par
l'Université Toulouse II - Le Mirail
Discipline ou spécialité :
Informatique

Présentée et soutenue par
Manzoor AHMAD
Le 07-*10-2013*

Titre :
Modeling and Verification of Functional and Non Functional Requirements of Ambient, Self-Adaptive Systems

---

JURY
Charles CONSEL - Professor, University of Bordeaux , France (Examiner)
João ARAÚJO - Assistant Professor, Universidade Nova de Lisboa, Portugal (Examiner)
Pierre-Jean CHARREL - Professor, University of Toulouse, France (Member)
Régine LALEAU - Professor, Université Paris-Est Créteil, France (Member)
Jean-Michel BRUEL - Professor, University of Toulouse, France (Supervisor)
Nicolas BELLOIR - Maître de Conférences, UPPA Pau, France (Co-Supervisor)

Ecole doctorale : Mathématiques Informatique Télécommunications (MITT)
Unité de recherche : Institut de Recherche en Informatique de Toulouse (IRIT)
Directeur(s) de Thèse : *Jean-Michel BRUEL et Nicolas BELLOIR*
Rapporteurs : *João ARAÚJO et Charles CONSEL*

Abstract

The context of this research work is situated in the field of Software Engineering for Self Adaptive Systems. Software Engineering applies a systematic approach to the development, operation, and maintenance of a software product using a sequence of activities. Our work resides in the very early stages of the software development life cycle i.e. at the Requirements Engineering phase. We focus on requirements elicitation, modeling and verification aspects of the Requirements Engineering process.

Due to the continuous growth in size, complexity, heterogeneity and the inherent uncertainty of systems, it becomes increasingly necessary for computing based systems to dynamically self adapt to changing environmental conditions, these systems are called Self Adaptive Systems. These systems modify their behavior at run-time in response to changing environmental conditions. In order to take into account the changing environmental factors, Requirements Engineering languages must deal with the inherent uncertainty present in these systems, which we can capture during the early phases of its development life cycle. Requirements provide the basis for estimating costs and schedules as well as developing design and testing specifications. Changes identified during the later stages of the software development life cycle are hard to reflect and impacts the cost, schedule, and quality of the resulting software product. For Self Adaptive Systems, Non Functional Requirements play an important role, and one has to identify as early as possible those Non Functional Requirements that are adaptable. Non Functional Requirements are requirements that are not specifically concerned with the functionality of a system, they place restrictions on the product being developed and the development process.

The overall contribution of this thesis is to propose an integrated approach for modeling and verifying the requirements of Self Adaptive Systems using Model Driven Engineering techniques. Model Driven Engineering is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem level abstractions to software implementations. By using these techniques, we have bridged this gap through the use of models that describe complex systems at multiple levels of abstraction and through automated support for transforming and analyzing these models. We take requirements as input and divide it into Functional and Non Functional Requirements. We then use a process to identify those requirements that are adaptable and those that cannot be changed. We then introduce the concepts of Goal Oriented Requirements Engineering for modeling the requirements of Self Adaptive Systems, where

Non Functional Requirements are expressed in the form of goals which is much more rich and complete in defining relations between requirements. We have identified some problems in the conventional methods of requirements modeling and properties verification using existing techniques, which do not take into account the adaptability features associated with Self Adaptive Systems. Our proposed approach takes into account these adaptable requirements and we provide various tools and processes that we developed for the requirements modeling and verification of Self Adaptive Systems. We validate our proposed approach by applying it on two different case studies in the domain of Self Adaptive Systems.

Acknowledgement

Completing my PhD degree is probably the most challenging activity of my first 32 years of life. The best and worst moments of my doctoral journey have been shared with many people. It has been a great privilege to spend several years in MACAO (IRIT) at University of Toulouse, and its members will always remain dear to me.

Foremost, I would like to express my sincere gratitude to my advisor Prof. Jean-Michel BRUEL for the continuous support during my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. I could not have imagined having a better advisor and mentor for my Ph.D study. I would also like to express my sincere gratitude to my co-advisor Nicolas BELLOIR. Their guidance helped me in all the time of research and writing of this thesis.

Besides my advisors, I would like to thank the thesis committee: Prof. Charles CONSEL, Assistant Prof. João ARAÚJO, Prof. Pierre-Jean CHARREL and Prof. Régine LALEAU for their encouragement, insightful comments, and questions. It is no easy task, reviewing a thesis, and I am grateful for their thoughtful and detailed comments.

I would like to thank Cristophe GNAHO and Farida SEMMAK whose work on goal oriented requirements engineering and the discussions that we have had inspired us to integrate it in our research work.

I thank my fellow lab mates specially Iulia DRAGOMIR for the healthy discussion that we have had on properties verification.

I also offer my gratitude to HEC (Higher Education Commission) of the Government of Pakistan which financed this thesis in its early days.

I would like to thank my parents, their love provided me inspiration and was my driving force. I owe them everything and wish I could show them just how much I love and appreciate them.

I would like to thank my family and friends: my brother and sisters who were always there to encourage and motivate me.

Last but not the least, I would like to thank my wife Faiza AHMAD whose love and encouragement allowed me to finish this journey. She was always there cheering me up and stood by me through good and bad times.

Contents

List of Figures

CHAPTER 1

Introduction

Contents

1.1 Problem Statement

Self Adaptive Systems (SAS) are highly adaptive, they modify their behavior at run-time in response to changing environmental conditions. For these systems, Non Functional Requirements (NFRs) play an important role, and one has to identify as early as possible those requirements that are adaptable. The distributed nature of SAS and changing environmental factors (including human interaction) makes it difficult to anticipate all the explicit states in which the system will be during its lifetime. As such, an SAS needs to be able to tolerate a range of environmental conditions and contexts, but the exact nature of these contexts remains imperfectly understood. A feature common to all the previous works regarding Requirements Engineering (RE) for SAS is that they assume that all adaptation choices are known and enumerated at design time and does not take unanticipated adaptations into account. The fact that the point of adaptation for SAS should be identified as early as possible drive the research on these systems. We base our hypothesis on how to identify the point of adaptation while defining the requirements for these systems and how to verify these requirements while keeping in mind the adaptability associated with these systems.

We have identified some problems associated with requirements modeling and verification using existing approaches. On one hand, existing requirements modeling approaches do not take into account the uncertainty factors associated with SAS, on the other, hand we are exposed to the state space explosion problems associated with existing properties verification techniques for these systems.

1.2 Objectives of the Thesis

The main objective of this thesis is to provide an integrated approach to identify the point of flexibility as early as possible while defining the requirements of SAS and to provide a mechanism for verifying the requirements of these systems. We are of the view that, on one hand, requirements for SAS should consider the notion of uncertainty while defining it, on the other hand, there should be a way to verify these requirements as early as possible, even before the development of these systems starts. In order to handle the notion of uncertainty in SAS, RE languages for these systems should include explicit constructs for identifying the point of flexibility in its requirements [96]. Goal Oriented Requirements Engineering (GORE) techniques can be used to define and model the requirements of SAS [37, 100, 58, 99] and we aim to provide an integrated approach where we can define and model the requirements of these systems to obtain a detailed description of the system and its environment.

We base our proposition on using Relax [96] which is an RE language for SAS and which helps in the differentiation of those requirements that can be Relax-ed from those requirements that can not be changed. For SAS, Relax-ed requirements play an important role as these are associated with the adaptability features of these systems. We then integrate Relax with SysML/Kaos [36] which is a GORE approach for modeling the requirements of SAS. Then we aim to provide an efficient approach for verifying the properties of these

systems. The concepts of Model Driven Engineering (MDE) are used at each step of our proposed approach as it is the main theme of our research team [1] and also due to the benefits associated with it [33]. MDE is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations.

Our proposed approach comprises of some tools and processes to address the problems defined above, which are listed below.

1. To bridge the gap between requirements and the overall system model, we provide a Domain Specific Language (DSL) for Relax, which takes requirements in textual format as input and transform it into SysML requirements diagram.
2. To automate the requirements of SAS by taking into account the different uncertainty factors associated with each Relax-ed requirement, we provide a tool called Relax COOL editor.
3. To take benefits of GORE approaches, we provide a correlation between Relax and SysML/Kaos concepts.
4. To use SysML/Kaos in the context of SAS, we provide a mechanism so that only Relax-ed requirements are injected into SysML/Kaos and not the NFRs in general.
5. To model the Relax-ed requirements in the form of goal models, these requirements are transformed into SysML/Kaos goal concepts using the correlation table.
6. The validation of our integrated approach by applying it on Ambient Assisted Living (AAL) and barbados Car Crash Crisis Management System (bCMS) case studies.
7. To verify some properties of the AAL against the system design, we adapt the profile and the toolset used for verification to take into account the adaptability features associated with SAS requirements.

The choice of Relax is motivated by the fact that it avoids the problem posed by unanticipated adaptations, i.e. by specifying declaratively the ways in which a requirement may be Relax-ed. Relax does not require all possible alternative adaptations to be specified during RE. This flexibility leaves open the design choice as to how to achieve adaptation and therefore supports designs based on adaptation rules, planning algorithms, control theory algorithms, etc. This work resides in the framework of self adaptation but we do not treat the development of self adaptation mechanisms as we are at the very early stage of the SAS development life cycle.

1.3 Structure of the Thesis

The thesis is organized as follows. In chapter 2, I give a description of the main context of our work i.e. what is SAS and how it differs from other systems, I then give few examples of SAS. In chapter 3, I give a description of Software Engineering (SE) and what are its

1. http://www.irit.fr/-Equipe-MACAO-

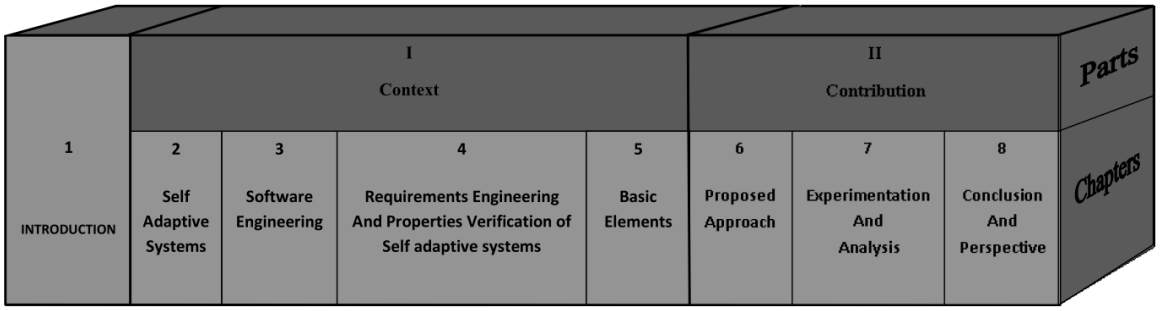

Figure 1.1: Thesis Chapters Organization

activities, then I introduce MDE along with its basic concepts and show why is it useful to use MDE techniques for SAS. As this thesis is centered around defining an integrated approach for the RE of SAS, I describe the different approaches of RE. GORE techniques are detailed as we base our proposition on using these techniques for defining and modeling the requirements of SAS. chapter 4 gives a description of the state of the art of the different approaches i.e. RE for SAS and properties verification using formal methods. In chapter 5, I describe Relax, which is an RE language for SAS and SysML/Kaos which is an extension of the SysML requirements model with concepts of the Kaos goal model. Both Relax and SysML/Kaos serve us as a starting point for proposing our integrated approach. I then introduce OMEGA2/IFx profile and toolset that we used for the properties verification and model simulation of the SAS. OMEGA2 is an executable Uml/SysML profile used for the formal specification and validation of critical real-time systems. It is based on a subset of Uml 2.2/SysML 1.1 containing the main constructs for defining the system structure and behavior. The OMEGA2 Profile is supported by the IFx toolset which provides mechanisms for the model simulation and properties verification of SAS. chapter 6 gives an overall view of our proposed approach, defining the various processes, tools and documents used in it. I start with introducing the reasons that motivated us to propose this approach by showing some problems that we identified with existing approaches. I then explain the need for an integrated approach which takes into account the different features that we propose and to provide an integrated tooling environment for dealing with the identified problems. I then give a description of the tools that we developed in order to validate the proposed approach. The tools comprises of: a DSL for Relax which is acting as a bridge between the requirements and overall system model and using this DSL, we have transformed requirements in textual format to a graphical format i.e. SysML requirements diagram using Relax grammar, a Relax editor which is capabale of automating the Relax-ed requirements by taking into account the uncertainty factors associated with each Relax-ed requirement, a Relax2SysML/Kaos editor which models the requirements of the case studies that we have worked with and which is the result of the application of transformation rules from Relax-ed requirements into SysML/Kaos goal concepts. In order to validate the proposed approach, chapter 7 gives a description of the two case studies i.e. AAL and bCMS, and the modeling of some of its requirements. I then show the verification of some of the properties of the AAL case

study using OMEGA2/IFx profile and toolset. This chapter also shows an assessment of the proposed approach in terms of the problems identified and the solution provided by using it. chapter 8 shows the summary of this thesis and the perspectives that we identified. Figure 1.1 shows the organization of the parts and chapters of this thesis.

Part I

CONTEXT

CHAPTER 2

Self Adaptive Systems

Contents

2.1 What are Self Adaptive Systems?

An adaptive system is a set of interacting or interdependent entities, real or abstract, forming an integrated whole that together are able to respond to environmental changes or changes in the interacting parts. A system has goals that must be satisfied and, whether these goals are explicitly identified or not, system requirements should be formulated to guarantee goal satisfaction. This fundamental principle has served systems development well for several decades but is founded on an assumption that goals are fixed. In general, goals can remain fixed if the environment in which the system operates is stable [95].

As applications continue to grow in size, complexity, and heterogeneity, it becomes increasingly necessary for computing based systems to dynamically self adapt to changing environmental conditions. These systems are called Dynamic Adaptive Systems (DAS) [96]. Example applications that require DAS capabilities include automotive systems, telecommunication systems, environmental monitoring, and power grid management systems. The distributed nature of DAS and changing environmental factors (including human interaction) makes it difficult to anticipate all the explicit states in which the system will be during its lifetime. As such, a DAS needs to be able to tolerate a range of environmental conditions and contexts, but the exact nature of these contexts remains imperfectly understood. One overarching challenge in developing DAS therefore, is how to handle the inherent uncertainty posed by the respective application domains.

2.2 How Self Adaptive Systems Differs from other Systems?

The complexity of current software systems has led the SE community to investigate innovative ways of developing, deploying, managing and evolving software-intensive systems and services. In addition to the ever increasing complexity, software systems must become more versatile, flexible, resilient, dependable, energy-efficient, recoverable, customizable, configurable, and self-optimizing by adapting to changes that may occur in their operational contexts, environments and system requirements. Therefore, self-adaptation:

> *"systems that are able to modify their behavior and/or structure in response to their perception of the environment and the system itself, and their goals"*

has become an important research topic in many diverse application areas.

It is generally accepted that errors in requirements are very costly to fix [62]. The avoidance of erroneous requirements is particularly important for the emerging class of systems that need to adapt dynamically to changes in their environment. Many such DAS are being conceived for applications that require a high degree of assurance [85], in which an erroneous requirement may result in a failure at run-time that has serious consequences. The requirement for high assurance is not unique to DAS, but the requirement for dynamic adaptation introduces complexity of a kind not seen in conventional systems where adaptation, if it is needed at all, can be done off-line. This dynamic adaptation consequent complexity is

manifested at all levels, from the services offered by the run-time platform, to the analytical tools needed to understand the environment in which the DAS must operate.

DAS are systems designed to continuously monitor their environment and then adapt their behavior in response to changing environmental conditions. DAS tend to be cyber-physical systems, where the physical environment is tightly intertwined with the computing-based system. For these systems, the software may need to be reconfigured at run time (e.g., software uploaded or removed) in order to handle new environmental conditions [20].

RE is concerned with what a system ought to do and within which constraints it must do it. RE for SAS, therefore, must address what adaptations are possible and what constrains how those adaptations are carried out. In particular, questions to be addressed include: what aspects of the environment are relevant for adaptation? Which requirements are allowed to vary or evolve at run-time and which must always be maintained? In short, RE for SAS must deal with uncertainty because the expectations on the environment frequently vary over time.

The need for DAS 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. IBM, for example, originally proposed the area of autonomic computing [48] to handle environmental uncertainty, thereby enabling computing-based systems to use high-level application goals and requirements to guide run-time self-management, including self-monitoring, self-healing, and self-configuration. 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. For example, the requirements of a space probe may change mid flight in order to pursue science opportunities not foreseen by the designers. It is difficult to know all requirements changes at design time and, in particular, it may not be possible to enumerate all possible alternatives [96].

An increasingly significant requisite for software based systems is the ability to handle resource variability, ever-changing user needs and system faults. Certain standard programming practices, such as capacitating extensive error handling capabilities through exception catching schemes, do contribute towards rendering systems fault-tolerant or self-adaptive, however, these methods are tightly coupled with software codes and are highly application-specific [34].

2.3 Self Adaptive Systems Examples

SAS like AAL are common in industrialized countries. Below, I cite few examples of such SAS.

- The Aware Home Research Initiative (AHRI) [1] at Georgia Institute of Technology is an interdisciplinary research endeavor aimed at addressing the fundamental technical, design, and social challenges for people in a home setting. Researchers at the AHRI

1. http://awarehome.imtc.gatech.edu/

are interested in three main research areas: Health and Well-being, Digital Media and Entertainment, and Sustainability, investigating how new technologies can impact the lives of people at home.
- Maison Intelligent IUT de Blagnac[2] provides services especially home support to people i.e. to provide dependent and disabled people a reliable service that allows them to live almost independently in its familiar environment and with the constant safety as continuously assisted by an adapted infrastructure, potentially making use of human assistance.
- Mary AAL home[3]: Mary is a patient and she need hypo caloric diet. She needs to maintain minimum liquid intake and the various sensor enabled tools helps her maintain her daily intake of food and water. Advanced smart homes, such as Mary's AAL, rely on adaptivity to work properly.
- DiaSuiteBox[4] is an Open application platform for smart homes. It proposes an application store that gathers the devices deployed at home. The DiaSuiteBox platform runs an open-ended set of applications leveraging a range of appliances and web services.

We use Mary's AAL home case study for the validation of our proposed approach. We model the requirements and then verify some of the properties of the AAL case study using our proposed approach.

2.4 From Requirements Discovery to Adaptation Choices of Self Adaptive Systems

Goal based modeling notations has been applied to the discovery and specification of DAS requirements [96] such as i* [98]. Goal models have proven to be useful for specifying the adaptation choices that a DAS must make [37, 58] as well as for the specification of monitoring and switching between adaptive behaviors [81]. A particular strength of goal based modeling is that it supports the modeling of non-functional trade-offs, which can be used to capture some elements of environmental uncertainty. [59] show that requirements goal models can be used as a foundation for designing software that supports a space of behaviours, all delivering the same function, and that is able to select at runtime the best behaviour based on the current context. The advantages of this approach include the support for traceability of software design to requirements as well as for the exploration of alternatives and for their analysis with respect to quality concerns of stakeholders.

2.5 Conclusion

In this chapter, I show a description of the context of our work i.e. SAS. Then, I describe the specificity of these systems and show how these systems differs from other systems. I

2. http://mi.iut-blagnac.fr/
3. http://www.iese.fraunhofer.de/fhg/iese/projects/medprojects/aal-lab/index.jsp
4. http://phoenix.inria.fr/software/diasuitebox

then cite few examples of SAS as these systems like Mary's AAL are developed to facilitate the daily lives of the patient. The technologies show promise in helping people to live independently and in comfort. As our work is about the RE of SAS, I talk about the very initial stages of the SAS life cycle as we are interested in providing a framework for self adaptation but not any mechanisms for achieving it.

CHAPTER 3

Software Engineering

Contents

In this chapter, I introduce SE and its main activities. I then introduce the MDE concepts which we used to develop the tools support for our proposed approach. Our proposed approach takes requirements as input, so I give a description of the different techniques used for RE. GORE are described in detail as these are most widely used for the RE of SAS.

3.1 Software Engineering

Software systems are abstract and intangible. They are not constrained by the properties of materials, governed by physical laws, or by manufacturing processes. This simplifies SE, as there are no natural limits to the potential of software. However, because of the lack of physical constraints, software systems can quickly become extremely complex, difficult to understand, and expensive to change. This is why they need specific engineering processes and techniques to take into account these considerations.

3.1.1 What is Software Engineering

SE is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use. Following are the definitions of SE:

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software [43].
- The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines [67].

A good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.

3.1.2 Origin of Software Engineering

The notion of SE was first proposed in 1968 at a conference held to discuss what was then called the *Software Crisis* [67]. It became clear that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late. Throughout the 1970's and 1980's, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding and object-oriented development. Tools and standard notations were developed and are now extensively used.

3.1.3 Activities of Software Engineering

The systematic approach that is used in SE is sometimes called a *software process*. A *software process* is a sequence of activities that leads to the production of a software product [86]. There are four fundamental activities that are more or less common to all software processes. I describe it in the following sub sections.

3.1.3.1 Software Specification

Software specification or RE is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. RE is a particularly a critical stage of the *software process* as errors at this stage inevitably lead to later problems in the system design and implementation. The RE process in Figure 3.8 aims to produce an agreed requirements document that specifies a system satisfying stakeholder requirements. Requirements are usually presented at two levels of detail. End-users and customers need a high-level statement of the requirements; system developers need a more detailed system specification. I describe the RE activities in section 3.3.4.

3.1.3.2 Software Design and Implementation

A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used. Designers do not arrive at a finished design immediately but develop the design iteratively. They add formality and detail as they develop their design with constant backtracking to correct earlier designs. Depending upon the system, it may not be possible to do constant backtracking as in the case of system engineering of complex systems.

The implementation stage of software development is the process of converting a system specification into an executable system. The development of a program to implement the system follows naturally from the system design processes. Although some classes of programs, such as safety-critical systems, are usually designed in detail before any implementation begins, it is common for the later stages of design and program development to be interleaved. Software development tools may be used to generate a skeleton program from a design. This includes code to define and implement interfaces, and, in many cases, the developer need only add details of the operation of each program component.

3.1.3.3 Software Validation

Software validation or, more generally, verification and validation is intended to show that a system both conforms to its specification and that it meets the expectations of the system customer. Program testing, where the system is executed using simulated test data, is the principal validation technique. Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development. Because of the predominance of testing in general, the majority of validation costs are incurred during and after implementation.

A three-stage testing process is often used: first the system components are tested then the integrated system is tested and, finally, the system is tested with the customer's data. Ideally, component defects are discovered early in the process, and interface problems are found when the system is integrated. However, as defects are discovered, the program must

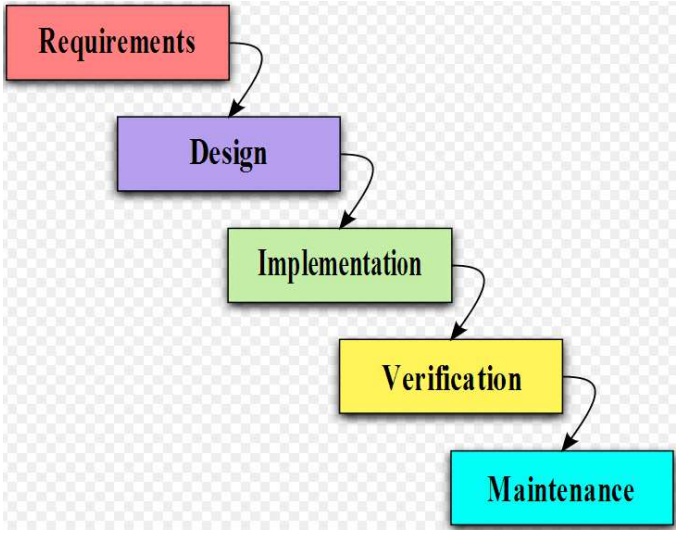

Figure 3.1: The Software Engineering Process [86]

be debugged and this may require other stages in the testing process to be repeated. Errors in program components may come to light during system testing. The process is therefore an iterative one with information being fed back from later stages to earlier parts of the process.

3.1.3.4 Software Evolution

The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems. Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design. However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware. However, it does not mean that changes at software level are cheaper, according to [39, 40], errors related to RE activity of the SE process are more costlier and even results in the failure of software projects.

The different activities of SE varies extensively with the type of software system being developed. Different types of software systems exists ranging from simple embedded systems to complex, worldwide information systems, SAS etc. It is pointless to look for universal notations, methods, or techniques for SE because different types of software require different approaches. Developing an organizational information system is completely different from developing an SAS. Neither of these systems has much in common with a graphics-intensive computer game. All of these applications need SE; they do not all need the same SE techniques. While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of systems. Therefore, one can not say which method is better than another.

Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes. General process models

describe the organization of software processes. Examples of these general models include the 'waterfall' model, incremental development, and reuse-oriented development. The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. Figure 3.1 shows the four software process activities. In our context, we are interested in using MDE techniques for the requirements modeling and verification of SAS, so I describe the MDE concepts in the following section.

3.2 Model Driven Engineering

MDE aims at shifting the focus of software development from coding to modeling [97]. The term MDE is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations [33]. MDE software development approach has the potential to address the identified challenges of SE. It offers an environment that ensures the systematic and disciplined use of models throughout the development process of software systems. The essential idea of MDE is to shift the attention from program code to models. This way models become the primary development artifacts that are used in a formal and precise way [61].

A significant factor behind the difficulty of developing complex software systems e.g. SAS is the wide conceptual gap between the problem and the implementation domains of discourse [33]. Bridging the gap using approaches that require extensive handcrafting of implementations gives rise to accidental complexities that make the development of complex software difficult and costly. MDE is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations. The complexity of bridging the gap is tackled through the use of models that describe complex systems at multiple levels of abstraction and from a variety of perspectives, and through automated support for transforming and analyzing models. In the MDE vision of software development, models are the primary artifacts of development and developers rely on computer-based technologies to transform models to running systems.

3.2.1 General Principles of Model Driven Engineering

3.2.1.1 What is a Model?

In literature, different definitions of a model exists [66, 82, 52, 19], at various levels of abstraction, introducing conceptual frameworks or pragmatic tools, describing languages or environments, discussing practices and processes. Modeling is now permeating all fields of SE.

> *"A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system. [19]"*

The answers provided by the model should be the same as those given by the system itself, on the condition that questions are within the domain defined by the general goal of

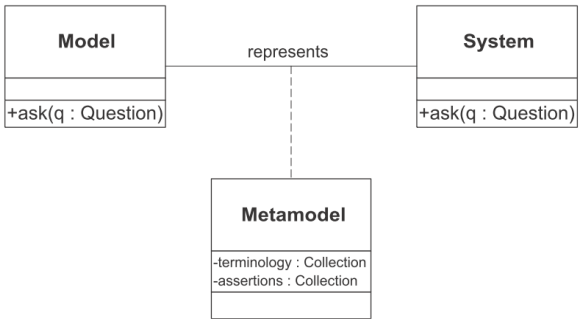

Figure 3.2: Relationship between System and Model [19]

the system. In order to be useful, a model should be easier to use than the original system. To achieve this, many details from the source system are abstracted out, and only a few are implemented in the target model. This simplification or abstraction is the essence of modeling. Following are the characteristics of useful models [83].

- Abstract: Emphasize important aspects while removing irrelevant ones.
- Understandable: Expressed in a form that is readily understood by observers.
- Accurate: Faithfully represents the modeled system.
- Predictive: Can be used to answer questions about the modeled system.
- Inexpensive: Much cheaper to construct and study than the modeled system.

3.2.1.2 The Concept of Meta Model

A meta-model is the explicit specification of an abstraction (a simplification). It uses a specific language for expressing this abstraction: e.g. Meta Object Facility (MOF) [1]. In order to define the abstraction, the meta-model identifies a list of relevant concepts and a list of relevant relationships between these concepts. In some cases this may suffice, but in many situations it needs to be completed by a set of logical assertions. With MOF, a specific formalism for the assertions must be added like Object Constrained Language (OCL) [2]. Figure 3.2 illustrates relationships between systems, models and meta-models.

> *"A meta-model is a collection of concepts, it defines the set of well-formed productions to represent a given reality."*

A model *M* is said to *ConformTo* a given meta-model *MM* if it can be obtained through a legal collection of concepts as defined by meta-model *MM*. The organization of the classical four level architecture of the Object Management Group (OMG) [3] should more precisely be named a 3+1 architecture as illustrated in Figure 3.3 [18]. At the bottom level, the *M0* layer is the real system. A model represents this system at level *M1*. This model conforms to its meta-model defined at level *M2* and the meta-model itself conforms to the meta-meta-model at level *M3*. The meta-meta-model conforms to itself.

1. http://www.omg.org/mof/
2. http://www.omg.org/spec/OCL/
3. http://www.omg.org

3.2.1.3 Model Driven Architecture

In 1997, OMG created the UML [4] standard which is a general purpose modeling language. The OMG is an international, open membership, computer industry standards consortium. Founded in 1989, OMG standards are driven by vendors, end-users, academic institutions and government agencies. The UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. It has become one of the most used modeling languages.

In 2000, the OMG proposed the Model Driven Architecture (MDA) [5]. MDA engulfs the definition of several standards like MOF, OCL, Query View Transformation (QVT) [6] and XML Metadata Interchange (XMI) [7]. The need for MOF resulted from the fact that UML was only one of the meta-models in the software development landscape. Because of the risk posed by the presence of a variety of different, incompatible meta-models being defined and evolving independently (data warehouse, work flow, software process, etc.), there was an urgent need for a global integration framework for all the meta-models in the software development industry. The solution was therefore a language for defining meta-models, i.e. a meta-meta-model. This is the role of the MOF. As a consequence, a layered architecture has now been defined. This layered architecture has the following levels.

- M3: the meta-meta-model level (contains only the MOF)
- M2: the meta-model level (contains any kind of meta-model, including the UML meta-model)
- M1: the model level (any model with a corresponding meta-model from M2)
- M0: the concrete level (any real situation, unique in space and time, represented by a given model from M1)

3.2.1.4 Model Transformations

MDE ensures that models are formally defined and precise, thus a partial automation of the software development process can be achieved. It is commonly accepted that automation is by far the most effective technological means for boosting productivity and reliability [83]. The automated parts of the development process are achieved through model transformations. According to [50]:

> *"A model transformation is the automatic generation of one or more target models from one or more source models, according to transformation definition(s). A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language . A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language."*

4. http://www.omg.org/spec/UML/2.4/Infrastructure/Beta2/PDF/
5. http://www.omg.org/mda/
6. http://www.omg.org/spec/QVT/
7. http://www.omg.org/technology/xml/

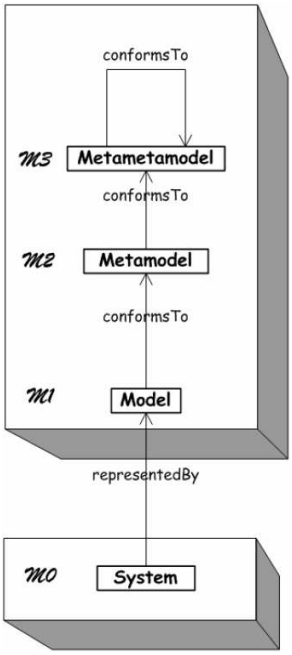

Figure 3.3: The 3+1 MDA Architecture [18]

In a nutshell, a transformation define a mapping between a source and a target model. MDE aims to automate many of the complex but routine development tasks which still have to be done manually today [9] with model transformations. Figure 3.4 shows a snapshot of the model transformation from Model *Ma* to Model *Mb* [44].

There exists many classes of model transformations, [63] proposes a taxonomy of model transformations. In order to transform models, these models need to be expressed in some modeling language (e.g., Uml for design models, and programming languages for source code models). The syntax and semantics of the modeling language itself is expressed by a meta-model (e.g., the Uml meta-model). Based on the language in which the source and target models of a transformation are expressed, a distinction can be made between *endogenous* and *exogenous* transformations. *Endogenous* transformations are transformations between models expressed in the same language. *Exogenous* transformations are transformations between models expressed using different languages. Another distinction to be made is between *horizontal* and *vertical* transformation. A *horizontal* transformation is a transformation where the source and target models reside at the same level of abstraction. A typical example of *horizontal* transformation is refactoring. A *vertical* transformation is a transformation where the source and target models reside at different levels of abstraction. A typical example is refinement, where a specification is gradually refined into a full-fledged implementation by means of successive refinement steps that add more concrete details.

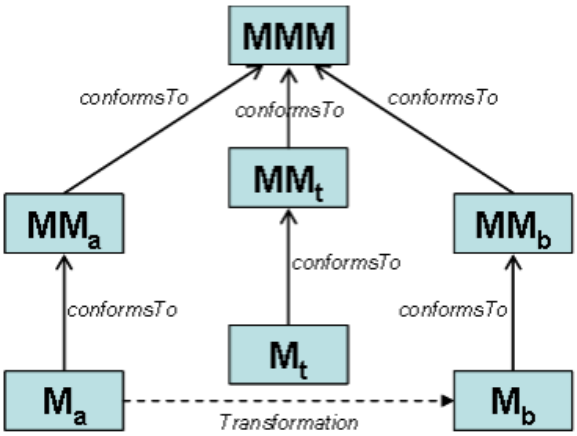

Figure 3.4: Model Transformation Example [44]

3.2.2 Why Model Driven Engineering for Self Adaptive Systems?

The major advantage of using MDE is that we express models using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain relative to most popular programming languages. This makes the models easier to specify, understand, and maintain. It also makes models less sensitive to the chosen computing technology and to evolutionary changes to that technology.

Research on MDE has mainly focused on the use of models during software development. This has produced relatively mature techniques and tools that are currently being used in industry and academia to manage software complexity during software development. Works on SAS has produced significant results, but many problems remain. In our proposed approach, we provide tools and processes for the requirements modeling and verification of SAS. MDE techniques help us to develop these various tools. It provides a mechanism through automated support for models transformation and analysis to accomplish our desired objectives.

3.3 Requirements Engineering

RE emphasizes the use of systematic and repeatable techniques that ensure the completeness, consistency, and relevance of the system requirements [87].

3.3.1 The Importance of Requirements Engineering

Various studies of the Standish Group [8] state that inappropriate RE is one of the most important reasons for project failures. The frequently cited *Standish Group Report* from 1995 [39] reports that only 52.7% of the projects analyzed in the study were finished, but they exceeded the estimated budget by up to 189%. Moreover, on average only 42% of the planned

8. http://standishgroup.com

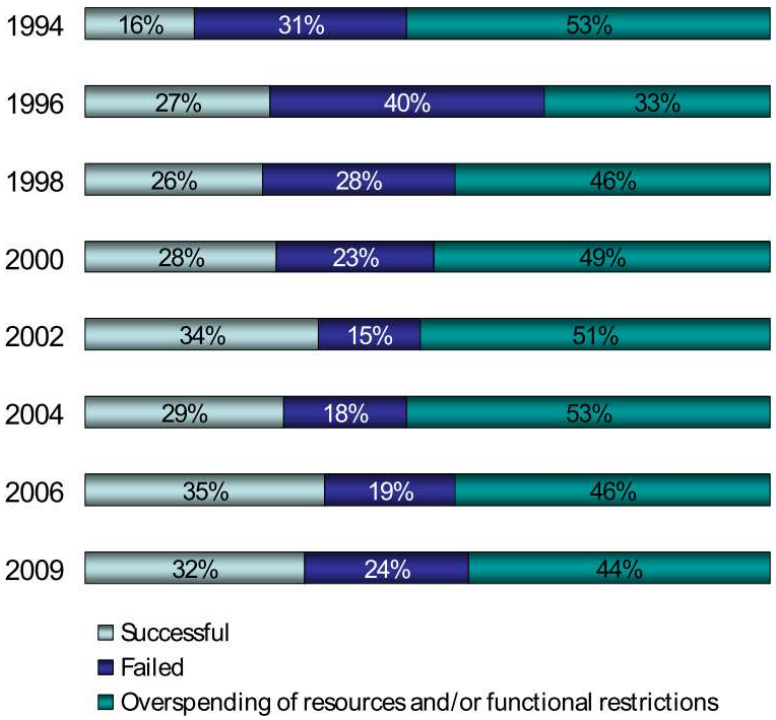

Figure 3.5: Projects Success Rates 1994 to 2009 [40]

system functions were implemented. Of the projects, 16.1% were finished on time, within budget and realized all the planned system functions. Of all projects, 31.1% were canceled and did not deliver the desired results.

Figure 3.5 provides an overview of the project success rates from 1994 to 2009 [40]. It shows that the percentage of successfully finished projects in 2009 was significantly higher than in 1994. However, the value has been stagnating since 1996 at a rate around 30%. In all the studies summarized in Figure 3.5 at least 65% of the projects failed or finished with an overspend above the estimated resources and/or with restricted implemented functionality. Thus the situation has not changed significantly since 1996. Figure 3.6 depicts the different reasons and the percentage of each reason stated by the project participants. Reasons that are definitely related to insufficient and poor requirements engineering are highlighted in dark grey. Together, they sum up to 48%.

3.3.2 Definition of Requirement

The importance of complete, consistent and well documented software requirements is difficult to overstate [31]. As requirements are the starting point of every software development process. The IEEE 610.12-1990 standard [43] defines the term "requirement" as follows:

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.

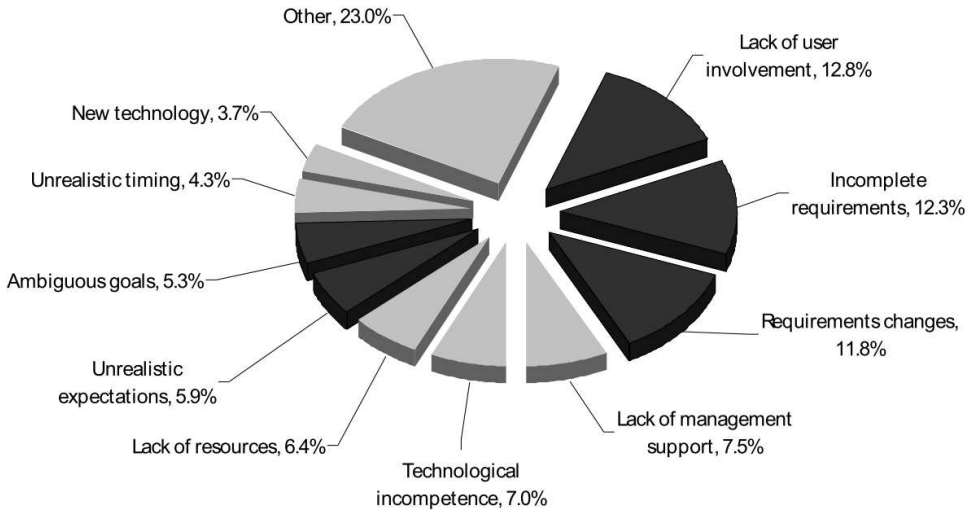

Figure 3.6: Reasons for Resource Overspend [39]

– A documented representation of a condition or capability as in the first two forms.

3.3.3 Functional Requirements v/s Non Functional Requirements

Requirements can be divided into functional and non functional requirements. IEEE 610.12-1990 standard [43] defines a Functional Requirements (FR) as a requirement that specifies a function that a system or system component must be able to perform. There is no such consensus for defining NFRs [35]. Figure 3.7 gives an overview of selected definitions from literature or the web, which are representative of the definitions that exist.

3.3.4 Activities of Requirements Engineering

RE needs several activities in order to provide a consistent requirements model. Specifically, RE encompasses the following activities.

1. Feasibility study: An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.
2. Requirements elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. It helps to understand the system to be specified.
3. Requirements specification: Requirements specification is the activity of translating the information gathered during the analysis activity into a document that defines a set

Source	Definition
Antón [1]	Describe the nonbehavioral aspects of a system, capturing the properties and constraints under which a system must operate.
Davis [3]	The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability.
IEEE 610.12 [6]	Term is not defined. The standard distinguishes design requirements, implementation requirements, interface requirements, performance requirements, and physical requirements.
IEEE 830-1998 [7]	Term is not defined. The standard defines the categories functionality, external interfaces, performance, attributes (portability, security, etc.), and design constraints. Project requirements (such as schedule, cost, or development requirements) are explicitly excluded.
Jacobson, Booch and Rumbaugh [10]	A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement.
Kotonya and Sommerville [11]	Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet.
Mylopoulos, Chung and Nixon [16]	"... global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. (...) There is not a formal definition or a complete list of nonfunctional requirements."
Ncube [17]	The behavioral properties that the specified functions must have, such as performance, usability.
Robertson and Robertson [18]	A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property.
SCREEN Glossary [19]	A requirement on a service that does not have a bearing on its functionality, but describes attributes, constraints, performance considerations, design, quality of service, environmental considerations, failure and recovery.
Wiegers [21]	A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior.
Wikipedia: Non-Functional Requirements [22]	Requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors.
Wikipedia: Requirements Analysis [23]	Requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

Figure 3.7: Non Functional Requirement Definitions [35]

of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.

4. Requirements validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

The activities in the requirements process are not simply carried out in a strict sequence. Requirements analysis continues during definition and specification and new requirements come to light throughout the process. Therefore, the activities of analysis, definition, and specification are interleaved. Figure 3.8 shows the different activities of the RE process.

3.4 Approaches of Requirements Engineering

In literature, different approaches exist for RE. Each one of these approaches concentrate on a specific activity of the RE process. Following are some of the approaches that we have studied.

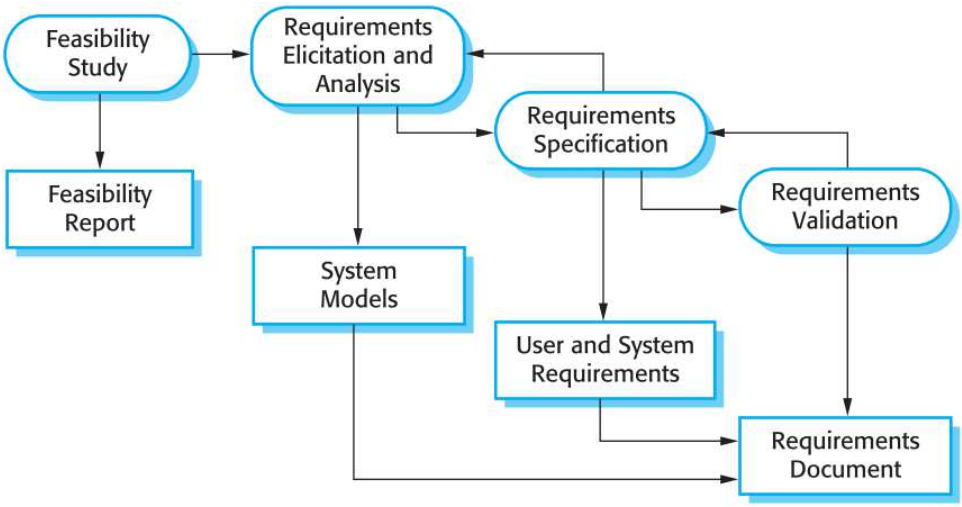

Figure 3.8: The Requirements Engineering Process [86]

3.4.1 Scenario Based Requirements Engineering

In Scenario Based RE [88, 79], requirements are described in the form of scenarios. A scenario can be defined as a description of a possible set of events that might reasonably take place. The main purpose of developing scenarios is to stimulate thinking about possible occurrences, assumptions relating these occurrences, possible opportunities and risks, and courses of action [46]. Scenarios have been advocated as an effective means of communicating between users and stakeholders and anchoring requirements analysis in real world experience. Unfortunately scenarios are extremely labor-intensive to capture and document; furthermore, few concrete recommendations exist about how scenario-based RE should be practiced, and even less tool support is available [88].

3.4.2 Aspect Oriented Requirement Engineering

Aspect Oriented Requirements Engineering (AORE) [78] is based on an improved support for separation of crosscutting functional and non-functional properties during RE which results in an improved understanding of the problem and ability to reason about it. AORE approaches adopt the principle of separation of concerns at the analysis phase (the early separation of concerns). In other words, AORE approaches provide a representation of crosscutting concerns in requirements artifacts. These approaches also focus on the composition principle: it should be possible to compose each concern/requirement with the rest of the concerns/requirements of the system under construction to understand interactions and trade-offs among concerns. This composability of requirements is a central notion of AORE [64]. It allows not only reviewing the requirements in their entirety, but also detection of potential conflicts very early in order to either take corrective measures or appropriate decisions for the next development step. The mapping of the concerns at the requirement

level to concerns in later life cycle stages reveal whether the concern maps to a crosscutting artifact or whether it becomes absorbed into other artifacts.

3.4.3 Problem Frame Requirements Engineering

Problem Frame Requirements Engineering [45] is based on the assumption that *You can't solve a software development problem until you understand it*. Problem Frames offer a systematic, realistic way to grasp the other half of the development challenge i.e. the problem that must be solved. This approach is mainly valid for functional requirements.

Problem Frames are a systematic approach to the decomposition of problems that allows us to relate requirements, domain properties, and machine specifications. Having decomposed a problem, one approach to solving it is through a process of composing solutions to sub-problems. Given a good decomposition of a problem into sub-problems, a range of benefits would arise if we were able to provide a solution by solving the sub-problems in isolation and then composing the partial solutions to give a complete system. The benefits include: scalability (due to working at the level of simpler sub-problems), traceability (since sub-problems map directly to their solutions), and easier system evolution (changes to sub-problems can be addressed by modifying corresponding solutions or compositions) [57].

3.4.4 Goal Oriented Requirement Engineering

The birth of the GORE approach goes back more than two decades with the precise work of K. Yue [101], who was the first to assert that the explicit modeling of goals in requirements models may provide a criterion for requirements completeness. [27, 55] define a goal as:

> *"An objective that the system and its environment must achieve through the cooperation of different agents (hardware, software or human)."*

A goal placed under the responsibility of a system agent is called a *requirement*, while a goal placed under the responsibility of an agent of the system environment is called an *expectation*. The following main activities are normally present in most of the GORE approaches: goals elicitation, refinement of goals, different types of goals analysis and the attribution of the responsibility for a goal to an agent. In what follows, I present a range of methods adopting the GORE paradigm.

3.4.4.1 The Non Functional Requirement Framework

The Non Functional Requirement (NFR) framework [22] is a framework of goal modeling. The analysis begins with softgoals that represent NFRs which stakeholders agree upon. Softgoals are goals that are hard to express, but tend to be global qualities of a software system. These could be usability, performance, security and flexibility in a given system. These softgoals are then usually decomposed and refined to uncover a tree structure of goals and sub-goals e.g. the flexibility softgoal. Once uncovering tree structures, one is bound to find interfering softgoals in different trees, e.g. security goals generally interferes with

usability. These softgoal trees now form a softgoal graph structure. The final step in this analysis is to pick some particular leaf softgoals, so that all the root softgoals are satisfied.

3.4.4.2 The i* Framework

The i* framework [98] proposes an agent-oriented approach to RE centering on the intentional characteristics of the agent. Agents attribute intentional properties (such as goals, beliefs, abilities, commitments) to each other and reason about strategic relationships. Dependencies between agents give rise to opportunities as well as vulnerabilities. Networks of dependencies are analyzed using a qualitative reasoning approach. Agents consider alternative configurations of dependencies to assess their strategic positioning in a social context.

The i* framework was developed for modeling and reasoning about organizational environments and their information systems. It consists of two main modeling components. The Strategic Dependency (SD) model, which is used to describe the dependency relationships among various actors in an organizational context and the Strategic Rationale (SR) model, which is used to describe stakeholder interests and concerns, and how they might be addressed by various configurations of systems and environments. The framework builds on a knowledge representation approach to information system development.

3.4.4.3 Goal Based Requirements Analysis Method (GBRAM)

The GBRAM [6] is useful to identify, elaborate, refine and organize goals for requirements specification. GBRAM comprises two activities: analysis and refinement of goals. Analysis of the goals is to explore the sources of information to identify goals, organize and classify them. The interest of GBRAM is that it makes the distinction between achievement goals (Functional Goal (FG)) and maintenance goals (Non Functional Goals (NFGs)). The goals refinement activity concerns the goals evolution from the time they are identified until they are translated into operational requirements. Throughout the goals refinement activity, GBRAM defines the precedence relation which is to find goals that precede other goals. All GBRAM concepts (goals, Agents, stakeholders) are specified only in textual form in goals patterns, without providing any graphical notation.

3.4.4.4 Attributed Goal-Oriented Requirements Analysis Method

Attributed Goal-Oriented Requirements Analysis Method (AGORA) [47] is an extended version of the goal-oriented requirements analysis method, where attribute values, e.g. contribution values and preference matrices, are added to goal graphs. An analyst attaches contribution values and preference values to edges and nodes of a goal graph respectively during the process for refining and decomposing the goals. The contribution value of an edge stands for the degree of the contribution of the sub-goal to the achievement of its parent goal, while the preference matrix of a goal represents the preference of the goal for each stakeholder. These values can help an analyst to choose and adopt a goal from the alternatives

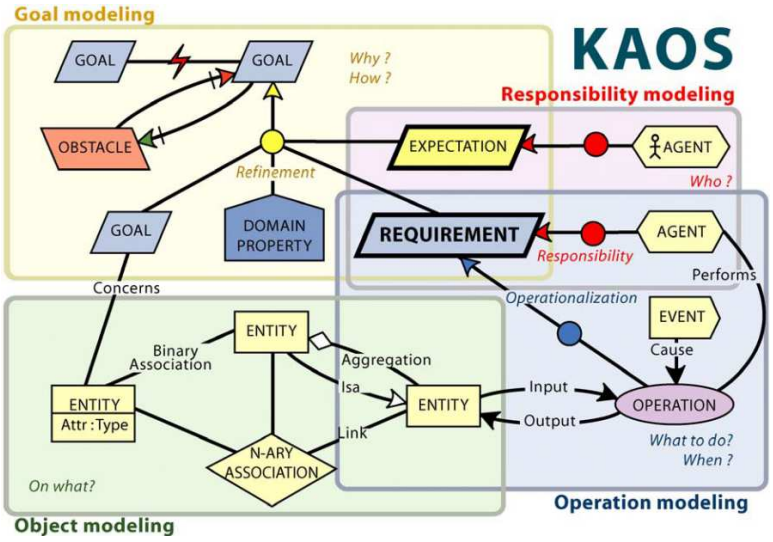

Figure 3.9: KAOS Models

of the goals, to recognize conflicts among goals, and to analyze the impact of requirements changes. Furthermore the values on a goal graph and its structural characteristics allow the analyst to estimate the quality of the resulting requirements specification, such as correctness, unambiguity, completeness etc. The estimated quality values can suggest which goals should be improved and/or refined.

3.4.4.5 ESPRIT CREWS

The ESPRIT CREWS [79] approach aims to discover requirements through a bi-directional coupling between goals and scenarios. Three characteristics of the proposed approach contribute to the achievement of this objective: First, a bidirectional goal-scenario coupling. The second characteristic of the approach is the distinction between the refinement relationship and the AND/OR relationships among goals. The third characteristic is the methodological support provided in the form of enactable guiding rules embodied in a computer software environment called L'Ecritoire. As a result, it is possible to guide the requirements elicitation process through interleaved goal modeling and scenario authoring. Here, the focus is on the discovery of goals from scenarios. The discovery process is centered around the notion of a Requirement Chunk (RC) which is a <Goal, Scenario> pair.

3.4.4.6 The Knowledge Acquistion in autOmated Specification (KAOS) Method

Kaos [55] is a methodology for RE enabling analysts to build requirement models and to derive requirement documents from Kaos models. Kaos is based on goals treated at a high level of abstraction. The Kaos language is a multi-paradigm language used for modeling goals in order to specify requirements in an RE process. It consists of several models, which are connected to each other through rules of inter-model consistency.

KAOS serves in building a model for the requirements which helps in describing the problem to be solved and the constraints that must be fulfilled by any solution provider. KAOS helps in: problem description by allowing to define and manipulate concepts relevant to problem description, to improve the problem analysis process by providing a systematic approach for discovering and structuring requirements, to clarify the responsibilities of all project stakeholders and to let the stakeholders communicate easily and efficiently about the requirements.

Goals are desired system properties that have been expressed by some stakeholder. Using KAOS, the analyst discovers the new system by interviewing current and future users and by analyzing the existing systems, reading the available technical documents. KAOS enables the analyst to structure the collected goals into directed, acyclic graphs so that each goal in the model (except the roots, the top most strategic goals) is typically justified by at least another goal that explains why the goal was introduced in the model and that each goal (except the leaves, the bottom goals) is refined as a collection of sub-goals describing how the refined goal can be reached. A KAOS model aggregates four complementary and interrelated views on the information system [9].

- The goal model refers to the goals wished by the stakeholders involved in the information system, i.e., the owners, the users, the business managers, regulations etc. and the requirements on the information that are needed in order to achieve these goals.
- The responsibility model refers to the inventory of human and automated agents located inside the system or belonging to its environment and to whom responsibility for achieving the requirements is assigned.
- The object model defines the problem domain in terms of domain concepts and relationships over those concepts.
- The operation model defines the behavior that those agents must undertake in order to achieve or maintain the requirements for which they are responsible.

Figure 3.9 shows the four KAOS models.

3.4.5 Discussion

The most important RE approaches of recent years are goal oriented. This success is due to the fact that when the GORE process finish its work, all the other approaches starts from there [56]. Goals concentrate on the activities before the requirements formalization [70]. Most of the other approaches of RE concentrate on what the software should do and how to do it but not on the reasoning about the requirements. Consequently, it will be difficult afterwards to understand the requirements and to judge whether, they really capture the stake holders needs.

Traditional approaches of RE focus on the specification of the system-to-be alone and do not consider its environment, which is one of the reason of the inadequacy of these approaches when dealing with more and more complex systems. In the context of this thesis, we aim to provide an integrated approach for defining and modeling the requirements of SAS,

9. http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf

thats the reason why traditional approaches of RE can not cope with the uncertainty posed by these systems, however, consideration of the environment is critical for SAS. Moreover, an SAS is treated as a collection of target systems with varying environmental conditions, so each target system's requirements are modeled, and the adaptive logic that serves for transition between configurations are treated as separate concerns [103]. Therefore, we need an approach that can provide support for reasoning about alternative system configurations where different solutions can be explored and compared. GORE approaches try to solve these issues. They take into account stakeholders' intentions and make use of goal models for specifying these intentions [54]. Various studies [37, 100, 58, 99] suggest that goals can be used to systematically model the requirements of a SAS.

The choice of Kaos [55] is motivated by the fact that it permits the expression of several models (goal, agent, object, behavioral models) and the relationship between them. It also provides a powerful and extensive set of concepts to specify goal models. This facilitates the design of goal hierarchies with a high level of expressiveness that can be considered at different levels of abstraction.

3.5 Conclusion

SE is concerned with all the phases of software development i.e. from software specification till maintenance of the resulting software. SE engulfs different activities while a software is developed i.e. software specification, software design and implementation, software validation and software evolution. Software specification or more specifically, RE is one of the most important activity of SE. Different technical reports have shown that errors at this stage of the software development life cycle are the most costlier to be addressed in the later stages i.e. it results in resource overspending and in some cases even in the failure of software. In literature, we have different approaches of RE. GORE approaches have gained success due to the use of goal models where we can easily identify the stakeholders involved, resolve conflicts, assign responsibility to agents. Due to the specific needs of SAS, we need an approach that best takes into account the uncertainty posed by SAS and which must support alternative system configurations, that are provided by GORE approaches. MDE techniques are permeating all the fields of SE. We take benefit of MDE techniques for the development of tools in order to validate our propose approach.

CHAPTER 4

Requirements Engineering & Properties Verification of Self Adaptive Systems

Contents

In this chapter, I describe the state of the art of the different approaches of requirements modeling and properties verification.

Different road map papers on SE for SAS [21, 28] discusses the state of the art, its limitations, and identify critical challenges. [21] presents research road map for SE of SAS focusing on four views, which are identified as essential: requirements, modeling, engineering, and assurances. The focus is on development methods, techniques, and tools that seem to be required to support the systematic development of complex software systems with dynamic self adaptive behavior. The most recent road map paper [28] discusses four essential topics of self-adaptation: design space for self-adaptive solutions, software engineering processes for self-adaptive systems, from centralized to decentralized control, and practical run-time verification and validation for SAS.

Traditionally, requirements documents make statements such as *The system shall do this*. For SAS, the prescriptive notion of *shall* needs to be relaxed and could, for example, be replaced with *The system may do this or it may do that* or *If the system cannot do this, then it should eventually do that*. This idea leads to a new requirements vocabulary for SAS that gives stakeholders the flexibility to account for uncertainty in their requirements documents. RE for SAS, therefore, must address what adaptations are possible and what constrains how those adaptations are carried out. In particular, questions to be addressed include: what aspects of the environment are relevant for adaptation? Which requirements are allowed to vary or evolve at run-time and which must always be maintained. In short, RE for SAS must deal with uncertainty because the expectations on the environment frequently vary over time.

4.1 Requirements Engineering for Self Adaptive Systems

An SAS is able to modify its behavior according to changes in its environment. As such, an SAS must continuously monitor changes in its context and react accordingly. But here the questions arises as what aspects of the environment should the SAS monitor. Clearly, the system cannot monitor everything. And exactly what should the system do if it detects a less than optimal pattern in the environment. Presumably, the system still needs to maintain a set of high level goals that should be maintained regardless of the environmental conditions. But non critical goals could well be Relax-ed, thus allowing the system a degree of flexibility during or after adaptation. These questions (and others) form the core considerations for building SAS. This section gives an overview of the state of the art regarding RE for SAS.

4.1.1 Levels of Requirement Engineering for Modeling

Levels of Requirement Engineering for Modeling (LoREM) [37] is an approach for modeling the requirements of a DAS using i* goal models [98]. This approach reifies each of the original levels to describe the work performed by a specific type of DAS developer to produce i* goal models that are intended to be integrated with those produced by the other developers. The i* goal models are used to represent the stakeholder objectives, non-adaptive system behavior (business logic), adaptive behavior, and adaptation mechanism needs of a

DAS. Each of these i* goal models addresses the three RE concerns (conditions to monitor, decision-making procedure, and possible adaptations) from a specific developer's perspective. The original four Levels of RE done for a DAS specified by [14] are as follows:

- Level 1 is the traditional RE work done for a system. Specifically, it deals with the application domain of a DAS and identifies all possible steady-state systems that can be executed by the DAS after adaptation.
- Level 2 is the RE work done by the DAS itself at run time to detect the need to adapt and to select the appropriate target system to adopt.
- Level 3 is the RE work done to select and configure the DAS adaptation infrastructure for a specific DAS application (e.g., what kinds of monitoring options exist to support a given monitoring need?). A given adaptation mechanism may be used for multiple adaptation needs within a given DAS, and there may be many different adaptation mechanisms from which to choose.
- Level 4 is the RE research into adaptation to identify the adaptation infrastructure needs. For example, what type of monitoring support (e.g., software sensors, hardware sensors) is needed? What is the granularity of the monitoring data types? What type of monitoring (e.g., centralized, distributed real-time) needs to be performed?

The LoREM reify the levels in four key ways: First, Level 2 describes the work performed by a developer. In the original Levels of RE, Level 2 referred to the RE work done by a DAS at run time. Feedback from DAS developers indicated that while this is true for idealistic (and futuristic) DAS, it is not realizable with current technology. Second, four types of developers are identified, where each level corresponds to the work of a different developer to construct goal model(s) describing their requirements for a DAS. Third, each level describes the modeling work performed by the specific developer, models constructed, and how to integrate the models constructed at the other levels. The task of each developer is identified by refining the work description in the original levels to describe modeling activities present in the Model Driven Development (MDD) of a DAS. Fourth, to provide context for the LoREM and guidance for integrating the RE work into an overall development process for a DAS, each level is annotated with the MDD phase(s) in which its activities occurs.

4.1.2 REcording of Assumptions in RE

Self adaptation is often used to mitigate an inability to accurately predict the range of environmental contexts that a system will encounter at run-time. The specification and design of SAS is thus often subject to uncertainty, forcing the developer to make assumptions in order to identify and define the means to achieve the system's goals. Using Claims, an assumption made in selecting a goal realization strategy can be made explicit. In REcording of Assumtions in RE (REAssuRE) [94], the i* SR models used in LoREM [37] is extended with Claims. Claims are attached to softgoal contribution links and are used to record the rationale for a choice of goal realization strategy when there is uncertainty about the optimum choice. In REAssuRE, if data collected by monitoring provides evidence that an assumption is false, the effects can be propagated to the goal models used to specify the goal realization

strategy, which can then be automatically re-evaluated. This may trigger the system to adapt by binding to an alternative means of goal realization.

4.1.3 Awareness Requirements

Awareness Requirements (AwReqs) [80] are requirements that talk about the success or failure of other requirements. More generally, AwReqs talk about the states requirements can assume during their execution at run-time. There is much and growing interest in software systems that can adapt to changes in their environment or their requirements in order to continue to fulfill their mandate. Such adaptive systems usually consist of a system proper that delivers a required functionality, along with a Monitor-Analyze-Plan-Execute (MAPE) [48] feedback loop that operationalizes the system's adaptability mechanisms. AwReqs are characterized syntactically as requirements that refer to other requirements or domain assumptions and their success or failure at run-time. AwReqs are represented in a formal language and can be directly monitored by a requirements monitoring framework.

Like other types of requirements, AwReqs are systematically elicited. Since they refer to the success/failure of other requirements, their elicitation takes place after the basic requirements have been elicited and the goal model constructed. In AwReqs, the alternative requirements refinement are captured using *OR* decompositions of goals and unanticipated adaptations are not taken into account.

4.1.4 RELAX augmented with CLAIMS

Claims [94, 93] were applied as markers of uncertainty to record the rationale for a decision made with incomplete information in a DAS. The work in [76] integrates Relax and Claims to assess the validity of Claims at run time while tolerating minor and unanticipated environmental conditions that can otherwise trigger adaptations. Certain properties about the DAS or its execution environment might not be known until run-time. This uncertainty forces developers to make assumptions about the design or configuration of the system. A Claim can be used at design time to document and analyze assumptions about how a DAS achieves its goals in different operational contexts. For example, a Claim can be used to document that data should be encrypted since a network might not be secure. A Claim can also be monitored at run time to prove or disprove its validity [94], thereby triggering an adaptation to reach more desirable system configurations if necessary. Nevertheless, Claims are also subject to uncertainty, in the form of unanticipated environmental conditions and unreliable monitoring information, that can adversely affect the behavior of the DAS if it spuriously falsifies a claim.

In this approach, a stepwise process for Relax-ing Claims is defined. First, sources of uncertainty that can disprove the validity of a Claim is identified. Next, a Claim applicability metric is derived to compute the veracity of a Claim at run time. Both ordinal and temporal Relax operators can be applied to Relax the constraints that define this applicability metric. At run time, if the value produced by a Relax-ed Claim applicability metric drops below

a predetermined threshold, then the value of the corresponding contribution link must be updated and, if necessary, the system may have to reconfigure towards a different goal realization strategy depending on the current set of valid Claims.

4.1.5 AutoRELAX

Relax can be used in goal oriented modeling approaches for specifying and mitigating sources of uncertainty in a DAS [20]. AutoRELAX [77], is an approach that generates Relax-ed goal models that address environmental uncertainty by identifying which goals to Relax, which Relax operators to apply, and the shape of the fuzzy logic function that defines the goal satisfaction criteria.

AutoRELAX explicitly handles environmental uncertainty in a DAS by automatically Relax-ing goals in a Kaos goal model. In particular, AutoRELAX specifies whether a goal should be Relax-ed, and if so, which Relax operator to apply, and to what degree to lessen the constraints or bounds that define a goal's satisfaction criteria. AutoRELAX can be applied to automatically generate one or more Relax-ed goal models, each of which enables a DAS to cope with specific manifestations of system and environmental uncertainty while reducing the number of adaptations performed. AutoRELAX follow the same analogy as that of requirements, a goal can also be classified either as an invariant or a non-invariant. While a system must always satisfy invariant goals, a system may tolerate the temporary violation of a non-invariant goal [77].

AutoRELAX requires a goal model of the Functional Requirements (FRs) that the DAS must satisfy. A requirements engineer must derive utility functions that can monitor the satisfaction of requirements in a DAS [75, 89]. Each utility function comprises mathematical relationships that map monitoring data to a scalar value between zero and one. This value is proportional to how well a given goal or requirement is satisfied at run time. AutoRELAX uses these utility functions to evaluate how goal Relax-ations can affect DAS behavior. AutoRELAX also requires an executable specification of the DAS, such as a simulation or a prototype, that applies the set of utility functions to measure how well the DAS satisfies its requirements in response to adverse conditions.

AutoRELAX provides automated tool support that relieves the requirements engineer from the daunting task of considering a large number of strategies for dealing with uncertainty. For the experimental setup of AutoRELAX, a null hypothesis is defined which states that there is no difference between a Relax-ed and an unRelax-ed goal model. For the time being, AutoRELAX focuses only on Relax-ing the satisfaction of functional non-invariant goals, it could be extended to also automatically Relax the satisfaction criteria of soft goals.

4.1.6 Fuzzy Live Adaptive Goals for Self-Adaptive Systems

Fuzzy Live Adaptive Goals for Self-Adaptive Systems (FLAGS) [11] is an innovative goal model which deal with the challenges posed by SAS. Goal models have been used for representing systems' requirements, and also for tracing them onto their underlying

operationalization. However, goals do not directly address adaptation. To do this, goals must cope with changes and how they can modify themselves and, if needed, the whole goal model.

FLAGS generalizes the basic features of the Kaos model (i.e., refinement and formalisation), and adds the concept of adaptive goal. These goals define the countermeasures that one must perform if one or more goals are not fulfilled satisfactorily. Each countermeasure produces changes in the goal model. Countermeasures may prevent the actual violation of a requirement, enforce a particular goal, or move to a substitute one. The selection at run-time depends on the satisfaction level of related goals and the actual conditions of the system and of the environment. As for goal satisfaction, a crisp notion (yes or no) would not provide the flexibility necessary in systems where some goals cannot be clearly measured (softgoals), properties are not fully known, their complete specification with all possible alternatives would be error-prone, and small/transient violations must be tolerated. This is why, besides crisp goals, fuzzy goals [74] are proposed, specified through fuzzy constraints, that quantify the degree to which a goal is satisfied/violated.

FLAGS is based on the fact that there is not a clear-cut criterion to decide whether soft goals are satisfied or not. This is why FLAGS distinguishes between crisp and fuzzy goals. The fulfillment of the former is boolean, while the latter can be satisfied up to a certain level. Crisp goals are rendered in Linear Temporal logic (LTL), while fuzzy goals use a fuzzy temporal language. The semantics of the fuzzy language is inspired by the theory of fuzzy sets, originally proposed by [102].

4.1.7 Discussion

Few of the existing approaches for RE provide the capability of capturing the uncertainty surrounding SAS. In goal modeling notations such as Kaos [27] and i* [98], there is no explicit support for uncertainty or adaptivity. Scenario based notations generally do not provide any support either although Live sequence charts (LSC) [41] have a notion of mandatory versus potential behavior which could possibly be used in specifying adaptive systems. In AwReqs, the fact that a system may fail in achieving any of its initial requirements is in place. Critical requirements are supplemented by AwReqs that ultimately lead to the introduction of feedback loop functionality into the system to control the degree of violation of critical requirements. Thus, the feedback infrastructure is there to reinforce critical requirements and not to monitor the satisfaction of Relax-able requirements. The introduction of feedback loops in AwReqs is ultimately justified by criticality concerns.

FLAGS proposes adaptive goals as means to conveniently describe adaptation countermeasures in a parametric way, that is with respect to the satisfaction level of the other goals or the environmental conditions. Each countermeasure comes with a set of constraints (trigger and conditions) that identify the execution points where it must be performed, an objective to be achieved, and a sequence of actions applied on the goal model to fulfill the aforementioned objective. FLAGS is based on the use of adaptation goals for introducing adaptivity in SAS. LoREM reifies the different levels of RE for DAS by integrating the concepts of MDD. We also

take benefit of the MDE techniques in our proposed approach for the requirements modeling of SAS.

4.2 Properties Verification of Self Adaptive Systems

Formal methods are intended to systemize and introduce rigor into all the phases of software development. This helps us to avoid overlooking critical issues, provides a standard means to record various assumptions and decisions, and forms a basis for consistency among related activities. By providing precise and unambiguous description mechanisms, formal methods facilitate the understanding required to coalesce the various phases of software development into a successful endeavor [1]. In this section, I give an overview of the state of the art of the existing techniques regarding properties verification.

4.2.1 MEDISTAM-RT

[13] present a verification approach based on MEDISTAM-RT which is a methodological framework for the design and analysis of real-time systems and timed traces semantics, to check the fulfillment of non-functional requirements. It focuses on safety and timeliness properties, to assure the correct functioning of AAL systems and to show the applicability of this methodology in the context of this kind of systems.

The specification and verification of critical systems starts by modeling the system using a semi-formal notation based on UML-RT [84]. The behavior of each one of the components of a critical system architecture has strong requirements that should always be satisfied. The formal specification of this behavior and the requirements these components should fulfill allows to verify that the components work as expected. For the formalizations of NFRs, the properties to be satisfied by a system or a process are defined in terms of timed traces. This definition characterizes some traces as acceptable and some as non-acceptable. A process complies with its specification if all its executions are acceptable, that is, none of its executions by the system violates its specification.

4.2.2 Timed UML and RT-LOTOS Environment

[7] introduce a profile named Timed UML and RT-LOTOS Environment (TURTLE) which extends the UML class and activity diagrams with composition and temporal operators. TURTLE is a real-time UML profile with a formal semantics expressed in Real Time Language Of Temporal Ordering Specifications (RTLOTOS) [26]. With its formal semantics and toolkit, TURTLE enables a priori detection of design errors through a combination of simulation and verification/validation techniques. The "simulation" means a partial exploration of the system state space. It is often used for debugging purposes and to quickly increase confidence in a design. For finite state space systems, exhaustive analysis is also possible. Verification relies on the exploration of the whole system state space in order to prove absence of deadlocks

1. http://www.cs.cmu.edu/~svc/

for instance and other general properties that should be satisfied by any system. Validation also relies on exhaustive analysis to demonstrate that a model meets specific requirements, or exhibits a certain behavior. Here verification and validation are distinguished; the term "verification" is used for checking general properties any system should exhibit, and the term "validation" for system's specific properties such as the validation of a design against the requirements.

4.2.3 UPPAAL

UPPAAL [2] is a tool box for validation (via graphical simulation) and verification (via automatic model-checking) of real-time systems [60]. The name UPPAAL is an acronym for the universities of Uppsala, Sweden and Aalborg, Denmark. It consists of two main parts: a graphical user interface and a model-checker engine. The idea is to model a system using timed automata, simulate it and then verify properties on it. Timed automata are finite state machines with time (clocks). A system consists of a network of processes that are composed of locations. Transitions between these locations define how the system behaves. The simulation step consists of running the system interactively to check that it works as intended. Then the verifier checks the reachability properties, i.e., if a certain state is reachable or not. The main purpose of a model-checker is to verify the model w.r.t. a requirement specification. Like the model, the requirement specification must be expressed in a formally well-defined and machine readable language. Several such logics exist in the scientific literature, and UPPAAL uses a simplified version of Timed Computation Tree Logic (TCTL).

4.2.4 SysML with B Specifications

The main idea behind this work is to extend the SysML with concepts of existing RE methods [53]. An extension to SysML with concepts from the goal model of the Kaos method (SysML/Kaos) is presented with rules to derive a formal B [1] specification from this goal model. The B formal method is a complete method that supports a large segment of the software development life cycle: specification, refinement and implementation. It ensures, thanks to refinement steps and proofs, that the code matches to the specification. The B method is based on Abstract Machine Notation (AMN) and the use of formally proved refinements. Its mathematical basis is extracted from first-order logic, integer arithmetic and set theory. A B specification is structured in machines which contain state variables, invariant properties expressed on the variables and operations specified in the generalised substitution language, which is a generalisation of the Dijkstra's guarded command notations. Each operation have to preserve the invariant of the machine. Kaos on the other hand, is a goal base RE method. Kaos requires the building of a data model in Uml-like notation. The particularity of Kaos is that it is able to implement goal-based reasoning. A goal defines an objective the system should meet, usually through the cooperation of multiple agents such as devices or humans.

2. http://www.it.uu.se/research/group/darts/uppaal

The transition from the requirements phase to the formal specification phase is one of the most difficult steps in software development. A possible solution for bridging this gap is to automatically derive B specifications from KAOS goal models. Unfortunately, this solution is very complicated and hard to set up since it is necessary to construct the body of B operations. To overcome these difficulties, this approach provides a simpler solution that consists in defining a mapping between requirements and B models to improve traceability. The main idea behind this approach is to build the architecture of the specifications from the goal model. It consists in defining what a B machine contains and the links between the different machines.

4.2.5 The OMEGA2 UML/SysML Profile

OMEGA2 [71] is an executable UML/SysML profile dedicated to the formal specification and validation of critical real-time systems. It is based on a subset of UML 2.2/SysML 1.1 containing the main constructs for defining the system structure and behavior.

For specifying and verifying dynamic properties of models, OMEGA2 uses the notion of *observers*. *Observers* are special classes/blocks monitoring run-time state and events. They are defined by classes/blocks stereotyped with «observer». They may have local memory (attributes) and a state machine describes their behavior. States are classified as «success» and «error» states to express the (non)satisfaction of safety properties. The main issue in modeling *observers* is the choice of events which trigger their transitions, and which must include specific UML/SysML event types. The trigger of an *observers* transition is a match clause specifying the type of event (e.g., receive), some related information (e.g., the operation name) and observer variables that may receive related information (e.g., variables receiving the values of operation call parameters). Besides events, an observer may access any part of the state of the UML model: object attributes and state, signal queues.

4.2.6 IFx Toolset

OMEGA2 models can be simulated and properties can be verified using the IFx toolset [17]. The IFx toolset relies on a translation of UML/SysML models towards a simple specification language based on an asynchronous composition of extended timed automata and on the use of simulation and verification tools available for it. The translation takes an input model in XMI 2.0 format. The compiler verifies the set of well-formedness rules imposed by the profile and generates a model that can be further reduced by static analysis techniques. This model is subject to verification that either validates the model with respect to its properties or produces a list of error scenarios that can be further debugged using the simulator.

4.2.7 Discussion

For the properties verification of SAS, we use OMEGA2/IFx profile and toolset. The advantage of OMEGA2 profile is that it provides the notion of *observers* for specifying and verifying dynamic properties of models. OMEGA2 models can be simulated and properties

can be verified using the IFx toolset. The choice of OMEGA2I/Fx is in part due to our familiarity with it as it is developed and maintained in our research team and also it has been applied for the verification and validation of industry grade models [30] providing interesting results.

4.3 Conclusion

This chapter describes the state of the art of the different approaches that we used in this research work. It covers two main themes i.e. RE for SAS and properties verification of these systems. For the RE of SAS, the most promising approaches to date are goal oriented. Most of the work that I cited uses goal models for RE. For the properties verification part, I give a description of some formal methods techniques used for it. We use OMEGA2/IFx profile and toolset for the properties verification and model simulation of AAL system. A detailed description of OMEGA2/IFx is given in chapter 5.

CHAPTER 5

Basic Elements

Contents

In this chapter, I describe the basic concepts that we used in this research work. I introduce Relax which is an RE language for SAS along with its vocabulary, operators, grammar and process. I then give a description of SysML/Kaos, which is a GORE technique. We have integrated SysML/Kaos in our proposed approach (chapter 6) for modeling the requirements of SAS. The chapter concludes with an introduction to OMEGA2/IFx profile and toolset that we used for the properties verification and model simulation of SAS.

5.1 RELAX

Relax is an RE language for DAS, where explicit constructs are included to handle uncertainty. The need for DAS 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 [96].

5.1.1 RELAX Vocabulary

The vocabulary of Relax is designed to enable the analysts to identify the requirements that may be Relax-ed when the environment changes. Relax addresses both types of uncertainties. For example, the system might wish to temporarily Relax a non-critical requirement in order to ensure that critical requirements can still be met. Relax outlines a process for translating traditional requirements into Relax requirements. The only focal point is for the requirement engineers to identify the point of flexibility in their requirements.

5.1.2 RELAX-ed v/s Invariant Requirements

Relax takes the form of a structured natural language, including operators designed specifically to capture uncertainty [95], their semantics is also defined. Typically textual requirements prescribe behavior using a modal verb such as *SHALL* that defines the functionality that a software system must always provide. For SAS 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 non-critical statements in favor of other, more critical ones. Therefore Relax identifies two types of requirements: one that can be Relax-ed in favor of other ones called variant or Relax-ed and other that should never change called invariant. It is important to note that the decision of whether a requirement is invariant or not is an issue for the system stakeholders, aided by the requirements engineer.

5.1.3 RELAX Operators

Figure 5.1 shows the set of RELAX operators, organized into modal, temporal, ordinal operators and uncertainty factors. The conventional modal verb *SHALL* is retained for expressing a requirement with RELAX operators providing more flexibility in how and when that functionality may be delivered. More specifically, for a requirement that contributes to the satisfaction of goals that may be temporarily left unsatisfied, the inclusion of an alternative, temporal or ordinal RELAX-ation modifier will define the requirement as RELAX-able.

The RELAX operators are designed to enable requirements engineers to explicitly identify requirements that should never change (invariants) as well as requirements that a system could temporarily RELAX under certain conditions. RELAX can also be used to specify constraints on how these requirements can be RELAX-ed. Each of the RELAX-ation operators define constraints on how a requirement may be RELAX-ed at run-time. In addition, it is important to indicate what uncertainty factors warrant a RELAX-ation of these requirements, thereby requiring adaptive behavior. This information is specified using the MON (monitor), ENV (environment), REL (relationship) and DEP (dependency) keywords. The environment properties capture the *state of the world* i.e., they are characteristics of the operating context of the system. Often, however, environmental properties cannot be monitored directly because they are not observable. The MON keyword is used to define those properties which are directly observable and which may contribute information towards determining the state of the environment. RELAX is intended to be used at the software requirements phase once hardware constraints have already been defined. In particular, e.g. physical sensors (denoted by MON) are assumed to be known. The REL keyword is used to specify in what way the observables (given by MON) can be used to derive information about the environment (given by ENV). Finally requirements dependencies are delimited by DEP, as it is important to assess the impact on dependent requirements after RELAX-ing a given requirement [96].

5.1.4 RELAX Grammar

The syntax of RELAX expressions is defined by the grammar given below. Parameters of RELAX operators are typed as follows: p is an atomic proposition, e is an event, t is a time interval, f is a frequency and q is a quantity. An event is a notable occurrence that takes place at a particular instant in time. A time interval is any length of time bounded by two time instants. A frequency defines the number of occurrences of an event within a given time interval. If the number of occurrences is unspecified, then it is assumed to be one. A quantity is something measurable, that is, it can be enumerated. In particular, a RELAX expression φ is said to be quantifiable if and only if there exists a function Δ such that $\Delta(\varphi)$ is a quantity. A valid RELAX expression is any conjunction of statements $s_1 \ldots s_m$ where each s_i is generated by the following grammar:

RELAX operator	Description
Modal Operators	
SHALL	a requirement must hold
MAY ... OR	a requirement specifies one or more alternatives
Temporal Operators	
EVENTUALLY	a requirement must hold eventually
UNTIL	a requirement must hold until a future position
BEFORE, AFTER	a requirement must hold before or after a particular event
IN	a requirement must hold during a particular time interval
AS EARLY, LATE AS POSSIBLE	a requirement specifies something that should hold as soon as possible or should be delayed as long as possible
AS CLOSE AS POSSIBLE TO [frequency]	a requirement specifies something that happens repeatedly but the frequency may be relaxed
Ordinal Operators	
AS CLOSE AS POSSIBLE TO [quantity]	a requirement specifies a countable quantity but the exact count may be relaxed
AS MANY, FEW AS POSSIBLE	a requirement specifies a countable quantity but the exact count may be relaxed
Uncertainty Factors	
ENV	defines a set of properties that define the system's environment
MON	defines a set of properties that can be monitored by the system
REL	defines the relationship between the **ENV** and **MON** properties
DEP	identifies the dependencies between the (relaxed and invariant) requirements

Figure 5.1: Relax Operators [96]

$$
\begin{aligned}
\varphi := \mathit{true} \mid \mathit{false} \mid p \mid \mathit{SHALL}\ \varphi \\
\mid \mathit{MAY}\ \varphi_1 \mathit{OR\ MAY}\ \varphi_2 \\
\mid \mathit{EVENTUALLY}\ \varphi \mid \varphi_1 \mathit{UNTIL}\ \varphi_2 \\
\mid \mathit{BEFORE}\ e\ \varphi \mid \mathit{AFTER}\ e\ \varphi \mid \mathit{IN}\ t\ \varphi \\
\mid \mathit{AS\ CLOSE\ AS\ POSSIBLE\ TO}\ f\ \varphi \\
\mid \mathit{AS\ CLOSE\ AS\ POSSIBLE\ TO}\ q\ \varphi \\
\mid \mathit{AS}\ \{\mathit{EARLY}, \mathit{LATE}, \mathit{MANY}, \mathit{FEW}\} \\
\mathit{AS\ POSSIBLE}\ \varphi
\end{aligned}
$$

The semantics of Relax expressions is defined in terms of Fuzzy Branching Temporal Logic (FBTL) [42]. FBTL can describe a branching temporal model with uncertain temporal

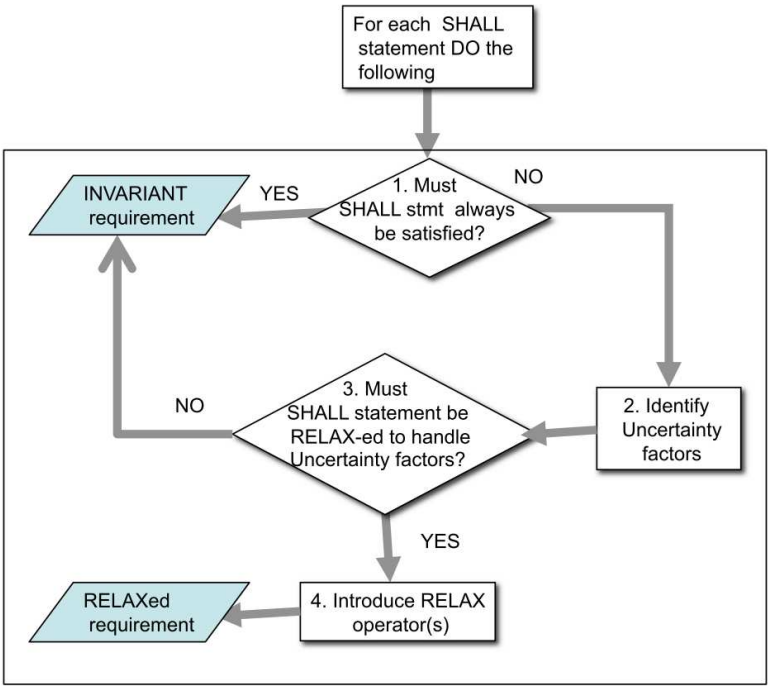

Figure 5.2: Relax Process [96]

and logical information. It is the representation of uncertainty in FBTL that makes it suitable as a formalism for RELAX.

5.1.5 RELAX Process

Figure 5.2 shows the RELAX process. First of all, the conventional process of requirement discovery has been applied to get *SHALL* statements. RELAX process is then used to identify the requirements as invariant and RELAX-ed. For each *SHALL* statement, the following steps are applied:

1. Must *SHALL* statement always be satisfied?: For each *SHALL* statement, determine whether it must always be satisfied (e.g., safety property), or whether it could be RELAX-ed under certain circumstances. In the former case, leave the *SHALL* statement as is, and denote it as an invariant requirement. A non-invariant requirement is potentially RELAX-able, thus implying that some form of run-time adaptation may be necessary to make the best use of the available resources while delivering acceptable behavior.
2. Identify uncertainty factors for each potentially RELAX-able requirement: Here the objective is to help the requirement engineer ascertain whether uncertainty exists in the ENV, thus potentially making satisfaction of the requirement problematic and necessitate its RELAX-ation. Here also the observable properties of the environment are identified. The ENV/MON relationship is made explicit by REL and DEP is used

to identify the inter-dependencies between requirements as these are important to understand when assessing the uncertainty surrounding a requirement.

3. Must *SHALL* statement be Relax-ed to handle uncertainty factors?: Analyze the uncertainty factors to determine if sufficient uncertainty exists in the environment that makes absolute satisfaction of the requirement problematic or undesirable. If so, then this *SHALL* statement needs to proceed to the next step for introducing Relax operators. If, however, the analysis reveals no uncertainty in its scope of the environment, then the requirement is potentially always satisfiable and therefore identified as an invariant.
4. Introduce Relax operator(s): Given the sources of uncertainty, determine whether a requirement should be Relax-ed to introduce ordinal, temporal, or modal behavior flexibility at run time. Sources for uncertainty include: contention for resources, adverse environmental conditions, timing of events, and the duration of conditions.

Note that the process describes a way of incrementally building up a model of the environment. This approach is in contrast to including an explicit task to model the environment. The latter is difficult in practice because it may not be clear which environmental factors might be relevant. It is also important to note that each iteration of the Relax-ation process implicitly includes a form of regression assessment to ensure that the dependencies between the requirements are considered. After the application of Relax process on traditional requirements, we obtain invariant and Relax-ed requirements.

5.1.6 Discussion

In Relax, some requirements are treated as invariants that must always be achieved and Non critical requirements - those that can be violated from time to time - are Relax-ed. Relax then provide the machinery to conclude at run-time that while the system may have failed to fully achieve its Relax-ed requirements, this is acceptable. So, while Relax-ed requirements are monitored at run-time, invariant ones are analyzed at design time and must be guaranteed to be always achievable at run-time.

The advantage of Relax for the RE of SAS is that it gives a means to establish the boundaries of adaptive behavior. That is, one must explicitly distinguish invariant from non-invariant requirements, identify and monitor the sources of uncertainty, and then describe what dimensions of the requirements can be Relax-ed and satisfied by adaptive behavior. The invariants provide a point of reference for adaptive behavior. Second, the Relax process consider each non-invariant requirement in isolation, with the effect of incrementally revealing each requirements interdependencies and generating what is effectively trace information in the DEP attribute [92]. Third, by separately describing the environment and the monitoring, one can identify deficiencies in the monitoring infrastructure. Given that a DAS can only adapt based on its monitoring information, e.g. missing or insufficient sensors for the environment in question significantly impact the effectiveness of the DAS.

Relax-ed requirement supports a high degree of flexibility that goes well beyond the original requirements. Once the requirements engineer determines that indeed a level of

flexibility can be tolerated, then the downstream developers, including the designers and programmers have the flexibility to incorporate the most suitable adaptive mechanisms to support the desired functionality. These decisions may be made at design time and/or run time [15, 21]. To uncover missing monitoring requirements for each RELAX-ed requirement, the requirements engineer asks what information is necessary to invoke the adaptation corresponding to the requirement. If the current monitoring infrastructure does not provide enough information about the environment, then either the monitoring infrastructure should be extended or, if resources do not permit it, the adaptive capability must be reduced. This kind of trade-off analysis can be effectively modeled using goal-based requirements languages [96], in particular, obstacle analysis in KAOS [55]. In the next section, I introduce SYSML/KAOS, which extends SYSML with KAOS concepts.

5.2 SysML/KAOS

The SYSML/KAOS [36] model is an extension of the SYSML [1] requirements model with concepts of the KAOS goal model [55]. As SYSML is an extension of UML [2], it provides concepts to represent requirements and to relate them to other model elements, allowing the definition of traceability links between requirements and system models. The SYSML/KAOS meta-model is implemented as a new profile, importing the SYSML profile.

5.2.1 SysML

SYSML is a general purpose modeling language for systems engineering applications. SYSML is a UML profile that represents a subset of UML 2.0 with extensions. Figure 5.3 shows the relationship between SYSML and UML. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametrics, which is used to integrate with other engineering analysis models. The SYSML diagram types are shown in Figure 5.4.

The *«block»* is the basic unit of structure in SYSML and can be used to represent hardware, software, facilities, personnel, or any other system element. The system structure is represented by Block Definition Diagram (BDD) and Internal Block Diagram (IBD).

- The BDD describes the system hierarchy and system/component classifications.
- The IBD describes the internal structure of a system in terms of its parts, ports, and connectors.
- The package diagram is used to organize the model.

The behavior diagrams include the use case diagram, activity diagram, sequence diagram, and State Machine Diagram (SMD).

1. http://www.omgsysml.org/
2. http://www.omg.org/spec/UML/

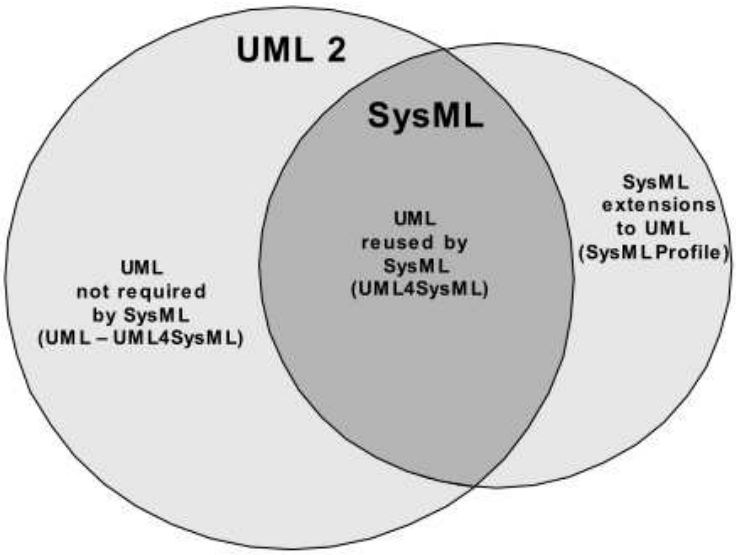

Figure 5.3: SysML UML Relationship

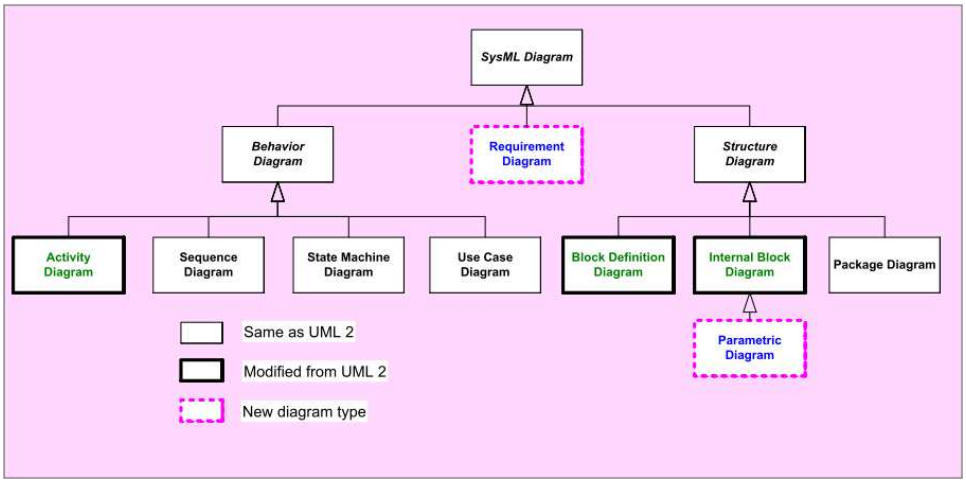

Figure 5.4: SysML Diagrams

- The use-case diagram provides a high-level description of functionality that is achieved through interaction among systems or system parts.
- The activity diagram represents the flow of data and control between activities.
- The sequence diagram represents the interaction between collaborating parts of a system.
- The SMD describes the state transitions and actions that a system or its parts perform in response to events.
- The parametric diagram represents constraints on system property values such as performance, reliability, and mass properties, and serves as a means to integrate the specification and design models with engineering analysis models.

SysML also includes an allocation relationship to represent various types of allocation, including allocation of functions to components, logical to physical components, and software to hardware. SysML includes a graphical construct to represent text based requirements and relate them to other model elements. The requirements diagram captures requirements hierarchies and requirements derivation, and the «*satisfy*» and «*verify*» relationships allow a modeler to relate a requirement to a model element, e.g. «*block*», that satisfies or verifies the requirements. The requirement diagram provides a bridge between typical requirements management tools and system models.

5.2.2 KAOS

Kaos is a methodology for RE enabling analysts to build requirements models and to derive requirements documents from Kaos models. The first key idea behind Kaos is to build a model for the requirements, that is, for describing the problem to be solved and the constraints that must be fulfilled by any solution provider. Kaos has been designed:

- to fit problem descriptions by allowing to define and manipulate concepts relevant to problem description
- to improve the problem analysis process by providing a systematic approach for discovering and structuring requirements
- to clarify the responsibilities of all the project stakeholders
- to let the stakeholders communicate easily and efficiently about the requirements

5.2.3 Why SysML/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 [3] which is proprietary.

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

The benefit of SysML is that it allows throughout the development cycle to relate requirements to other model elements, thus ensuring continuity from the requirements phase to the

3. http://www.objectiver.com/

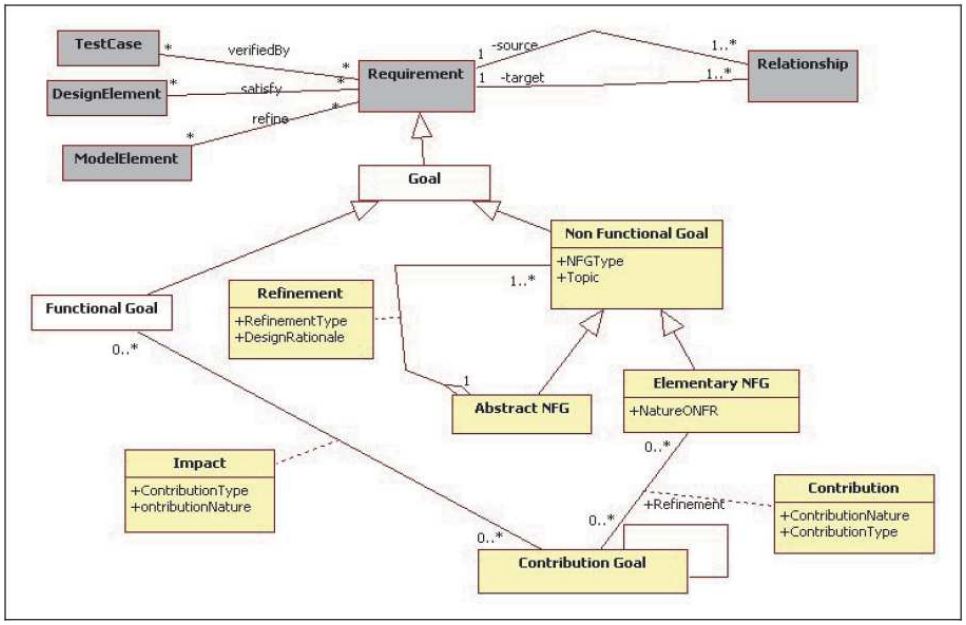

Figure 5.5: SysML/Kaos Meta Model [36]

implementation phase. However, the proposed concepts of requirements in SysML are not as rich as in the other RE methods (especially GORE). SysML/Kaos is the result of motivation to benefit from the contributions of SysML, while ensuring a more precise definition of the concepts. The SysML/Kaos model allows both FRs [53] and NFRs [36] to be modeled.

5.2.4 SysML/KAOS Meta Model

Figure 5.5 shows the extended meta-model of SysML/Kaos, non functional concepts are represented as yellow boxes, the gray boxes represent the SysML concepts. The instantiation of the meta-model allows us to obtain a hierarchy of NFRs in the form of goals. NFGs are organized in refinement hierarchies. The meta-class *Non Functional Goal* represents the Non Functional Goal (NFG), 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. An NFG represents a quality that the future system must have in order to satisfy a FR. The *NFGTYPE* specify the type of NFG and the attribute *TOPIC* represent the domain concept concerned by this type of requirement. An NFG can thus be represented with the following syntax: *NFGType [Topic]*. An NFG is either an *Abstract NFG* or an *Elementary NFG*. A goal that cannot be further refined is an *Elementary Goal*. The refinement of an *Abstract Goal* by either abstract or elementary goals is represented by the *Association Class Refinement*. An *Abstract NFG* may contain several combinations of sub goals (abstract or elementary). The relationship *Refinement* becomes an *Association Class* between an *Abstract NFG* and its sub goals. It can be specialized to represent And/Or goal refinements. At the end of the refinement process, it is necessary to identify and express the various alternative ways to satisfy the *Elementary Goals*. For that, the

SysML/Kaos meta-model consider the concept of contribution goal meta-class *Contribution Goal*. A *Contribution Goal* captures a possible way to satisfy an *Elementary Goal*. The *Association Class Contribution* describes the characteristics of the contribution. It provides two properties: *ContributionNature* and *ContributionType*. The first one specifies whether the contribution is *positive* or *negative*, whereas the second one specifies whether the contribution is *direct* or *indirect*. A *positive* (or *negative*) contribution helps positively (or negatively) to the satisfaction of an *Elementary Goal*. A *direct contribution* describes an explicit contribution to the *Elementary NFG*. An *indirect contribution* describes a kind of contribution that is a direct contribution to a given goal but induces an unexpected contribution to another goal. Finally, the concept of *Impact* is used to connect NFGs to Functional Goals (FGs). It captures the fact that a *Contribution Goal* has an effect on FGs.

5.2.5 Discussion

SysML/Kaos extends the requirements model of SysML with Kaos goal model concepts particularly the NFGs. To support this extension, a tool SysML/Kaos editor is developed. In order to provide traceability links between the requirements analysis and specification phases, the idea behind SysML/Kaos is to include SysML because it covers all the phases of system development. The follow up of SysML/Kaos is to take into account the formal specification of the system. This need arises from the gap between textual or semi formal requirements and the initial formal specification. The validation of this initial formal specification is very difficult due to the inability for customers to understand formal models. Moreover it is hard for designers to link them with the initial requirements. Consequently, the gap between the requirements phase and the formal specification phase becomes larger and larger and the reconciliation seems more and more difficult. To bridge this gap a method is defined using an RE approach and the B formal method [53]. Using this method, derivation rules from SysML/Kaos to a formal B specification are defined.

We aim to provide a mechanism to bridge the gap between the requirements phase and the initial formal specification phase by using MDE techniques and more specifically using model checking techniques in our proposed approach. The constraint of using MDE techniques in this thesis for the properties verification of SAS is due to the fact that the main theme of our research team is MDE and also the tool that we use is developed and maintained by our team members [4]. In the next section, I give a description of OMEGA2/IFx profile and toolset that we use for the properties verification and model simulation of AAL system.

5.3 The OMEGA2 UML/SysML Profile and IFx Toolset

The specification and verification of NFRs in the early stages of the AAL (chapter 7) development cycle is a crucial issue [68]. These systems require clear and precise specifications in order to describe the system behavior and its environment. The formal specification of the

4. http://www.irit.fr/ifx/index.html

system behavior supported by mathematical analysis and reasoning techniques improve their development process and enable the verification of these systems.

5.3.1 The OMEGA2 Profile

Formal methods provide tools to verify the consistency and correctness of a specification with respect to the desired properties of the system. For this reason, We use these methods to prove some of the properties of the system before the system development even starts. OMEGA2 profile is an executable Uml/SysML profile used for the formal specification and validation of critical real-time systems. It is based on a subset of Uml 2.2/SysML 1.1 containing the main constructs for defining the system structure and behavior. The OMEGA2 Profile is supported by the IFx toolset [73] which provides mechanisms for the model simulation and properties verification of the AAL system. The OMEGA2/IFx approach has been applied for the verification and validation of industry grade models [30] providing interesting results.

The OMEGA2 Uml/SysML profile [71] defines the semantics of Uml/SysML elements providing the means to model coherent and unambiguous system models. In order to make the models verifiable, it presents as extension the *observers* mechanism for specifying dynamic properties of models. The OMEGA2 Uml/SysML Profile is implemented by the IFx toolbox which provides static analysis, simulation and timed automaton based model checking [23] techniques for validation.

5.3.1.1 System Structure

The architecture of an OMEGA2 model is described in Class/Block Definition Diagrams by classes/blocks with their relationships. Each class/block defines properties and operations, as well as a state machine. The hierarchical structure of a model is defined in composite structures/Internal Block Diagrams: parts that communicate through ports and connectors. The Uml/SysML Profile leaves open several semantic variation points for which OMEGA2 defines a set of well formedness rules that result in a strong typing language [72].

- Class diagrams (Uml): Classes and their relationships, interfaces, basic types, signals, composite structures (ports, parts, connectors).
- Block diagrams (SysML): Block Definition Diagram (BDD): blocks and their relationships (Association, Aggregation, Generalization), interfaces, basic types, Signals.

5.3.1.2 Class/Block behavior

- State machines (excluding: history states, entry point, exit point, junction).
- Actions: the profile defines a concrete syntax for Uml 2.2 actions. This syntax is used for example to define operation bodies and transition effects in state machines. The textual action language is compatible with the Uml 2.2 action meta model and implements its main elements: object creation and destruction, operation calls, expression evaluation,

variable assignment, signal output, return action as well as control flow structuring statements.

5.3.1.3 Operational and Timed Semantics of OMEGA2

The operational semantics of OMEGA2 relies on an asynchronous timed execution model. Each class/block is represented by a timed input-output automata, potentially executing in parallel with other blocks and communicating via asynchronous operation calls and signals. The OMEGA2 profile can model timed behavior, where the model time base can either be discrete or continuous and it is specified by the user at verification. The time model is controlled by primitives from automata with urgency [16]: clocks, time guards and transition urgency annotations. The *clock* is represented by a *Timer* block on which we can perform actions as: *set* for setting the clock a delay and *reset* to restore the clock to 0. Time guards are either described as inequalities or specified via the *timeout* operation that verifies that a certain delay has elapsed. With respect to time progress, transitions can also define a particular semantics based on their stereotype: *eager* defines that time progress is disabled in a state (i.e., the actions on a transition are executed as soon as possible), *delayable* means that the time progress is enabled but it is bounded by a limit and *lazy* specifies that time progress is enabled and unbounded (i.e. time can progress to infinity). Based on these notions, one can also model synchronous communication in OMEGA2 model.

5.3.1.4 Observers

For specifying and verifying dynamic properties of models, OMEGA2 uses the notion of *observers*. *Observers* are special classes/blocks monitoring run-time state and events. They are defined by classes/blocks stereotyped with «*observer*». They may have local memory (attributes) and a state machine describes their behavior. States are classified as «*success*» and «*error*» states to express the (non)satisfaction of safety properties. The main issue in modeling *observers* is the choice of events which trigger their transitions, and which must include specific UML/SysML event types. One can observe:

- Events related to signal exchange: send, receivesignal, acceptsignal.
- Events related to operation calls: invoke, receive (reception of call), accept (start of actual processing of call – may be different from receive), invokereturn (sending of a return value), receivereturn (reception of the return value), acceptreturn (actual consumption of the return value).
- Informal events explicitly specified by the modeler using the informal action.

The trigger of an *observers* transition is a match clause specifying the type of event (e.g., receive), some related information (e.g., the operation name) and observer variables that may receive related information (e.g., variables receiving the values of operation call parameters). Besides events, an observer may access any part of the state of the UML model: object attributes and state, signal queues.

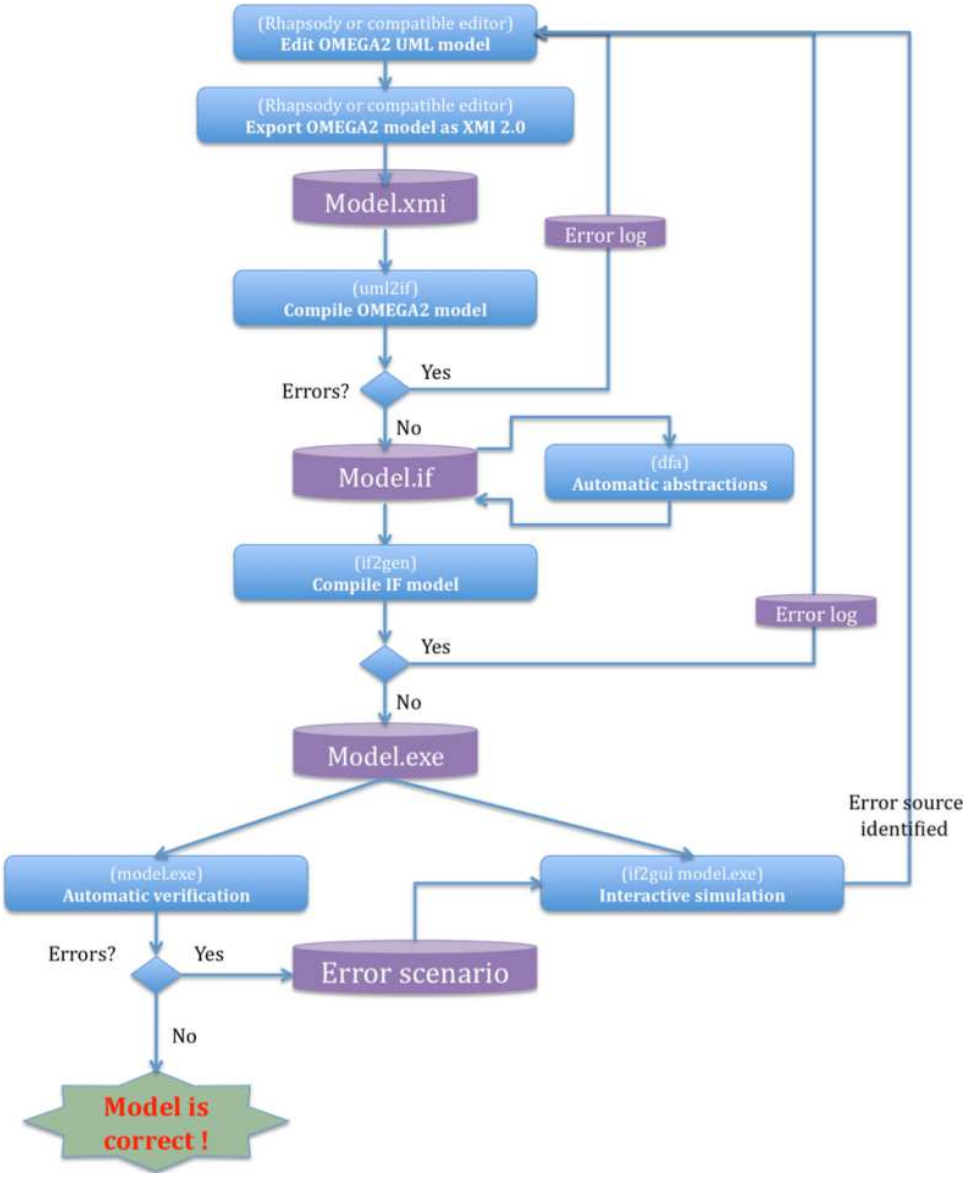

Figure 5.6: IFX Workflow [91]

5.3.2 IFx Toolset

OMEGA2 models can be simulated and properties can be verified using the IFx toolset [17]. The IFx toolset provides the following functions:

- Verification: It designates the automatic process of verifying whether an OMEGA2 UML/SysML model satisfies (some of) the properties (i.e. *observers*) defined on it. The verification method employed in IFx is based on systematic exploration of the system state space (i.e., enumerative model checking).
- Simulation: It designates the interactive execution of an OMEGA2 UML/SysML model.

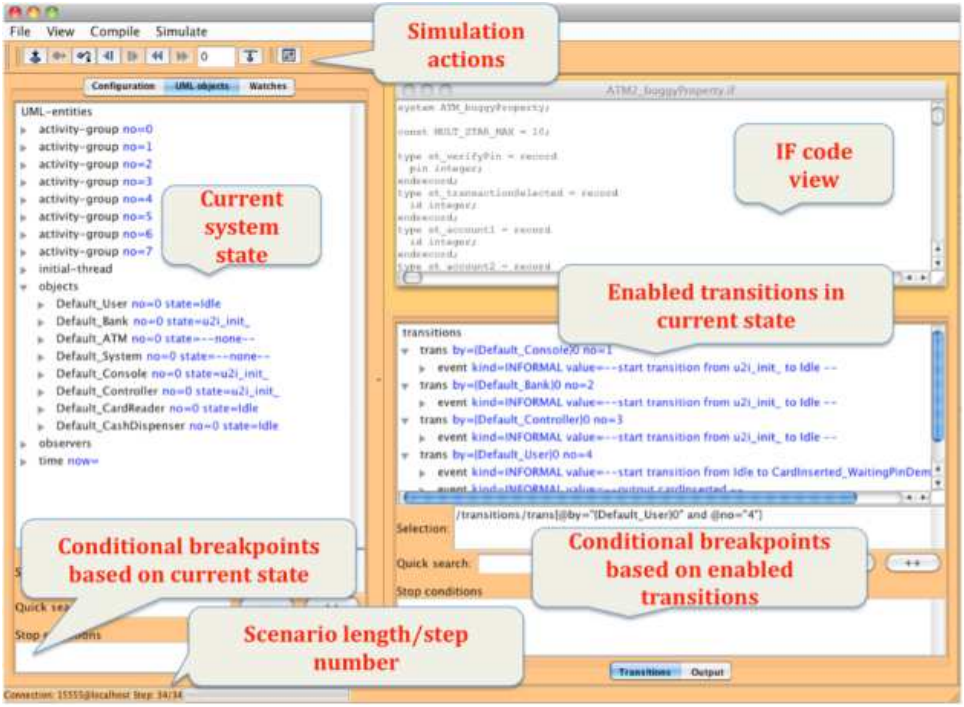

Figure 5.7: If2gui Interface [91]

The execution can be performed step-by-step, random, or guided by a simulation scenario (for example an error scenario generated during a verification activity).

The IFx toolset relies on a translation of UML/SysML models towards a simple specification language based on an asynchronous composition of extended timed automata, the *IF* language [5], and on the use of simulation and verification tools available for *IF*. The translation takes an input model in XMI 2.0 format. The compiler verifies the set of well-formedness rules imposed by the profile and generates an *IF* model that can be further reduced by static analysis techniques. This model is subject to verification that either validates the model with respect to its properties or produces a list of error scenarios that can be further debugged using the simulator. The overall workflow of IFx Toolset [91] is shown in Figure 5.6.

Following are the steps for the compilation and execution of *IF* statements.

1. Export OMEGA2 model as XMI 2.0: In this step, we export the OMEGA2 model into XMI format.
2. Compiling the OMEGA2 model to *if* file by using *uml2if*: In this step we compile the OMEGA2 model in XMI format to an *if* file. The output of *uml2if* is either a list of error messages or, if there is no error, an *if* file containing the translation.
3. Compiling the *if* model to an executable file using *if2gen*: The output of *if2gen* is either a list of error messages or, if there is no error, an executable file containing the *if* model

5. http://www-if.imag.fr/

and the simulation/verification functions. The executable generated by *if2gen* is then used to perform both automatic verification and interactive simulation.

4. Automatic verification (model.exe): At this step, model checking is performed. Here the properties that are defined on the model are verified, if there is no error at this step, it means that the properties are satisfied and that all the states and variables are checked. If there are errors at this step, we can simulate it through the interactive simulation function of the IFx toolset. The errors found in this step correspond to an error in the specification of the system and/or the property.
5. Interactive Simulation using *if2gui*: *if2gui* is a graphical user interface that can be launched from command line.

Figure 5.7 shows the Simulation interface.

5.4 Conclusion

This chapter shows the concepts that serves us as a basis for this research work. I introduce Relax which is an RE language for SAS. I give a description of the Relax vocabulary, operators, grammar and process. Relax base its assumptions on the distinction between two types of requirements i.e. invariant and Relax-ed. This distinction motivated us to use Relax and then to combine it with SysML/Kaos approach which extends the SysML meta-model with Kaos goal concepts. SysML/Kaos has its own meta-model and an editor is developed with the help of which one can model the requirements of SAS. For the properties verification part, I give a detail description of OMEGA2/IFx profile and toolset that we used for the properties verification and model simulation of AAL system (chapter 7).

Part II

CONTRIBUTION

CHAPTER 6

Proposed Approach

Contents

In this chapter, I introduce the overall view of our proposed approach keeping in mind the context that is already introduced in chapter 5 i.e. Relax and SysML/Kaos. I highlight some problems that we identified while modeling and verifying the requirements of SAS. I then show how these problems are treated in our proposed approach. To validate our proposed approach, we need an integrated tooling environment, to show what processes or tools are needed, consequently showing the steps to follow in order to achieve the aforementioned objectives described in chapter 1. We have developed these tools and processes during the course of this thesis. The proposed approach is applied on two different case studies that I introduce in chapter 7.

6.1 Overall View of our Proposed Approach

In the previous chapter, I have described the basic elements of our research study. I now instantiate the problems identified in the problem statement (section 1.1 of chapter 1) with existing approaches of requirements modeling and properties verification. SysML/Kaos is used for the requirements modeling and OMEGA2/IFx is used for the properties verification of SAS. We have addressed these problems in our proposed approach as shown below.

6.1.1 Problem 1

In the conventional process of modeling NFRs, the requirement engineers elicit requirements from the system and then divide it into FRs and NFRs. These NFRs are then mapped to NFGs. NFGs are then modeled using a GORE approach. For SAS, this process is not suitable as for these systems, an important aspect is to differentiate between those requirements that are adaptable and those that are invariant. Most of the existing GORE approaches does not take into account the adaptability features associated with SAS. The requirements that are adaptable i.e. Relax-ed, should be considered as early as possible while modeling the requirement of SAS.

We use SysML/Kaos for modeling the NFRs of SAS. After eliciting requirements from the system and then mapping it to NFGs using the SysML/Kaos meta-model, these NFGs are then modeled using SysML/Kaos. To take into account those requirements that are important for SAS, we must provide a mechanism as early as possible. In Kaos [27], there is no explicit support for uncertainty or adaptivity [21], so the SysML/Kaos approach does not take into account the adaptability features associated with SAS. The requirements that are adaptable i.e. Relax-ed, should be considered as early as possible while modeling the requirement of SAS. Figure 6.1 shows the conventional process of NFRs modeling using SysML/Kaos.

6.1.2 Problem 2

The second problem that we identified concerns the existing approaches of properties verification where properties are injected into a properties verification framework without mentioning which properties are to be given priority. For SAS it is very important to

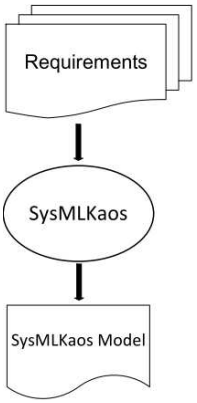

Figure 6.1: Conventional Requirements Modeling using SysML/Kaos

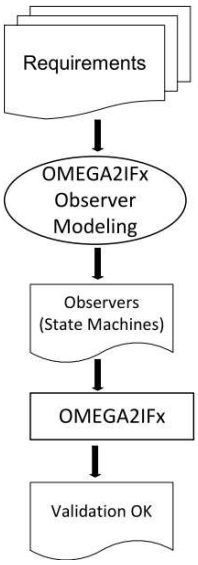

Figure 6.2: Conventional Properties Verification using OMEGA2/IFx

differentiate those requirements that are adaptable from those that are invariant. Most of the properties verification frameworks use model checking techniques for the properties verification and model simulation, so when the number of state variables in the system increases, the size of the system state space grows exponentially and we are exposed to the problems linked with state space explosion [24]. The exponential growth of the state space is normal for systems like SAS.

We use OMEGA2/IFx profile and toolset for the properties verification of SAS. In OMEGA2/IFx, we inject the FRs and NFRs, without differentiating those requirements that are adaptable from those that are invariant, which in the case of SAS is very important. As OMEGA2/IFx use model checking techniques for the properties verification and model simulation, so we are exposed to the problems linked with state space explosion i.e. the more we have state variables, there will be an exponential growth in the state space. We provide a

way to tackle this problem in our proposed approach. Figure 6.2 shows the conventional way of properties verification using OMEGA2/IFx profile and toolset.

6.1.3 Problem 3

Relax is a structured natural language which include operators to define the requirements of self adaptive systems. It defines uncertainty factors, a process to derive Relax-ed and invariant requirements but it does not provide any tool. We had to cope with this problem. We had to take into account this problem in our proposed approach as Relax serves as the basis of our work.

6.2 The Proposed Approach

We provide an integrated approach comprising of tools, processes and documents for the requirements modeling and properties verification of SAS. In the following, each step of the proposed approach is explained with associated input and output. Figure 6.3 shows the overall view of our proposed approach.

1. The overall approach that we proposed, takes requirements as input. These requirements are elicited in the form of *SHALL* statements by a requirement engineer. These requirements are then divided into FRs and NFRs.
2. For SAS, an important aspect is to find the adaptation point in requirements as early as possible. In order to achieve this objective, we apply Relax process (section 5.1.5) on these FRs and NFRs to get those requirements that are associated with the adaptability features of SAS called Relax-ed requirements and those that are fixed called invariant requirements.
3. The resulting Relax-ed requirements are then automated using an editor that we developed called Relax COOL editor. This editor takes into account the uncertainty factors associated with each Relax-ed requirement. Xtext [1] is used for the development of this editor which provides many functions including editor generation, text highlighting, text completion and code generation etc.
4. At this point, we use a process for the conversion of Relax-ed requirements into goal concepts i.e. SysML/Kaos. We use a correlation table (section 6.4.1) for the correspondence between Relax-ed requirements and SysML/Kaos concepts [5].
5. At this step, we have a full list of Relax-ed requirements with uncertainty factors converted into SysML/Kaos goal concepts.
6. The mapping between Relax and SysML/Kaos concepts results in the transformation of Relax uncertainty factors into SysML/Kaos goal concepts. For this purpose, we have developed a tool called Relax2SysML/Kaos editor, which is based on Atlas

1. http://www.eclipse.org/Xtext/

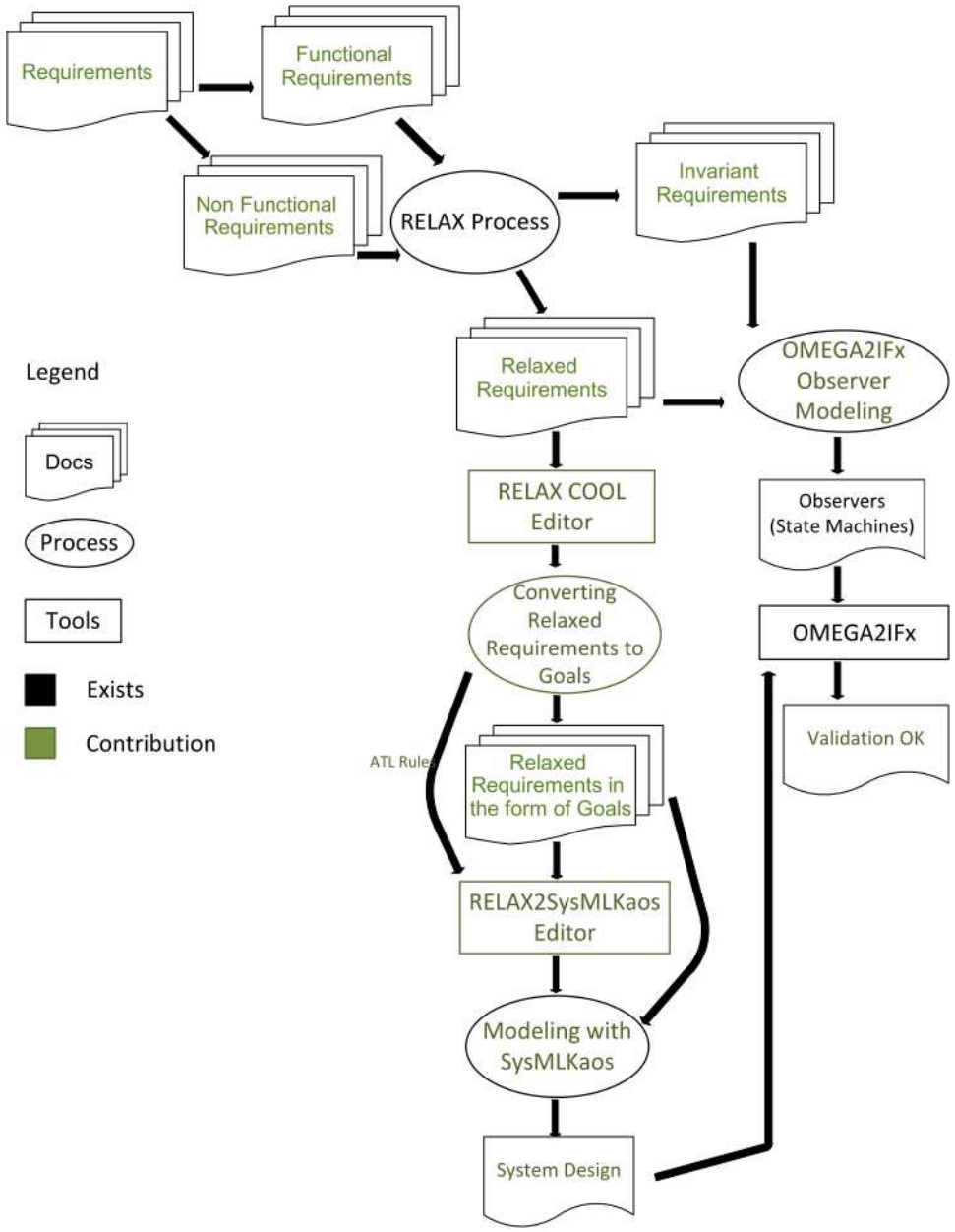

Figure 6.3: Overall View of Our Approach

Transformation Language (ATL) transformations. For the time being, the tool helps in mapping the RELAX concepts to SysML/Kaos concepts but not the inverse.

7. The non functional RELAX-ed requirements in the form of SysML/Kaos goal concepts can now be modeled with the help of SysML/Kaos editor.
8. This step shows the system design. The RELAX-ed requirements of the SAS are now modeled and we have a snapshot of the system design.
9. The resulting system design and the OMEGA2/IFx observers are used to verify the

properties of SAS. For specifying and verifying dynamic properties of models, OMEGA2 uses the notion of observers. Observers are special classes/blocks monitoring run-time states and events. They are defined by classes/blocks stereotyped with «observer». They may have local memory (attributes) and a state machine describes their behavior. The input to this step are the OMEGA2/IFx observers which are the Relax-ed and invariant requirements that we identified in an earlier step. The verification either results in the fulfillment of all the properties or if there is an error produced during verification, it can be simulated through the interactive simulation interface of the IFx toolset in order to identify the source of the error and then subsequently correct it in the model.

In chapter 7, I introduce two case studies i.e. AAL and bCMS, that we used for the validation of our proposed approach. The requirements of both case studies are modeled using Relax and SysML/Kaos and properties of AAL case study are verified using OMEGA2/IFx. The solutions that we provide in our proposed approach for the identified problems are given below. The provided solutions takes the form of an integrated approach.

6.3 The Solutions

6.3.1 Solution 1

To solve the problem of which properties to be modeled, we propose the introduction of the Relax process in the overall approach. The application of Relax process on FRs and NFRs results in adaptable i.e. Relax-able and invariant requirements. Relax-able requirements can then be mapped to SysML/Kaos NFGs and can be modeled. This will help in taking into account the uncertainty factors associated with each Relax-ed requirement of the SAS. The resulting SysML/Kaos model will now best represent the uncertainty associated with these systems.

6.3.2 Solution 2

The problem that we identified with the existing approaches of properties verification is that they do not take into account Relax-ed and invariant requirements for the properties verification but a bunch of FRs and NFRs. In our approach, we consider those requirements for verification that better suits the needs of SAS i.e. Relax-ed requirements. Invariant requirements can also be verified using our proposed approach. The verification method employed in IFx is based on systematic exploration of the system state space (enumerative model checking). Properties Verification techniques explore all possible system states in a brute-force manner. A model checker, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. In this way, it can be shown that a given system model truly satisfies a certain property or not [10]. The state space of a system may be very large for any reasons (the system is complex by nature, contains a lot of dependent parallel activities etc), this is why we base our proposition to verify only those

	Approach/Tool	Input	Output	Contribution
1	RELAX Process	Functional & Non Functional Requirements	RELAX-ed & Invariant Requirements	Divide Requirements into RELAX-ed and Invariant Requirements **(MA)**
2	RELAX COOL Editor **(MA)**	RELAX-ed Requirements	Automated RELAX-ed Requirements	To automate the RELAX-ed Requirements using our RELAX COOL Editor **(MA)**
3	Transformation Towards Goal Concepts **(MA)**	RELAX-ed Requirements	SysML/Kaos Goal Model Concepts	Using the correlation b/w RELAX-ed Requirements and SysML/Kaos Goal concepts **(MA)**
4	RELAX2SysMLKaos Editor **(MA)**	Goal Model Concepts	System Design	RELAX2SysMLKaos Editor **(MA)**
5	Modeling Observers **(MA)**	RELAX-ed & Invariant Requirements	Observers	To model the observers **(MA)**
6	OMEGA2IFx	System Design & Observers	Verified Requirements	Integration of OMEGA2/IFx in the overall approach **(MA)**

Figure 6.4: Overall View of Our Approach Showing Input Output and Contributions

properties of the SAS that are important e.g. RELAX-ed or invariant requirements to avoid the state explosion problem associated with model checking techniques.

6.3.3 RELAX Improvements

Our first experience with RELAX started with the automation of its uncertainty factors i.e. ENV, MON, REL and DEP. The RELAX vocabulary, grammar and process were already defined but no tool was developed. We have developed RELAX COOL editor. The editor is capable of taking into account the uncertainty factors associated with RELAX-ed requirements of SAS. All the operators of RELAX are also implemented while RELAX-ing a requirement. We then use RELAX in our proposed approach for modeling the requirements of SAS using the correlation between RELAX and SysML/Kaos.

6.4 Integration of the Approaches

We start from the very early analysis of requirements and we provide a high level design of the system. The design along with *observers* is then verified using OMEGA2/IFx to look for errors in the system. We provide an integrated model based environment for early analysis and verification of requirements. The problems that we identified in the previous section are solved using our integrated approach. The tools, processes and documents needed to achieve

the integrated environment are developed during the course of this research work. Figure 6.4 shows the overall view of our integrated approach in a tabular form with the associated input and output at each step. The table also shows the contribution at each step, with **(MA)** showing our contribution. In the following section, I explain the different steps that forms the basis of our integrated approach.

6.4.1 Relatonship b/w RELAX, SysML/KAOS and SysML

In our integrated approach, we take benefit of SysML/Kaos while modeling Relax-ed requirements of SAS. In Figure 6.5, we show how several key concepts are taken into account in the selected approaches. Most of the time, the concepts are not fully covered (e.g. `«satisfy»` for monitoring in SysML, this stereotype is used between a block and a requirement), but we have indicated in the Figure 6.5 the closest mechanism that supports the concepts. The concepts are taken from Relax and are then compared with the three different approaches.

- In SysML/Kaos, requirements are described in the form of goals *Abstract Goals* and *Elementary Goals*; *Abstract Goal* is a goal that needs further refinement, a goal that cannot be further refined is an *Elementary Goal*. SysML describes requirements in textual form, it provides requirements diagram for modeling the requirements and the relationship between requirements and other SysML model elements. 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. The uncertainty factors helps in capturing the uncertainty present in SAS. We have developed Relax COOL editor for automating the Relax-ed requirements along with the uncertainty factors associated with each Relax-ed requirement. For the correspondence between Relax and SysML/Kaos,Relax-ed requirement serves as an *Abstract Goal* in SysML/Kaos and the ENV uncertainty factor of Relax serves as an *Elementary Goal* of SysML/Kaos.
- In Relax monitoring is used to define a set of properties that can be monitored by the system. To deal with monitoring, SysML/Kaos has the *Contribution Goal* concept which is used to satisfy an *Elementary NFG*, SysML has *«satisfy»* which is 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. The MON uncertainty factor of Relax serves as a *Contribution Goal* in SysML/Kaos.
- SysML/Kaos has the concept of *Contribution* which is an *Association Class* between *Contribution Goal* and *Elementary NFG*. *Contribution* describes the characteristics of the contribution. It provides two properties: *ContributionNature* and *ContributionType*. The first one specifies whether the contribution is *positive* or *negative*, whereas the second one specifies whether the contribution is *direct* or *indirect*. A *positive* (or *negative*) contribution helps positively (or negatively) to the satisfaction of an *Elementary NFG*. A *direct contribution* describes an explicit contribution to the *Elementary NFG*. An *indirect contribution* describes a kind of contribution that is a direct contribution to a given goal but induces an unexpected contribution to another goal. SysML has *«verify»* and *«refine»* relationships while for Relax, we have REL variable which identifies the relationship

between ENV and MON or more precisely how MON achieves ENV. For the correlation, the REL uncertainty factor of RELAX becomes *Contribution* in SYSML/KAOS.

– For Dependency/Impact, SYSML/KAOS describes it as an *Impact* of an NFG on an FG. In SYSML/KAOS meta-model (Figure 5.5), the *Impact* is an *Association Class* between *Contribution Goal* and *FG*. It captures the fact that a *Contribution Goal* has an impact on an FG. The *Impact* describes the characteristics of this impact. It also has the same two properties i.e. *ContributionNature* and *ContributionType*. This impact can be *positive* or *negative* and *direct* or *indirect*. In SYSML, we have the concept of «*derive*» which shows the dependency between requirements, RELAX has *positive* and *negative* dependency which shows the dependency of a RELAX-ed requirement on other requirements as it is important to assess the impact on dependent requirements after RELAX-ing a given requirement. For the correlation between RELAX and SYSML/KAOS, the DEP uncertainty factor of RELAX is equivalent to the *Impact* of SYSML/KAOS.
– For the tools available for each approach, SYSML/KAOS has a tool called SYSML/KAOS editor, SYSML has a number of tools e.g. eclipse [2], papyrus [3], topcased [4] etc. and for RELAX, we have developed eclipse based RELAX COOL editor [12].

The table in Figure 6.5 shows the correspondence between different concepts treated by each approach. Based on this, we have developed RELAX2SYSML/KAOS editor which map the RELAX uncertainty factors to SYSML/KAOS concepts.

6.4.2 Uncertainty Factors/Impacts

RELAX Uncertainty factors especially ENV and MON are particularly important for documenting whether the system has means for monitoring the important aspects of 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 monitors its environment. Having said this, SYSML/KAOS can complement RELAX by injecting more information in the form of *positive/negative* and *direct/indirect* impacts. The grammar of RELAX acts as a meta-model for our RELAX COOL editor, while SYSML/KAOS has extended the meta-model of SYSML with goal concept. As both meta-models are close to the SYSML meta-model, we have bridged RELAX and SYSML/KAOS using our proposed approach.

6.4.3 Verification of Ambient System's Properties through Formal Methods

Using our proposed approach, we provide a strong consistency between models. This can be ensured thanks to the use of formal methods that provide verification tools for the properties verification and model simulation of SAS. We have integrated OMEGA2/IFx for properties verification and model simulation of these systems in our proposed approach. By doing this, we bridge the gap between the requirements phase and the initial formal specification phase.

2. http://www.eclipse.org/
3. http://www.papyrusuml.org
4. http://www.topcased.org/

Concepts/Approaches	SysML/KAOS	SysML	RELAX
Requirements Description	AbstractGoal ElementaryGoal	Textual Requirements	Relaxed Requirement ENV
Monitoring	Contribution Goal	<<satisfy>>	MON
Relationship	Contribution Nature: Positive Negative Contribution Type: Direct (Explicit) Indirect (Implicit)	<<verify>> <<refine>>	REL
Dependency/Impact	Contribution Nature: Positive Negative Contribution Type: Direct (Explicit) Indirect (Implicit)	<<derive>> <<contain>>	DEP: Positive Negative
Tools	Eclipse based SysML/KAOS Editor	Eclipse/Papyrus/Topcased/ …..	Eclipse based COOL RELAX editor

Figure 6.5: Relationship b/w SysML/Kaos SysML and Relax

6.4.4 Discussion

Both Relax and SysML/Kaos are complementary for each other [4]. Relax can take benefit from the *direct/indirect* and *positive/negative* contributions and impacts of the SysML/Kaos approach. To find a correlation between the two approaches, we have proposed a correspondence table which shows how the two approaches deals with different concepts. Based on this correlation table, we model the requirements of two case studies (section 7.1.1 and section 7.3.1 of chapter 7). Another important aspect is to provide a means to verify the properties of SAS. In our proposed approach, we treat properties verification through the use of OMEGA2/IFx profile and toolset. We verify three properties of the AAL system [5] which is discussed in section 7.2.2 of chapter 7.

6.5 Tools Support

In the previous section, I described our proposed approach for the requirements modeling and properties verification of SAS. The proposed approach requires different tools and processes to provide an integrated approach in order to take into account the different aspects of SAS. We have developed these tools during the course of this research work. In this section, I introduce these tools that validates our proposed approach.

Figure 6.6: Relax Grammar

6.5.1 DSL for RELAX

Domain Specific Languages (DSLs) are small languages, focused on a particular aspect of a software system [32]. [90] define a DSL as:

> *"A domain specific language is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain."*

As DSLs are conceived for a specific domain, it is easy to absorb the main concepts and features inherent to this domain , and bring them to the language constructors [51]. The most claimed advantage of using DSLs is the possibility of integrating domain experts in later stages of the software development life-cycle [38].

The need for a DSL for RELAX is the result of the motivation to bridge the gap between requirements even in a formatted version such as RELAX statements and the overall system model [2]. For the development of DSL for RELAX, Xtext is used. Xtext is a framework for the development of DSLs and other textual programming languages and helps in the development of an Integrated Development Environment (IDE) for the DSL. Some of the IDE features that are either derived from the grammar or easily implementable are: syntax coloring, model navigation, code completion, outline view, and code templates. For this purpose, Xtext is used for the code editor generation of RELAX.

Xtext comes up with a handful of wizards, with the help of which one can define their DSL. After defining the DSL, the next step to follow is to define the grammar. The grammar specifies the meta-model and the concrete syntax for the DSL. The grammar of RELAX is used as a meta-model for the DSL. Here we introduce the MDE concepts. To specify the grammar,

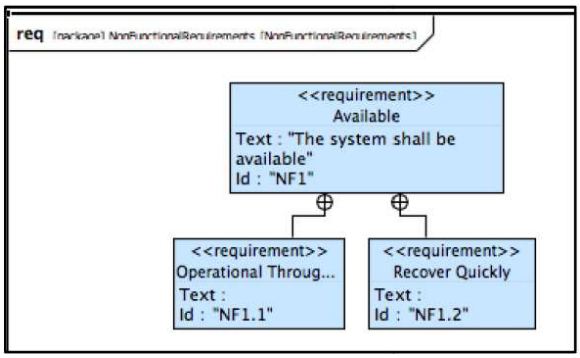

Figure 6.7: Generated Requirement Diagram

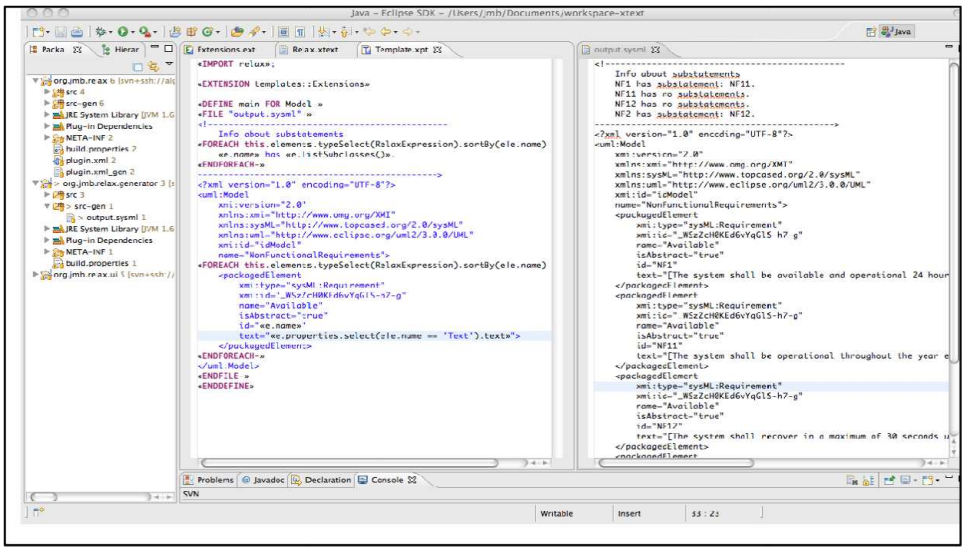

Figure 6.8: Generated Code

Xtext grammar language is used. Figure 6.6 shows a snapshot of the Relax grammar written in Xtext.

6.5.1.1 Requirements Transformation

To show the DSL for Relax; we start by taking some NFRs. These requirements are taken from a case study for a car crash crisis management system [49].

- Non Functional Requirement 1: The system shall be available and operational 24 hours a day.
- Non Functional Requirement 1.1: The system shall be operational throughout the year except for a maximum downtime of 2 hours every 30 days for maintenance.
- Non Functional Requirement 1.2: The system shall recover in a maximum of 30 seconds upon failure.

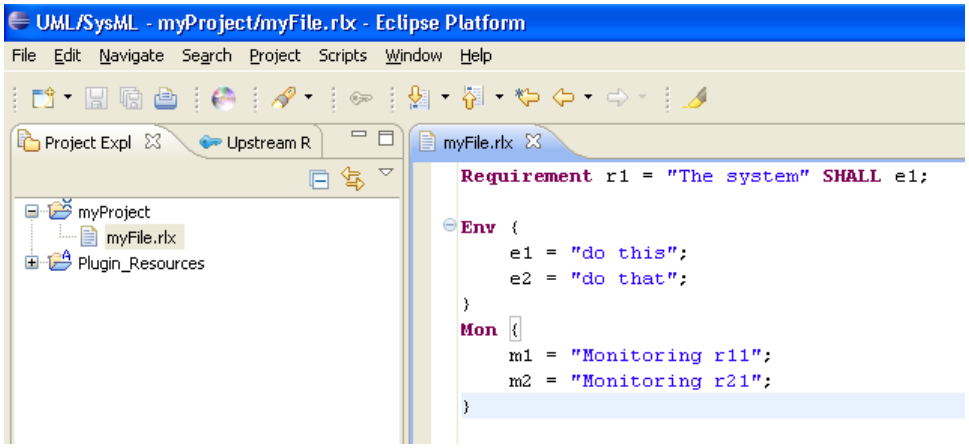

Figure 6.9: RELAX File

These requirements are shown in text format and we are interested in transforming these textual requirements into graphical form so that the gap between the textual requirements and the overall system model should be bridged. NFRs in textual format are transformed into graphical format with the help of RELAX grammar. Figure 6.7 shows a snapshot of the generated requirement diagram.

6.5.1.2 Code Generator

Among the benefits of Xtext, we have benefited from the code generation framework that is automatically generated from the grammar. A code generator has been written that is capable of processing the models created with the DSL editor. Figure 6.8 shows the generated code. On the right side of the screenshot, the generated SysML requirements are shown.

6.5.2 RELAX Editor

For the generation of RELAX editor, Xtext is used. The RELAX grammar is used as a meta-model for this editor. The RELAX meta-model is generated by Xtext which we call RELAX.ecore. Figure 6.9 shows an example of the RELAX file with uncertainty factors. The RELAX file is represented with an extension *.rlx*. Once we have the *.rlx* file, we can transform it into an XMI model. The XMI model can then be manipulated and will serve us for the model transformation from RELAX to SysML/KAOS as explained in the next section. Figure 6.10 shows the generated RELAX model in XMI format. The XMI is an OMG standard for exchanging meta-data information via Extensible Markup Language (XML). It can be used for any meta-data whose meta-model can be expressed in MOF. Effectively, the XMI format standardizes how any set of meta-data is described and requires users across many industries and operating environments to see data the same way. All the RELAX operators are implemented in RELAX editor.

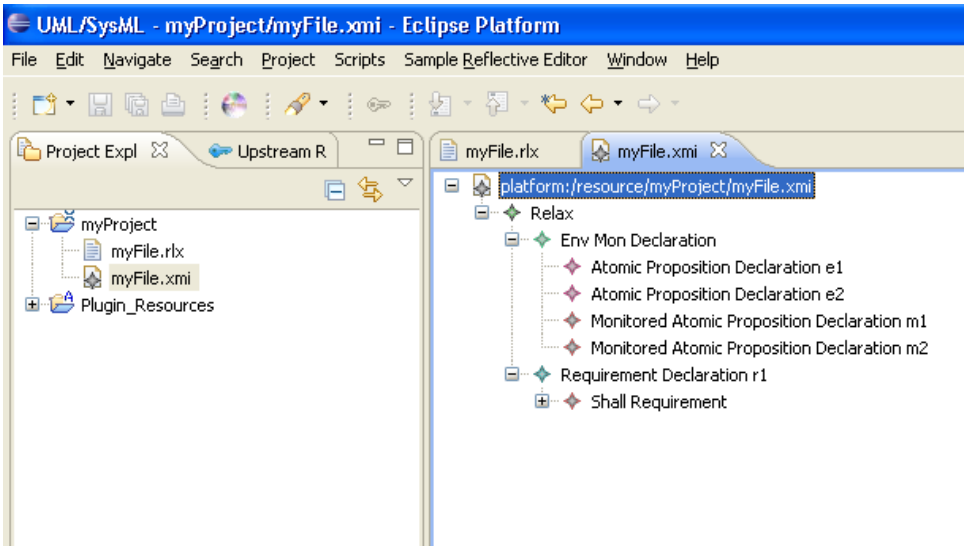

Figure 6.10: RELAX Model Example

```
-- @path RelaxMetaModel=/Relax2SysMLKAOS/MetaModel/Relax.ecore
-- @path SysMLKAOSMetaModel=/Relax2SysMLKAOS/MetaModel/SysMLKAOS.ecore
```

Figure 6.11: Meta Models Paths

6.5.3 RELAX to SysML/Kaos Transformation

The use of model transformation languages in GORE helps in the correspondence between different approaches. The need for model transformation can be attributed to: refine an organizational level model to a system level model; compare different approaches and decide which one is more expressive for a particular domain or to an organization culture; facilitate the communication between professionals specialized in different approaches.

In [65], a model transformation framework is proposed, called MDGore, to transform i* models into Kaos models through rules defined in ATL [5]. To achieve this, i* and Kaos models are specified using tools that were developed in Eclipse platform, LDE i* [69] and modularKAOS framework [29]. The application of the transformation rules is done at abstract syntax level, defined in the meta-model of each one of the frameworks, and uses the ATL model transformation language.

In our approach, we want to transform Relax-ed requirements uncertainty factors into SysML/Kaos goal concepts. This transformation will help in taking into account the adaptability features associated with SAS in the form of uncertainty factors of Relax-ed requirements and then modeling these requirements in SysML/Kaos. In this way, we can benefit from the advantages offered by GORE. For this purpose, the Relax and SysML/Kaos meta-models are used. The correlation between these two concepts is shown in chapter 6.

5. http://wiki.eclipse.org/ATL/User_Guide

```
module Relax2SysMLKAOS;
create OUT : SysMLKAOSMetaModel from IN : RelaxMetaModel;
```

Figure 6.12: In and Out Declarations

```
rule RelaxedRequirement2AbstractGoal {
	from
		relaxedRequirement : RelaxMetaModel!RelaxedRequirementDeclaration
	to
		abstractGoal : SysMLKAOSMetaModel!AbstractGoal (name <- relaxedRequirement.name)
}
```

Figure 6.13: Relaxed Requirement to Abstract Goal Mapping

```
rule AtomicPropositionDeclaration2ElementryGoal {
	from
		env : RelaxMetaModel!AtomicPropositionDeclaration
	to
		elementryGoal : SysMLKAOSMetaModel!ElementryGoal (name <- env.name)
}
```

Figure 6.14: ENV to Elementary Goal Mapping

```
rule MonitoredDeclaration2ContributionGoal {
	from
		mon : RelaxMetaModel!MonitoredAtomicPropositionDeclaration
	to
		goal : SysMLKAOSMetaModel!Goal (name <- mon.name)
}
```

Figure 6.15: MON to Contribution Goal Mapping

6.5.3.1 ATL Rules

ATL is a model transformation language and toolkit. It provides a way to produce a number of target models from a set of source models. An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. The generation of target model elements is achieved through the specification of transformation rules. ATL defines two different kinds of transformation rules: the matched and the called rules. A matched rule enables to match some of the model elements of a source model, and to generate from them a number of distinct target model elements. A called rule has to be invoked from an ATL imperative block in order to be executed. ATL imperative code can be defined within either the action block of matched rules, or the body of the called rules.

Figure 6.11 shows the two meta-models path. In Figure 6.12, after the module name declaration, the meta-model ID with *create* and *from* keywords are specified.

6.5.3.2 Mapping between RELAX and SysML/Kaos Elements

Here, we present the relationship between RELAX and SysML/KAOS elements. This relationship is established based on the concepts of the two approaches introduced in chapter 5 and based on the abstract syntax of each approach. The RELAX abstract syntax is

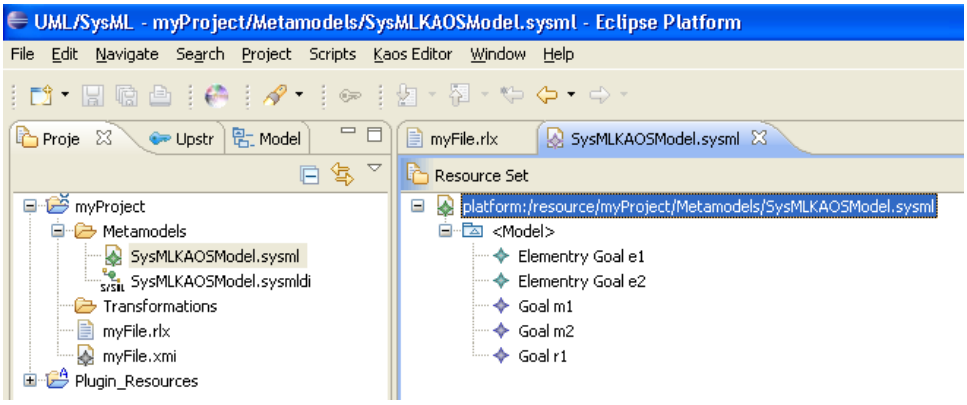

Figure 6.16: SysML/Kaos Model

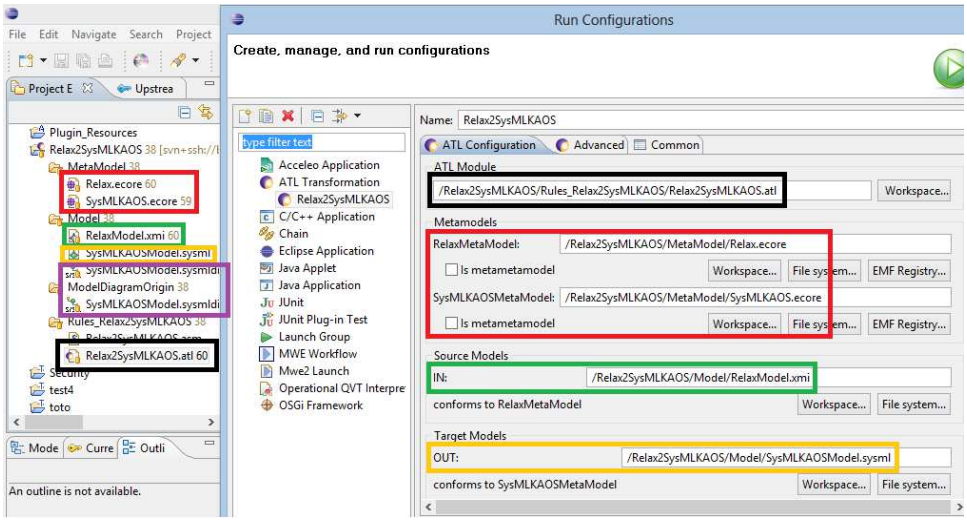

Figure 6.17: ATL Transformation Hierarchy

defined in the Relax meta-model. In turn, the SysML/Kaos abstract syntax is defined in the SysML/Kaos meta-model.

Figure 6.5 shows the mapping between the two concepts. For the ATL transformation rules, a Relax-ed requirement is mapped to an *Abstract Goal* as shown in Figure 6.13, an ENV is mapped to an *Elementry Goal* as shown in Figure 6.14 and MON is mapped to *Contribution Goal* as shown in Figure 6.15. Figure 6.16 shows the generated SysML/Kaos model after the application of ATL rules.

Figure 6.17 shows the ATL transformation hierarchy. In black box, we have the ATL file path; in red, we have the source and target meta-models paths i.e. Relax and SysML/Kaos respectively; in green, the path of the Relax model in XMI format which is the source model and which confirms to the Relax meta-model; in yellow, we have the SysML/Kaos model path which is the target model and which confirms to the SysML/Kaos meta-model; this is the model that will be created by the ATL transformation. Figure 6.18 shows the SysML/Kaos

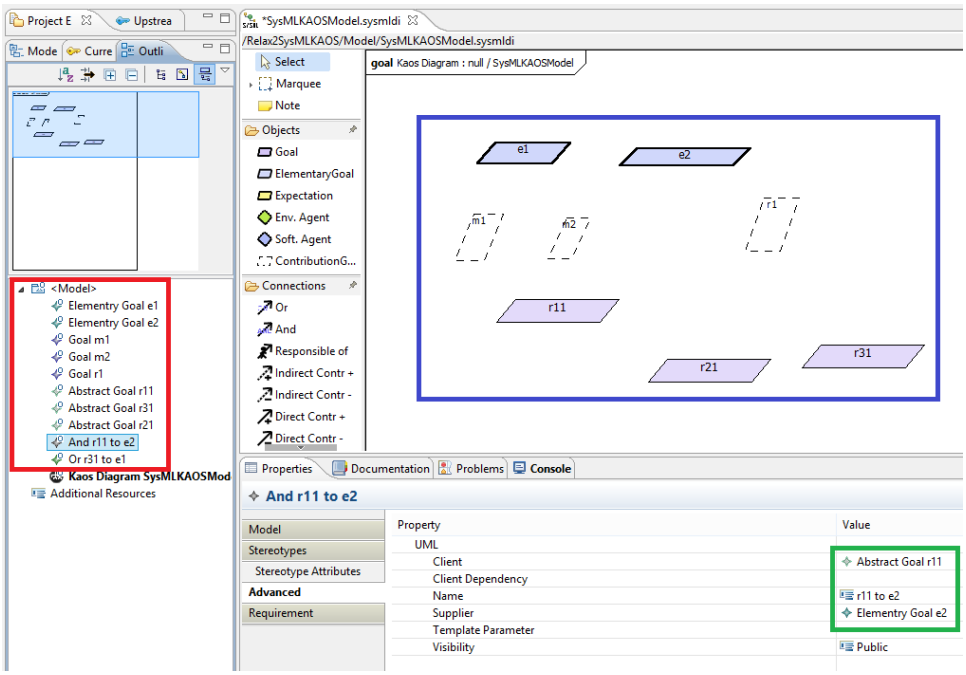

Figure 6.18: Generated SysML/Kaos Model using ATL Transformations

model.

6.6 Conclusion

In this chapter, I described the overall view of our proposed approach. We identified some problems related to requirements modeling and properties verification of SAS. Requirements modeling using SysML/Kaos does not take into account the adaptability features associated with SAS. For properties verification using OMEGA2/IFx, we are exposed to the state explosion problem which comes with model checking techniques. The identified problems are taken into account in our proposed approach by providing processes and tools to solve it. The proposed approach also highlights the need for an integrated tooling environment which is based on different correlations that exists between the basic concepts that we treated in this research work i.e. Relax and SysML/Kaos. We base our proposition on using MDE techniques for the requirements modeling and properties verification of SAS. For the tools support that we provide, we have developed: a DSL for Relax which takes requirements in textual format and transforms it into SysML requirements diagram, Relax COOL editor which takes into account the uncertainty factors associated with Relax-ed requirements and which provide a mechanism to automate Relax-ed requirements and finally Relax2SysML/Kaos editor which is based on ATL transformations, the tool helps in mapping the Relax concepts to SysML/Kaos concepts. These tools are developed during the course of this thesis work.

CHAPTER 7

Experimentation And Analysis

Contents

The ongoing aging process that is noticeable in all industrialized societies, especially in North America and Europe, has raised serious problems with the growing share of handicapped and elderly people being unable to conduct their normal lives at home, thereby becoming more and more isolated from family, friends, and public life. The goal of AAL solutions is to apply ambient intelligence technology to enable people with specific demands, e.g. handicapped or elderly, to live in their preferred environment [13]. In order to achieve this goal, different kinds of AAL systems can be proposed and most of them pose reliability issues and describe important constraints upon the development of software systems [25].

To validate our proposed approach, we model the requirements of two different case studies. The first case study is about an AAL [1] home which ensures the health of a *Patient*. The second case study is about bCMS [2] which is responsible for coordinating the communication between a Fire Station Coordinator (FSC) and a Police Station Coordinator (PSC) to handle a crisis in a timely manner.

7.1 Requirements Modeling of the AAL Case Study

7.1.1 The AAL Case Study

The AAL case study describes a smart home for assisted living. Figure 7.1 shows an excerpt of the case study which highlights the need to ensure *Patient*'s health in the AAL home. Advanced smart homes, such as Mary's AAL, rely on adaptivity to work properly. For example, the sensor-enabled cups may fail, but since maintaining a minimum of liquid intake is a life-critical feature, the AAL should be able to respond by achieving this requirement in some other way.

In our proposed approach (chapter 6), we start by applying the Relax process on FRs and NFRs of SAS. The Relax process results in the distinction of Relax-ed requirements which are adaptable and invariant requirements which are fixed. Figure 7.2 shows an example of Relax-ed requirement from the Mary's AAL home, which results from the application of the Relax process on the traditional requirement: *The Fridge shall read, store and communicate RFID information on food packages*. In the following, I show the modeling of the AAL case study Relax-ed requirements using SysML/Kaos. We have mapped the Relax-ed requirements to SysML/Kaos goal concepts through a correlation table as shown in Figure 6.5.

7.1.1.1 AAL Case Study Requirements Modeling

We have merged Relax and SysML/Kaos in order to obtain a detailed and strong requirements description of the system and its context. For modeling the requirements of AAL system, we start by identifying an NFG: *Reliablility [AAL System]* which in Relax terminology is a Relax-ed requirement. This goal must be refined progressively using goal

1. http://www.iese.fraunhofer.de/fhg/iese/projects/med_projects/aal--lab/index.jsp
2. http://cserg0.site.uottawa.ca/cma2012/CaseStudy.pdf

Mary is a widow. She is 65 years old, overweight and has high blood pressure and cholesterol levels. Mary gets a new intelligent fridge. It comes with 4 temperature and 2 humidity sensors and is able to read, store, and communicate RFID information on food packages. The fridge communicates with the Ambient Assisted Living (AAL) system in the house and integrates itself. In particular, it detects the presence of spoiled food and discovers and receives a diet plan to be monitored based on what food items Mary is consuming.
An important part of Mary's diet is to ensure minimum liquid intake. The intelligent fridge partially contributes to it. To improve the accuracy, special sensor-enabled cups are used: some have sensors that beep when fluid intake is necessary and have a level to monitor the fluid consumed; others additionally have a gyro detecting spillage. They seamlessly coordinate in order to estimate the amount of liquid taken: the latter informs the former about spillages so that it can update the water intake level. However, Mary sometimes uses the cup to water flowers. Sensors in the faucets and in the toilet also provide a means to monitor this measurement.

Figure 7.1: AAL Case Study

Relaxed Requirement:
The fridge SHALL detect and communicate information with AS MANY food packages AS POSSIBLE

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)

Figure 7.2: RELAX Requirement Example

models to obtain final requirements of the system so that they can be satisfied by *Contribution Goal*.

7.1.1.2 High Level Goal Model

From the AAL system problem statement, we have identified *Reliability[AAL system]* as a non functional high level goal. In fact, one of the expected qualities of the system is to run reliably. This is very important for several reasons and particularly because frequent visit of technician could be a factor of disturbance for *Mary* and unfeasible due to the large number of AAL houses across the world. In SysML/Kaos, an NFG can be written in the form of: *NFGType [Topic]* where the attribute *NFGType* specify the type of NFG and the attribute *Topic*

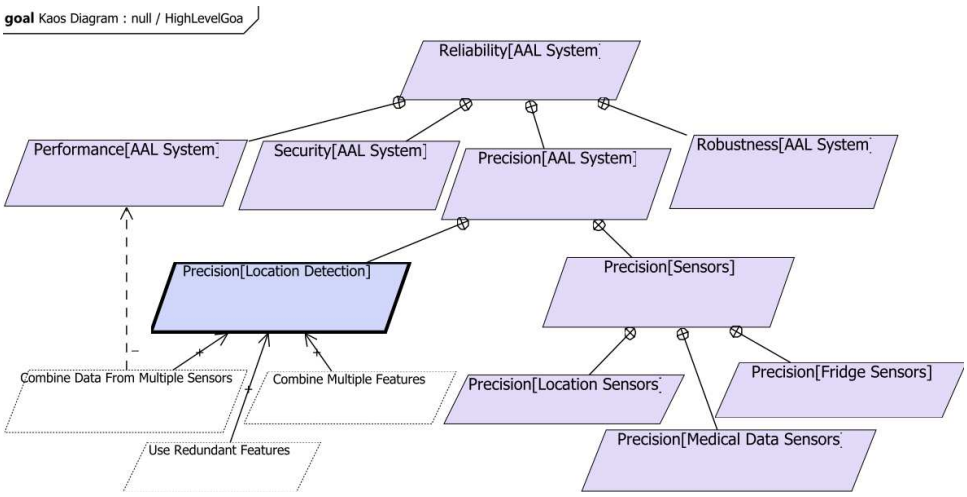

Figure 7.3: High Level Goal Model

represent the domain element effected by this type of requirement. The refinement of an NFG can be either refinement by type *NFGType* or refinement by subject *Topic*. The high level goal *Reliability [AAL System]* is AND-refined into four sub goals using refinement by type: *Precision [AAL System]*, *Security [AAL System]*, *Robustness [AAL System]* and *Performance [AAL System]*. Each sub goal can be further refined until the refinement stops and we reach an *Elementary Goal* which can then be assigned to a *Contribtuiot Goal*. The sub goal *Precision [AAL System]* is AND-refined into two sub goals: *Precision [Location Detection]* and *Precision [Sensors]* using refinement by subject. The sub goal *Precision [Sensors]* is then AND-refined into three sub goals using refinement by subject: *Precision [Location Sensors]*, *Precision [Medical Data Sensors]* and *Precision [Fridge Sensors]*. The sub goal *Precision [Location Detection]* can be satisfied by a positive and direct contribution by one of the following *Contribution Goal*: *combine data from multiple sensors*, *combine multiple features* and *use redundant features*. The *Contribtuiot Goal combine data from multiple sensors*, contribute indirectly and negatively to the satisfaction of sub goal *Performance [AAL System]*. Figure 7.3 shows the high level goal model of AAL.

7.1.1.3 Low Level Goal Model

In order to further extract new goals from the AAL system, we identify another goal *Security [fridge input data]* which is an abstract NFG that can be AND-refined into three sub goals using refinement by type: *Confidentiality [fridge input data]*, *Integrity [fridge input data]* and *Availability [fridge input data]*. Similarly, the sub goal *Availability [fridge input data]* can be refined into two sub goals using refinement by subject: *Availability [Storing RFID information]* and *Availability [Sensors data]*. Consider for example the *Elementary Goal Confidentiality [fridge input data]*, a possible solution to meet this goal is to use a code *PIN*; another solution is to *require an additional identifier*. These two solutions represent thus direct and positive contribution to this goal. Similarly, *having high-end sensors* contributes directly and positively

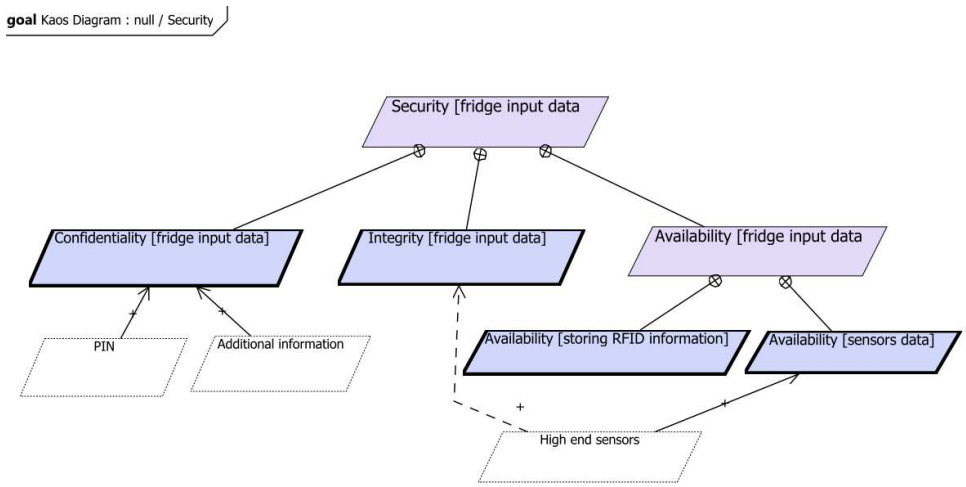

Figure 7.4: Security Goal Model

to the goal *Availability [Sensors data]*, and may contributes indirectly and positively to *Integrity [fridge input data]*. Figure 7.4 shows the security goal model of AAL.

7.1.2 Discussion

Our proposition is based on the fact that both RELAX and SysML/KAOS are complementary for each other [4]. Both these approaches treats NFRs. We provide tools and processes in our proposed approach to take benefit of the two approaches. The correlation table of Figure 6.5 is used for the correspondence between RELAX and SysML/KAOS, based on this, we have modeled RELAX-ed requirements of the AAL case study. Another important aspect is to provide a means to verify the properties of the AAL system. In our proposed approach, we treat properties verification through the use of OMEGA2/IFx profile and toolset. We verify three properties of the AAL system [5]. The concepts related to properties verification is discussed in the next section.

7.2 Properties Verification of the AAL system with OMEGA2/IFx Profile and Toolset

The specification and verification of NFRs in the early stages of the AAL development cycle is a crucial issue [68]. These systems require clear and precise specifications in order to describe the system behavior and its environment. The formal specification of the system behavior supported by mathematical analysis and reasoning techniques improve their development process and enable the verification of these systems.

Formal methods provide tools to verify the consistency and correctness of a specification with respect to the desired properties of the system. It is important as the development of an AAL system involves different technologies e.g. medical services, surveillance cameras,

intelligent devices etc. and it will provide a strong consistency between models. For this reason, we are interested to use these methods to prove some of the properties of the system before the development even starts. We provide a mechanism to bridge the gap between the requirements phase and the initial formal specification phase by using MDE techniques and more specifically using model checking techniques in our proposed approach. To model the AAL system, we used Rational Rhapsody 7.5.2 in combination with OMEGA2 profile [71] which is an executable Uml/SysML profile used for the formal specification and validation of critical real-time systems. The OMEGA2 Profile is supported by the IFx toolset which provides mechanisms for the model simulation and properties verification of the AAL system. The OMEGA2/IFx approach has been applied for the verification and validation of industry grade models [30] providing interesting results.

The OMEGA2 Uml/SysML profile defines the semantics of Uml/SysML elements providing the means to model coherent and unambiguous system models. In order to make the models verifiable, it presents as extension the *observers* mechanism for specifying dynamic properties of models. The OMEGA2 Uml/SysML Profile is implemented by the IFx toolbox [73] which provides static analysis, simulation and timed automaton based model checking [23] techniques for validation.

7.2.1 Modeling the AAL system with OMEGA2 Profile

This section shows the AAL system specification and architecture. It shows the structural diagrams i.e. IBD, BDD, SMD and behavioral diagrams and the properties defined on the AAL system that we verified using the OMEGA2/IFx profile and toolset.

7.2.1.1 Using OMEGA2 Profile

OMEGA2 models use a profile and a predefined library provided with the tool (OMEGA2.sbs and OMEGA2Predefined.sbs). Any other Uml2.2 or SysML1.1 editor supporting profiling and exporting in the XMI 2.0 standard compatible with Eclipse *ecore* can be used for OMEGA2 models. In OMEGA2, ports are unidirectional, meaning that we can only send messages through the port or receive messages through the port. If a component can send and receive messages, we have to model this with at least two ports. Every port has to define a contract. A contract is an interface containing operations definition (without the body) and signals reception for which the block and/or the port knows how to react. If we have two or more interfaces defining the needed requests, we need to define a new interface that inherits from the others. This interface has to be stereotyped with *InterfaceGroup* and it defines the type of the port. A port's usual behavior is to forward messages according to its direction. A required port may be modeled either using the stereotype reversed or using the reversed checkbox in the port's properties. In this case the requests are transferred to the environment. If the port is owned by a block and has to forward messages to the block, the port must be set declared as provided port (i.e. in Rhapsody a provided port is a behavior port).

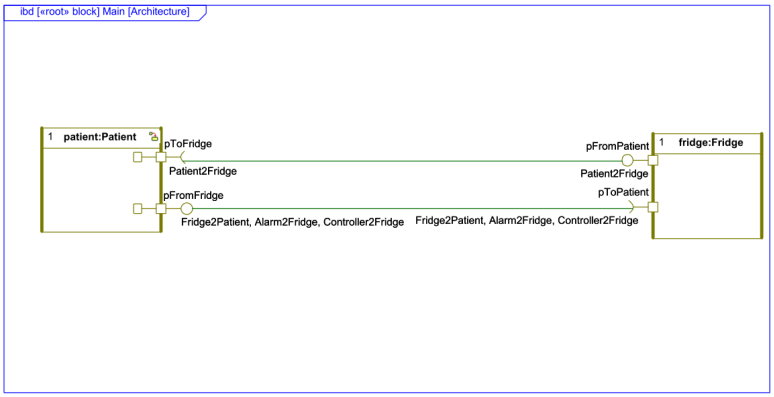

Figure 7.5: Main Internal Block Diagram

7.2.1.2 System Specification

First, we start by taking into account the structural part of the AAL system. Those parts are considered that are concerned with the daily calories intake of the *Patient* in the AAL house. The AAL system is composed of *Fridge* and *Patient*, these parts are modeled along with the interaction that takes place between them. The *Fridge* partially contributes to the minimum liquid intake of the *Patient*; it also looks at the calories consumption of the *Patient* as the *Patient* needs not to exceed it after a certain threshold.

7.2.1.3 System Architecture

Figure 7.5 shows the main IBD. The important parts of the AAL system are *Patient* and *Fridge*. Figure 7.6 shows the blocks with ports that are identified in the AAL system i.e. *Fridge*, *Display*, *Alarm*, *Controller*, *Food* and *Patient*. The *Fridge* interacts with the AAL system. Figure 7.7 shows the IBD for the *Fridge* block. Each of the four blocks behaviors is modeled in a separate SMD. A *Fridge* is composed of *Display*, *Alarm*, *Controller*, and *Food* blocks. The block *Food* contains information about the *Food* items in the *Fridge*, the calories contained by each item, the total number of calories the *Patient* has accumulated and the calories threshold that should not be surpassed. The *Fridge Display* is used to show the amount of calories consumed by the *Patient*. The *Alarm* is activated in case the *Patient*'s calories level surpasses a certain threshold.

The communication between different blocks takes place through ports. A port bears a type. In OMEGA2, the type of a port must be an Interface. The type specifies the set of requests (operation calls and/or signals) that are transferred between parts (components) by means of ports and connectors. In Figure 7.5, the *Patient* block has a standard port named *pToFridge*. This port has a contract named *Patient2Fridge* and is acting as a provided interface of the *Patient* block. The Interface *Patient2Fridge* defines an operation *eat(int item, int quantity)*. This interface is then used as a type of *pToFridge* port. At the same time, the *Patient* block has a required interface named *pFromPatient*.

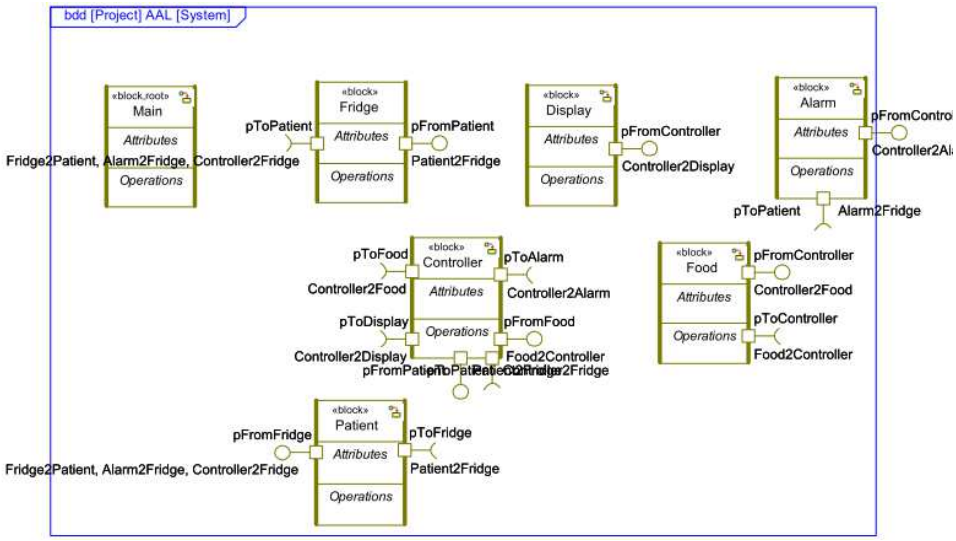

Figure 7.6: AAL System Blocks

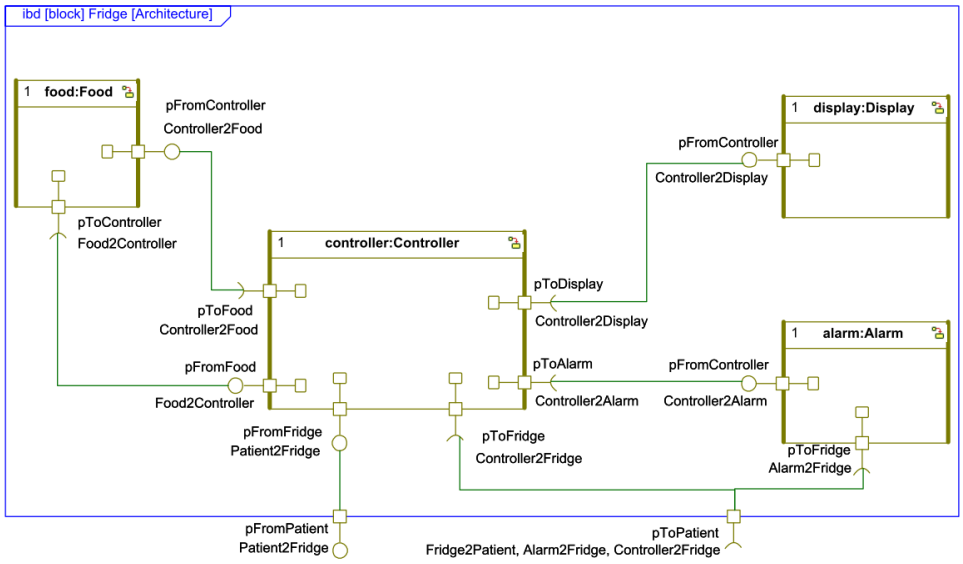

Figure 7.7: Fridge Internal Block Diagram

Figure 7.8 shows the SMD for the *Patient* block. In the *Patient* SMD, the exchange of information between *Patient* and *Fridge* takes place. The number and quantity of each item present in the *Fridge* are identified. If a certain product still present in the *Fridge* is chosen by the *Patient* then the information is communicated with the *Fridge*. Otherwise the *Fridge* is empty and the *Patient* will wait for the *Fridge* to be refilled. Also, if the *Alarm* of the *Fridge* is raised due to high intake of calories, the *Patient* stops eating and waits for the system to be unblocked.

The *Food* block models the knowledge of the *Fridge* about what it contains. Figure 7.9

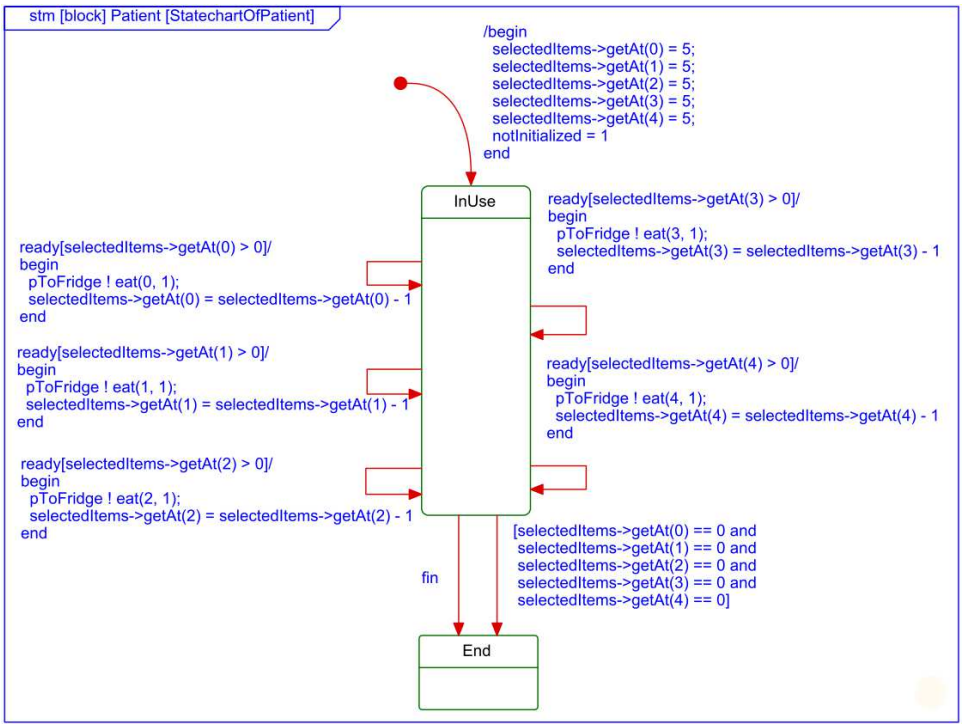

Figure 7.8: Patient State Machine Diagram

shows its SMD. Here, the number of items and the amount of calories associated with each item present in the *Fridge* are defined. Then the total number of calories accumulated by the *Patient* is calculated. If the total number of calories is greater than or equal to the maximum calories allowed for the *Patient*, then a message is sent and the *Alarm* is raised or if the total number of calories is greater than the maximum calories allowed minus 500, then the *Patient* is warned with a message that the calories level is approaching the maximum amount of calories allowed.

7.2.2 Properties Verification of the AAL system

The properties of AAL system that are modeled and verified are obtained after RELAX process is applied on its traditional requirements. Below are the properties that we verified [5].

7.2.2.1 Traditional/RELAX-ed Requirement

The Fridge SHALL detect and communicate with Food packages

RELAX-ed version of this requirement is as follows:

Property 1 : The Fridge SHALL detect and communicate information with AS MANY Food packages AS POSSIBLE

Below are the uncertainty factors associated with the given RELAX-ed requirement.

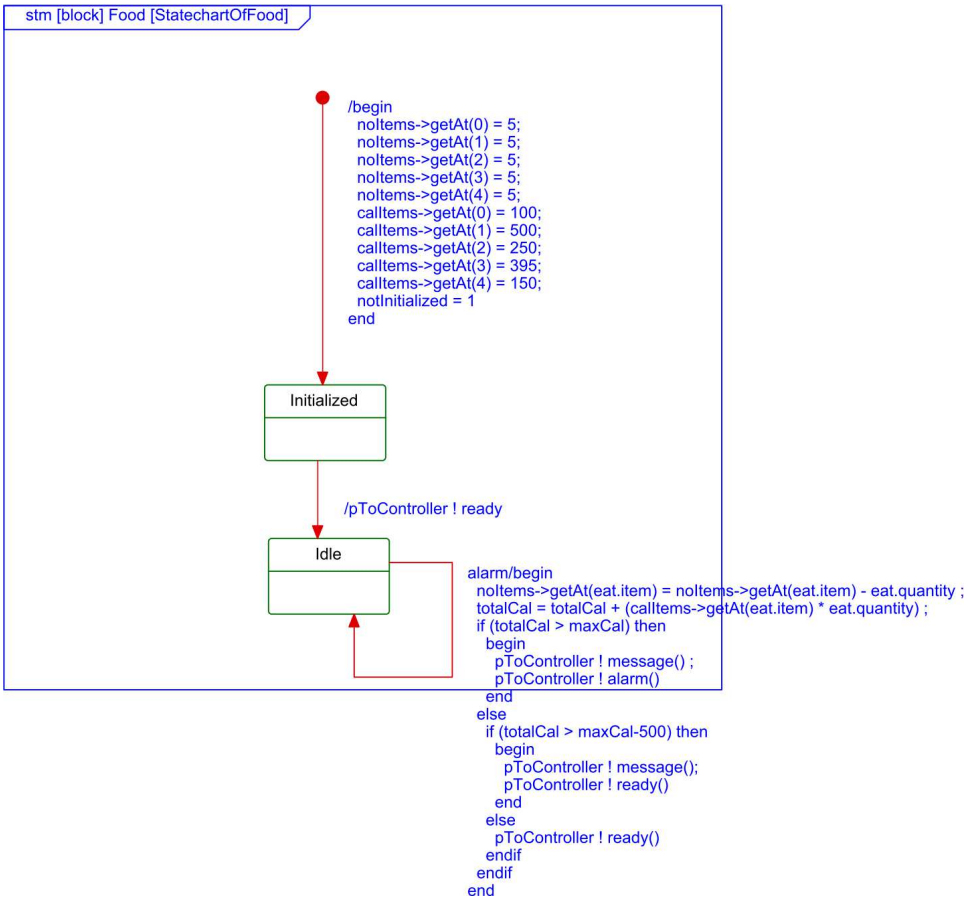

Figure 7.9: Food State Machine Diagram

- ENV: *Food* locations, *Food* 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)

The satisfaction of this requirement contributes to the balanced diet of the *Patient*. The choice of this property for verification is motivated by the fact that it is important for the AAL system to know about how many *Food* items are present in the *Fridge*. Figure 7.10 shows the SMD of the Property 1. Here, the first task is to identify the number of items consumed by the *Patient* and the total number of items in the *Fridge*. Then the identity of the *Patient* is verified, if the person is identified as the *Patient*, the next step is to calculate the number of items consumed by the *Patient*. After this, the number of items left in the *Fridge* is calculated which is equal to the sum of all the items present in the *Fridge*. Then in the last step, we calculate if ((total number of items - number of items consumed - number of items left) >-1) and ((total number of items - number of items consumed - number of items left) <1), it means

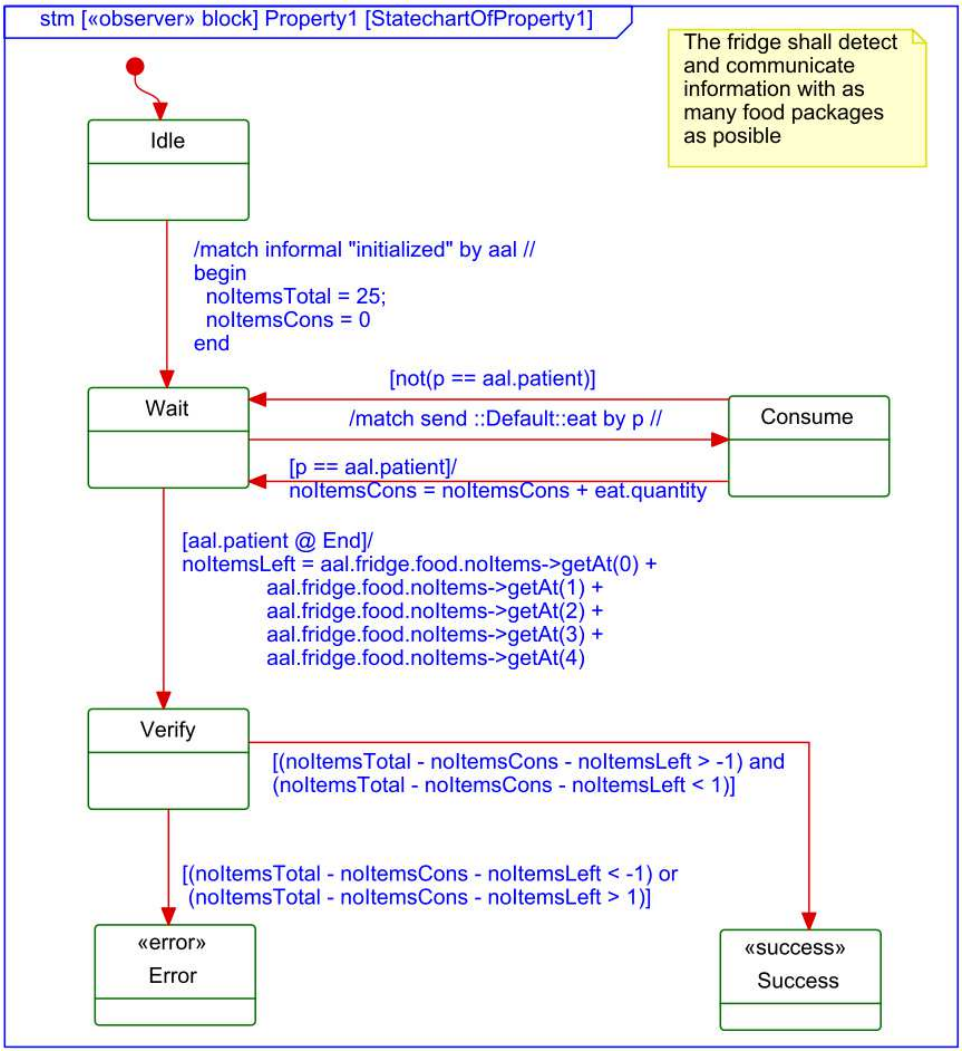

Figure 7.10: Property1 State Machine Diagram

that we have reached the «*success*» state by having information about all the items present in the *Fridge*, i.e. it should be 0 (which means that there is no information loss). Inversely, if it is less than -1 and greater than 1, then it means that we are missing information about some of the items present in the *Fridge* and the *observer* passes into the «*error*» state. Following are the two invariant requirements that we verified.

7.2.2.2 Invariant Requirement 1

Property 2 : The System SHALL raises an Alarm if the total number of calories is equal to or more than a certain threshold

The information gathered from the *Fridge* is used to calculate the amount of calories accumulated by the *Patient*. The total amount of calories accumulated by the *Patient* should be

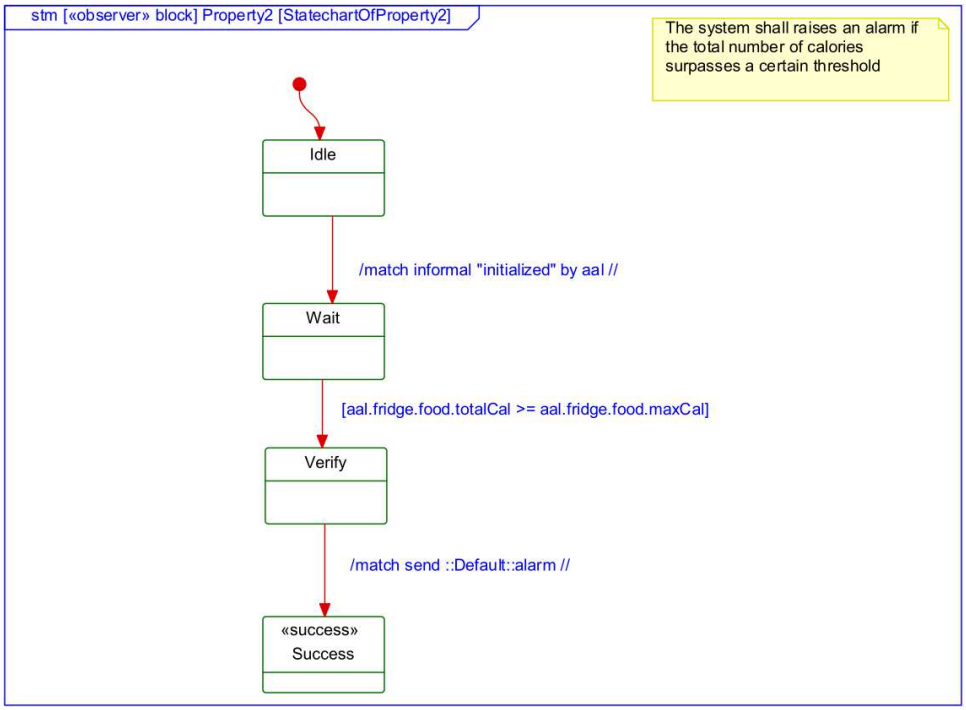

Figure 7.11: Property2 State Machine Diagram

checked and kept at a certain threshold as the *Patient* needs a hyper caloric diet. Figure 7.11 shows the SMD of the Property 2. Here, we check that the *Alarm* shall be raised as soon as the total number of calories equals or surpasses the maximum allowed calories for the *Patient*, thats the reason why, we have only one state i.e. a «*success*» state. This requirement is treated as invariant because if the number of calories equals or surpasses the threshold allowed and the *Alarm* is not raised, it will have serious consequences on the health of the *Patient*.

7.2.2.3 Invariant Requirement 2

Property 3 : The Alarm SHALL be raised instantaneously if the total number of calories surpasses the maximum calories allowed for the Patient

This property ensures that the *Patient* should stop eating as soon as the total number of calories surpasses the maximum calories allowed and that the *Alarm* should be raised. Figure 7.12 shows the SMD for this property. This requirement implies that the *Alarm* shall be immediately raised as soon as the total number of calories equals or surpasses the maximum calories allowed for the *Patient*. If it happens then the *Patient* should stop eating and we will reach a «*success*» state but if the *Patient* continues to eat, it means that we are reaching an «*error*» state. Here, we have modeled the two states which differentiate it from the first invariant requirement.

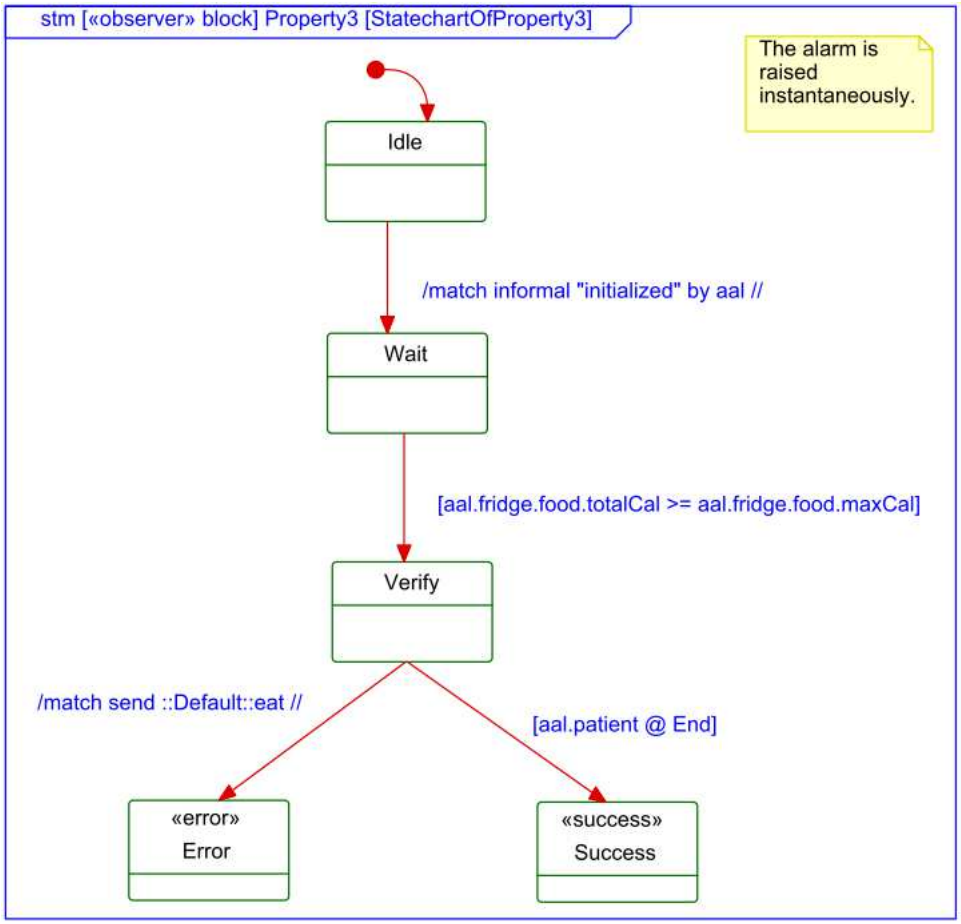

Figure 7.12: Property3 State Machine Diagram

```
[ahmad@topcased ~]$ uml2if -sysml -rhapsody -rhplang -eager AAL2.xmi
uml2if (OMEGA2) v2.0.1  (c) Verimag,IRIT 2009-2011
Analyzing input.
Please wait...
Success
Generated...
Preprocessed...
Indented...
Done.
[ahmad@topcased ~]$
```

Figure 7.13: XMI to IF Compilation

7.2.2.4 Verification Results

Untill now, the AAL system is modeled along with the properties to be verified on the model. We now show how to verify these properties using the IFx toolset. Figure 7.13 shows the snapshot of the compilation of the AAL model named AAL2. The AAL2 model is first exported into AAL2.xmi and then using the IFx toolset the AAL2.xmi is compiled into AAL2.if.

Figure 7.14 shows the snapshot of the compilation of AAL2.if into an executable file i.e.

Figure 7.14: IF to Executable file Compilation

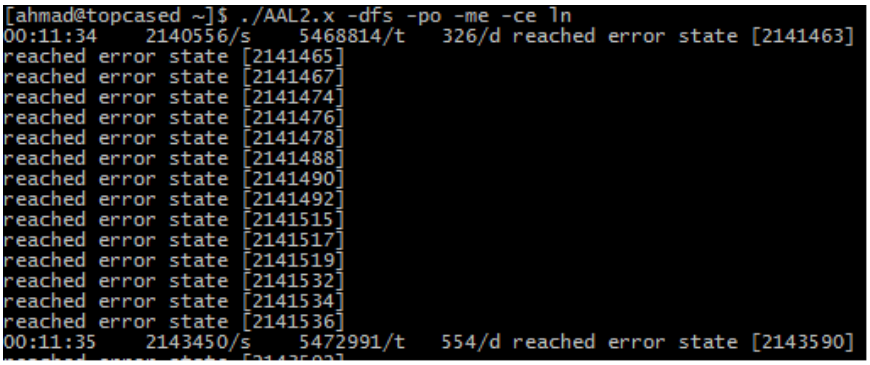

Figure 7.15: Model Checker results in Error Scenarios

AAL2.x. While verifying the AAL model, the model checker has found several error scenarios, as one can see in Figure 7.15. Any of the error scenario can then be loaded through the interactive simulation interface of the IFx toolset to trace back the error in the model and then correct it.

In order to debug a model, firstly we import it into the simulator as shown in Figure 7.16. We check the states of the *observers* in order to identify which property has not been satisfied. One can observe in Figure 7.17 that Property 3 fails. While checking the state of the entire system for this property, we discover that the «*error*» state contained the maximal allowed number of calories for the total number of calories consumed and subsequently eat requests are sent by the *Patient*. This implies that the *Alarm* function of the intelligent *Fridge* doesn't function properly. The *Alarm* function of the *Fridge* is strictly linked to its *Food* process. One can observe in the SMD of the *Food* block (Figure 7.9) that the *Alarm* is raised only if the total number of consumed calories is strictly superior than the maximum allowed; condition which doesn't satisfy the request that the *Alarm* is raised as soon as possible. The correction consists in raising the *Alarm* in case the total number of consumed calories is equal to the maximum allowed threshold. Once this error is corrected, the verification succeeds. Figure 7.18 shows

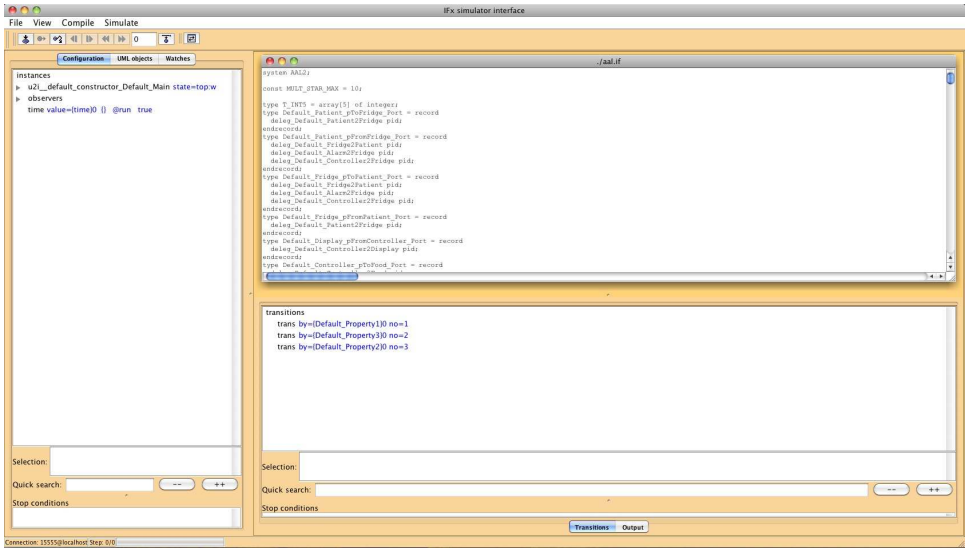

Figure 7.16: Initial Simulation Interface

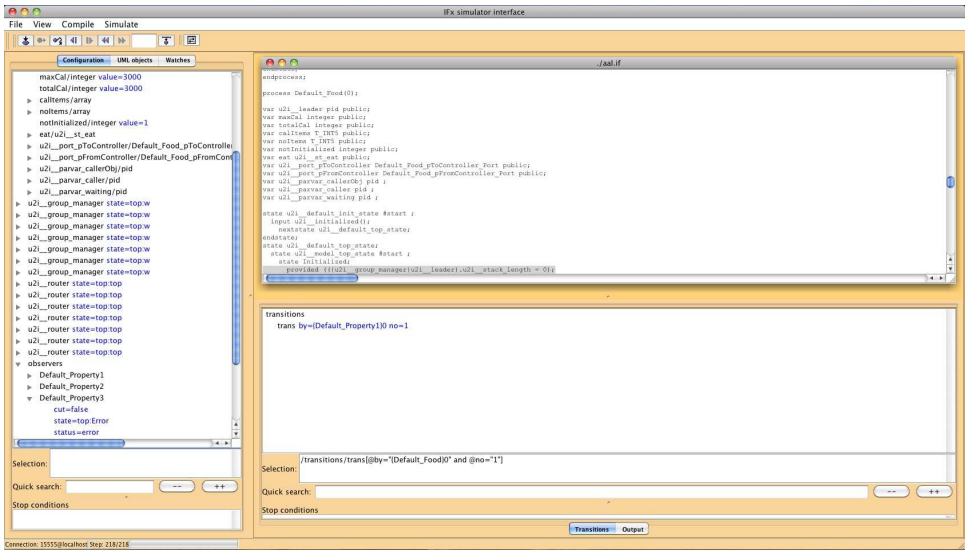

Figure 7.17: Error State Food Observer Simulation Interface

```
[ahmad@topcased ~]$ ./AAL2.x -dfs -po -me -ce 1n
01:46:03   17058583/s   42971642/t   179/d
```

Figure 7.18: Model checking successful

the result of the model checker on the correct model.

In the next section, I show the modeling of the bCMS requirements using our proposed approach.

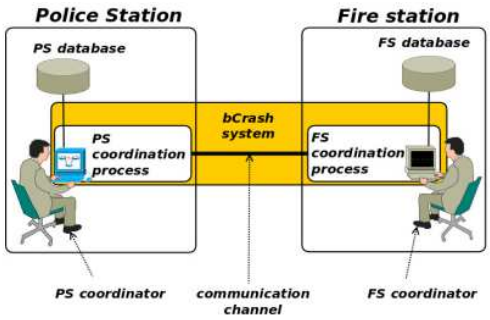

Figure 7.19: bCMS Case Study Overall View

7.3 Requirements Modeling of the bCMS Case Study

bCMS case study is responsible for coordinating the communication between an FSC and a PSC to handle a crisis in a timely manner.

7.3.1 The bCMS Case Study

We model the requirements of the bCMS case study using our proposed approach [3]. The usual process of requirements discovery is applied to extract FRs and NFRs. We then apply the Relax process on the FRs and NFRs of the bCMS case study to get invariant and Relax-ed requirements. For Relax-ed requirements, all the uncertainty factors are identified. Then using the correlation in Figure 6.5 [4] b/w Relax and SysML/Kaos, we model the bCMS Relax-ed requirements in the form of NFGs with SysML/Kaos editor.

The bCMS is a distributed crash management system that is responsible for coordinating the communication between an FSC and a PSC to handle a crisis in a timely manner. Information regarding the crisis as it pertains to the tasks of the coordinators is updated and maintained during and after the crisis. There are two collaborative sub-systems. Thus, the global coordination is the result of the parallel composition of the (software) coordination processes controlled by the two (human) distributed coordinators (i.e., PSC and FSC). Figure 7.19 shows the overall view of the bCMS case study.

7.3.2 High Level Goal Model

Figure 7.20 shows the uncertainty factors associated with the *Availability* (*The crisis details and route plan of the fire station and the police station shall be available with the exception of AS CLOSE AS POSSIBLE To 30 minutes for every 48 hours when no crisis is active.*) Relax-ed requirement. This requirement is then mapped to SysML/Kaos goal model concepts and modeled with SysML/Kaos editor.

For modeling the bCMS case study, only NFGs are considered. FGs are considered to show the impact of a *Contribution Goal* on an FG which in turn helps to show the impact of an NFG on an FG. The goal at the highest level is identified as *Security[bCMS System]*,

Uncertainty Factors	Details
ENV	The crisis details and route plan of the fire station shall be available, the crisis details and route plan of the police station shall be available
MON	Fire Station Coordinator, Police Station Coordinator, Communication Compromiser
REL	Fire Station Coordinator updates the crisis details and route plan of the fire station, Police Station Coordinator updates the crisis details and route plan of the police station, The Communication Compromiser compromises the availability of data

Figure 7.20: Availability RELAX-ed Requirement Uncertainty Factors

which is an abstract NFG and is AND-refined into two sub goals using refinement by type: *Integrity[bCMS System]* and *Availability[bCMS System]*. The goal *Availability[bCMS System]* is an abstract NFG and is AND-refined into three sub goals using refinement by type: *The crisis details and route plan of the fire station and the police station shall be available with the exception of a total of 30 minutes for every 48 hours when no crisis is active[bCMS System]*, *The crisis details and route plan of the fire station and the police station shall be available with the exception of a total of 5 minutes during the time period when at least one crisis is active[bCMS System]* and *The information related to the identification of the coordinators shall be available with the exception of a total of 5 minutes during the time period when at least one crisis is active[bCMS System]*. The goal *The crisis details and route plan of the fire station and the police station shall be available with the exception of a total of 30 minutes for every 48 hours when no crisis is active[bCMS System]* is an abstract NFG and is AND-refined into two elementary goals using refinement by type: *The crisis details and route plan of the fire station shall be available[bCMS System]* and *The crisis details and route plan of the police station shall be available[bCMS System]*. The goal *The crisis details and route plan of the fire station shall be available[bCMS System]* is satisfied by the *ContributionGoal Fire Station Coordinator* having a direct and positive contribution on it and the goal *The crisis details and route plan of the police station shall be available[bCMS System]* is satisfied by the *Contribution Goal Police Station Coordinator* which also has a direct and positive contribution on its accomplishment. The *Contribution Goal Communication Compromiser* contributes directly and negatively towards the satisfaction of the two elementary goals whose objective is to disrupt the response to the crisis for some personal gain. Figure 7.21 shows the high level goal model of the bCMS case study.

The SysML/Kaos approach helps in modeling the impact of an NFG on an FG. As shown in the SysML/Kaos meta-model of Figure 5.5, the concept of *Impact* helps in linking an NFG with an FG. It is clear from the meta-model that the *Contribution Goal* has an impact on an FG. An FG express a requirement that the future system will exhibit and an NFG is the quality to be achieved in the future system which is satisfied by a *Contribution Goal*. The concept of *Impact* is represented as an *Association Class* between a *Contribution Goal* and an FG which in turn helps to show the impact of an NFG on an FG. In Figure 7.21, the abstract FG *An FSC maintains control over a crisis situation by communicating with the PSC as well as firemen* is refined into three sub goals: *To get resources to the crisis location*, *To handle crisis related information* and

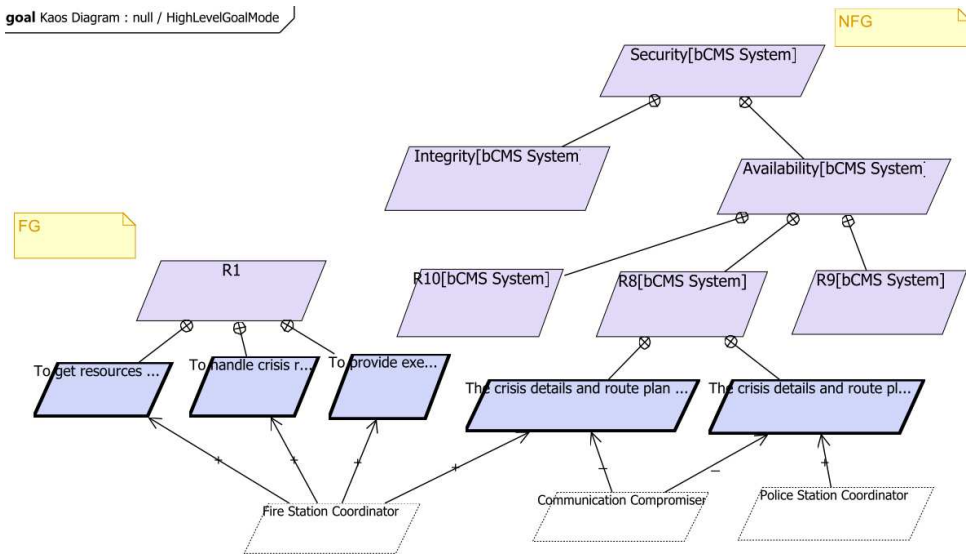

Figure 7.21: High Level Goal Model

Uncertainty Factors	Details
ENV	Integrity of the communication between coordinators, Authenticity of the coordinators to avoid the communication compromiser
MON	Secure communication channel, use PIN code, use Additional information, Communication Compromiser
REL	Secure communication channel ensures the integrity of the communication between coordinators, PIN code and Additional information ensures that the authenticity of the coordinators is in place, The communication compromiser compromises the integrity of coordinators

Figure 7.22: Integrity RELAX-ed Requirement Uncertainty Factors

To provide executable instructions to staff. The *Contribution Goal Fire Station Coordinator* has a direct and positive impact on each of the three functional sub goals.

7.3.3 Low Level Goal Model

Figure 7.22 shows the uncertainty factors associated with the *Integrity* (*The system shall ensure that the integrity of the communication between coordinators regarding crisis location, vehicle number, and vehicle location is preserved AS CLOSE AS POSSIBLE TO 99.99% of the time.*) Relax-ed requirement.

To continue further with modeling, another goal *Ensure the integrity of communications b/w coordinators[bCMS]* is identified which is an abstract NFG and is AND-refined into two sub goals using refinement by type: *Integrity of communication b/w coordinators[bCMS]* and *Authenticity of coordinators[bCMS]*. The goal *Integrity of communication b/w coordinators[bCMS]* is satisfied by the *Contribution Goal Secure communication channel*. Considering the goal *Authen-*

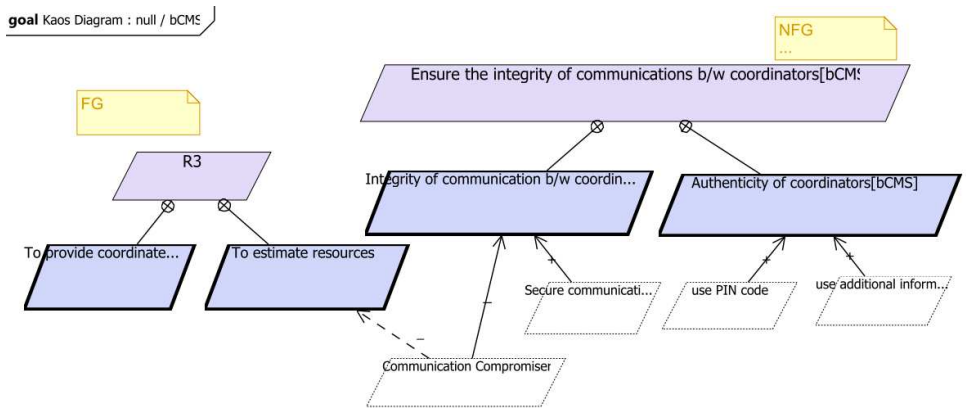

Figure 7.23: Integrity Goal Model

ticity of coordinators[bCMS], one possible way to achieve this goal is to use *PIN code*, another solution is to use *additional information*. The *Contribution Goal Communication Compromiser* has a direct and negative impact on the goal *Integrity of communication b/w coordinators[bCMS]*. Figure 7.23 shows the low level goal model of the bCMS case study.

7.3.4 Discussion

The modeling of the requirements of the bCMS case study has helped us in acquiring more information about the system. The *Contribution Goal Communication Compromiser* has a negative and direct impact on the elementary goal *Integrity of communication b/w coordinators[bCMS]*. If we have modeled the requirements of the bCMS case study using a GORE technique which does not take into account the concept of negative contribution, we would have missed important information. Also the *Contribution Goal Communication Compromiser* has a negative and indirect impact on the FG *To estimate resources*. Thanks to the use of SysML/Kaos in combination with Relax, we are able to obtain a detailed description of the requirements of the system and its environment.

In the following section, I give an assessment of our proposed approach. The assessment is explained with the help of a table which shows the problems that we identified in existing methods of requirements modeling and properties verification of SAS and the solutions that we provide in our proposed approach.

7.4 Assessment

We propose an integrated approach for requirements modeling and properties verification of SAS. The proposed approach takes requirements as input and results in its modeling and then verification using different processes and tools. It helps in bridging the gap between the requirements phase and the initial formal specification phase. In this section, I discuss different problems that we identified and addressed in our proposed approach.

Identified Problems	Solution in our Proposed Approach	Achieved
1. No difference between adaptable and invariant requirements of Self Adaptive Systems	The integration of RELAX in our proposed approach	Yes
2. No Correspondence between RELAX-ed Requirements and SysML/Kaos goal concepts	We provide correlation between RELAX-ed requirements and SysML/Kaos goal concepts to take benefit from GORE techniques while modeling requirements	Yes
3. RELAX-ed Requirements in SysML/Kaos	To use SysML/Kaos for Self Adaptive Systems, we need to apply RELAX process on its Functional Requirement and Non Functional Requirements to get RELAX-ed Requirements	Yes
4. State space explosion problem in model checking techniques for the properties verification of Self Adaptive Systems	Select those properties for verification which are important for Self Adaptive Systems	Yes
5. Case studies to validate the proposed approach	We have applied our proposed approach on two different case studies	Yes
6. Lack of integrated environment for the whole process	An integrated model based environment for early analysis of requirements and its verification	Yes

Figure 7.24: Assessement Table

We follow the **V** Software development life cycle, where we start from the requirements. We do early analysis of requirements and based on it we define the high level system design. For this, we take requirements as input and divide them into FRs and NFRs. These requirements are then passed through the Relax process which results in Relax-ed and invariant requirements. Relax-ed requirements are concerned with the adaptive features of SAS. It is at this point in time that we introduce the Relax process in our approach. We are interested in modeling those requirements of SAS which are adaptable i.e. Relax-ed. The resulting Relax-ed requirements are then automated using an editor that we developed called Relax COOL editor. This editor takes into account the uncertainty factors associated with each Relax-ed requirement. We then provide a mechanism to map Relax-ed requirement with goal oriented concepts, to take benefit from it while modeling the requirements of SAS. We use SysML/Kaos for this mapping which uses a correlation table for the correspondence between Relax-ed requirements and SysML/Kaos concepts. In the conventional process of requirements modeling using SysML/Kaos, it takes as input NFRs. We aim to inject those requirements that are Relax-able in place of NFRs to take into account the adaptability features of SAS which justifies the inclusion of the Relax process in our approach. The Relax process takes input as FRs and NFRs and divides into it Relax-ed and invariant requirements. The mapping between Relax and SysML/Kaos concepts results in the transformation of Relax uncertainty factors into SysML/Kaos goal concepts. For this purpose, we have developed a tool called Relax2SysML/Kaos editor, which is based on ATL transformations.

For the formal specification phase, we use OMEGA2/IFx for the properties verification

	Pros & Cons of our Approach	Reasons
+	Proof of concepts	The proposed approach is validated on two case studies
+	Tools	Tools were developed during the course of this thesis
+	Usability of our Approach	Easily usable and understandable as our proposition is based on concepts already present in Requirements Engineering
-	Lack of Empirical Studies	As this thesis was not part of an industrial project so we did not have the occasion to do some true empirical studies
-	No demonstration of ROI	We provide no information regarding the Return On Investment by using our proposed approach

Figure 7.25: Pros and Cons of our Proposed Approach

and model simulation of SAS. The conventional process of OMEGA2/IFx profile and toolset takes the FRs and NFRs, without differentiating those requirements that are adaptable and those that are invariant, which in the case of SAS is very important. This exposes us to the problems linked with state space explosion which is inherent in model checking techniques used by the IFx toolset for properties verification. As the number of state variables in the system increases, the size of the system state space grows exponentially. We provide a way to tackle this problem in our proposed approach. We take into account those requirements that are important for SAS e.g. RELAX-ed or invariant requirements to avoid the state explosion problem associated with model checking techniques.

For the validation of our proposed approach, we use two case studies i.e. AAL and bCMS. We model the RELAX-ed requirements of both case studies. For the properties verification part, we verify three properties of the AAL case study. The overall proposed approach need an integrated tooling environment for taking into account the different transformations and automations identified. We provide an integrated model based environment for early analysis and verification of requirements. Figure 7.24 shows a table with the problems that we addressed in our proposed approach.

Our proposed approach is usable and understandable as we base our proposition on using concepts already present in RE. The cons of our proposed approach is that we do not provide any empirical studies for the validation, as this research work is not conducted in an industrial setup. Apart from the two case studies, we did not applied our proposed approach on any such concrete case studies that can provide any empirical data e.g. what will we gain in terms of time and budget, which is one of the main criteria for the evaluation of any new proposed approach. We are of the view that an ideal situation will be to develop an SAS using our proposed approach that is already developed with another approach. This will provide interesting results and will show to what extent our proposed approach is better than the

other approaches. Consequently, we are not able to calculate the Return On Investment (ROI) by using our approach. Figure 7.25 shows a table with the pros and cons of our proposed approach.

7.5 Conclusion

We provided a model based integrated approach for the requirements modeling and properties verification of SAS. The context of our work resides in self adaptation and we are at the very initial stage of the SAS development life cycle which means that we do not provide any mechanism for self adaptation which is beyond the scope of this thesis.

The proposed combination of Relax and SysML/Kaos approach models the NFRs of SAS. Both these approaches treat NFRs at the high level of abstraction. The table in Figure 6.5 is used which shows the correlation between different concepts treated by each approach. First of all, Relax-ed requirements with uncertainty factors are identified in the case studies and based on the correlation table, they are mapped to SysML/Kaos concepts. SysML/Kaos is then used to model these Relax-ed requirements. The NFRs of the bCMS case study are also modeled using the proposed approach.

In our proposed approach, we provided a mechanism to verify the requirements of SAS. We start from modeling the structural and behavioral parts of AAL system. OMEGA2 profile is used for specification and verification of dynamic properties of models through *observers*. For the verification and simulation part, IFx is used which is a toolset for the simulation of OMEGA2 models and the verification of properties defined on these models. Three properties of the AAL system are verified using the IFx toolset, these properties are obtained after the Relax process is applied on its traditional properties. In one case, the verification results in errors which can then be simulated through the interactive simulation interface of the IFx toolset in order to identify the source of the error and then subsequently correct it in the model. In other case, after correcting the error, the verification results in the fulfillment of all the three properties.

CHAPTER 8

Conclusion & Perspectives

Contents

The context of this research work is situated in the field of SE for SAS. This work resides in the very early stages of the software development life cycle i.e. at the RE phase. The overall contribution of this thesis is to propose an integrated approach for modeling and verifying the requirements of SAS using MDE techniques.

Our proposed approach takes requirements as input and then by applying various processes and tools, we integrate the notion of adaptability in requirements which we model using GORE techniques. Once we have the system design, we then introduce a mechanism for the properties verification of SAS.

8.1 Problem Recall

Because of the high adaptive nature of SAS, they modify their behavior at run-time in response to changing environmental conditions. For these systems, NFRs play an important role, and one has to identify as early as possible those requirements that are adaptable. The distributed nature of SAS and changing environmental factors (including human interaction) make it difficult to anticipate all the explicit states in which the system will be during its lifetime. As such, an SAS needs to be able to tolerate a range of environmental conditions and contexts, but the exact nature of these contexts remains imperfectly understood. A feature common to all the previous works regarding RE for SAS is that they assume that all adaptation choices are known and enumerated at design time and does not take unanticipated adaptations into account.

We have identified some problems associated with requirements modeling and verification using existing approaches. On one hand, existing requirements modeling approaches do not take into account the uncertainty factors associated with SAS, on the other, hand we are exposed to the state space explosion problems associated with existing properties verification techniques of these systems.

8.2 Our Contribution

We treat NFRs from the very beginning i.e. at the same level of abstraction as FRs. RE methods for SAS must consider the inherent uncertainty of these systems. This results in the conception of new RE languages. Keeping in mind this uncertainty, we provide an integrated approach for dealing with requirements modeling and properties verification of SAS. For this we put forward our proposed approach which aim to solve different problems that we identified regarding requirements modeling and properties verification. We are interested in using MDE techniques for the RE and properties verification of SAS.

We identified the gap that exists between the requirements and the overall system model. SysML provides the notion of requirements diagram, which provide the relationship between requirements and also the relationship of requirements with other model elements thus allowing the definition of traceability links between requirements and other model elements. We start from the very beginning of the system development life cycle i.e. from requirements.

These requirements are then divided into NFRs and FRs. We used RELAX which is an RE language for SAS and which can introduce flexibility in NFRs to adapt to any changing environmental conditions. The essence of RELAX for SAS is that it provides a way to relax certain requirements for other requirements in situations where the resources are constrained or priority must be given to requirements. To bridge this gap, we provide a DSL for RELAX, which takes requirements in textual format as input and transform it into SYSML requirements diagram.

RELAX provides new vocabulary and terminology for the RE of SAS. It provides the notion of uncertainty factors which are associated with requirements. The RELAX process takes input as FRs and NFRs and results in the distinction of those requirements that are adaptable called RELAX-ed requirements and those that are fixed called invariant requirements. To take benefit of RELAX-ed and invariant requirements, we integrate it in our proposed approach. For this purpose we have developed a tool called RELAX COOL editor which is used to automate the formalization of SAS requirements by taking into account the different uncertainty factors associated with each RELAX-ed requirement. This tool is developed using Xtext which is a framework for the development of DSLs and other textual programming languages and helps in the development of an IDE for the DSL.

Because of the inherent uncertainty in SAS, goal based approaches can help in developing the requirements of these systems. We use SYSML/KAOS which is an extension of the SYSML requirements model with concepts of the KAOS goal model. SYSML/KAOS provides an editor for modeling the NFRs in the form of NFGs which is much more rich and complete in defining relations between requirements (refinement relations, conflict identification and resolution, positive/negative and direct/indirect impacts). Here, invariant requirements are captured by the concept of FGs whereas RELAX-ed requirements are captured by the concept of NFGs. SYSML/KAOS takes as input a set of NFRs without differentiating between those requirements that are adaptable and those that are not which is an important aspect of SAS. We have merged SYSML/KAOS and RELAX so that we obtain a detailed and strong requirements description of the system and its context and to take into account the adaptability features associated with SAS.

In our proposed approach, we have provided a correlation table that helps in mapping the RELAX and SYSML/KAOS concepts. Using this table, the RELAX-ed requirements are then transformed to SYSML/KAOS goal concepts. This mapping is done using ATL, which is a model transformation technique and which takes as input a source model and transforms it into a target model. For this purpose, we have developed a tool called RELAX2SYSML/KAOS editor. This editor is capable of modeling the RELAX-ed requirements in the form SYSML/KAOS goal concepts.

In order to validate our proposed approach, we modeled the requirements of two case studies: an AAL and a bCMS case study using the same correlation table that shows the mapping between RELAX and SYSML/KAOS. The RELAX2SYSML/KAOS editor is then used to model the requirements of the two case studies.

In our proposed approach, we provide a mechanism to verify some properties of the

	Processes/Tools Used	Results
1	RELAX Process	Divide requirements into RELAX-ed and Invariant requirements
2	RELAX COOL Editor	To automate the RELAX-ed Requirements using our RELAX COOL editor
3	Transformation towards Goal Concepts	Using the correlation b/w RELAX-ed Requirements and SysML/Kaos goal concepts
4	RELAX2SysMLKaos Editor	RELAX2SysMLKaos editor
5	Modeling Observers	To model the observers
6	OMEGA2IFx	Integration of OMEGA2/IFx in the proposed approach

Figure 8.1: Overall Conclusion

SAS even before the actual development of these systems. We used OMEGA2/IFx for the properties verification and model simulation of these systems. OMEGA2 is an executable Uml/SysML profile dedicated to the formal specification and validation of critical real-time systems. It is based on a subset of Uml 2.2/SysML 1.1 containing the main constructs for defining the system structure and behavior. We used IFx toolset for OMEGA2 models simulation and properties verification. The system design that resulted from SysML/Kaos can be injected into OMEGA2/IFx for the properties verification. The problem with the conventional method of properties verification using OMEGA2/IFx is that it takes FRs and NFRs as input and then verify it. In the context of SAS, we need to inject those properties into OMEGA2/IFx for verification that are either adaptable or invariant but not all the properties. In this way, we can overcome problems like state space explosion by verifying only those properties that are of interest for the SAS. This step is accompanied by *observers* which are the properties that we want to verify on the system. While verifying the properties, the compiler produced errors, which we simulated by using the interactive simulation interface of the IFx toolset to trace back the error in the model and then corrected it.

To sum up, we provide a means to bridge the gap between the requirements phase and the initial validation phase. The proposed approach takes input as requirements and then using different processes and tools that either existed or we developed, we model those NFRs that can be Relax-ed. This helps in capturing the adaptability features associated with SAS. We then verify some of the properties of these systems. As we are in the very early stages of the system development life cycle so the aspects related to adaptation mechanisms and strategies for adaptation are beyond the scope of this work. Figure 8.1 shows a table with the processes and tools that we used in our proposed approach and the results achieved.

8.3 Perspectives

In the following, I mention some of the perspectives that we identified regarding requirements modeling and properties verification of SAS.

8.3.1 Perspective for Requirements Modeling of Self Adaptive Systems

To take benefits of GORE techniques while modeling the requirements of SAS, we provided a correlation table between RELAX and SYSML/KAOS. Based on this table we have mapped the RELAX-ed requirements uncertainty factors to SYSML/KAOS goal concepts. ATL is used for the model transformation between the two approaches. For the time being, our RELAX2SYSML/KAOS tool is capable of mapping the RELAX concepts to SYSML/KAOS concepts but not the inverse. We have identified the inverse mapping between the two concepts as a perspective so that those people who are familiar with SYSML/KAOS can map goal concepts to RELAX concepts to which they are unfamiliar, so that to provide an additional knowledge base regarding requirements modeling of SAS. This will help in identifying the uncertainty factors associated with each requirement. The resulting RELAX-ed requirements can be subjected to various tools and processes and can be used with various other methods for answering questions regarding self adaptivity like which conditions to monitor, which decision-making procedures and possible adaptations strategies to follow.

8.3.2 Perspectives for Properties Verification of Self Adaptive Systems

The verification of RELAX-ed requirements in our proposed approach is done using OMEGA2/IFx. We can do the validation of these requirements at execution time. But the wealth of information associated with run-time phenomena is particularly a challenging problem. A promising approach to managing complexity in runtime environments is to develop adaptation mechanisms that leverage software models, referred to as Models@run.time [15]. Research on Models@run.time seeks to extend the applicability of models and abstractions to the runtime environment, with the goal of providing effective technologies for managing the complexity of evolving software behavior while it is executing [8]. Runtime adaptation mechanisms that leverage software models extend the applicability of MDE techniques to the runtime environment. Contemporary mission-critical software systems are often expected to safely adapt to changes in their execution environment. Given the critical roles SAS play, it is often inconvenient to take them offline to adapt their functionality. Consequently, these systems are required, when feasible, to adapt their behavior at run-time with little or no human intervention [15]. Different seminars [1] and workshops [2] are now dedicated to the mix of MDE and SAS.

We did not treated these aspects as we were at the very early stages of the SAS development life cycle. To be precise, our work resides in the framework of self adaptation but we do

1. http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=11481
2. http://st.inf.tu-dresden.de/MRT12/

not treat the development of self adaptation mechanisms. We help the SAS developers by providing a mechanism for identifying the adaptive features associated with the requirements of these systems. The use of Models@run.time in our proposed approach is a perspective that we have identified.

In our proposed approach, for the properties verification using OMEGA2/IFx, we model the *observers* and then we check these observers against the system design to see if the properties are verified or not. Right now, we model these *observers* as an SMD. We would like to automate this process of *observers* modeling by automatically generating it from Relax-ed and invariant requirements. We can then provide the resulting system design from the application of our proposed approach to be used with these automatically generated *observers*, to verify the properties against the system design using OMEGA2/IFx.

Another perspective that we identified is based on using formal method techniques like B. The use of model checking techniques used by OMEGA2/IFx exposes us to the problem of state explosion problem which is inherent in these techniques. We handle this problem in our proposed approach by only injecting Relax-ed or invariant requirements i.e. those requirements that are of interest for SAS. But we can counter this problem using formal methods like B. There are already some works done for the mapping between SysML/Kaos and B in this regard. A method is defined for bridging the gap between the requirements analysis level (Extended SysML) and the formal specification level (B) [53]. This method derives the architecture of B specifications from SysML goal hierarchies. We believe that using proof based formal methods like B can help in overcoming the state space explosion problem associated with model checking techniques.

8.3.3 Perspective for the Empirical Studies

As a proof of concept, we applied our proposed approach on two case studies in the domain of SAS. The requirements of both case studies were modeled and the properties of one of the case study were verified using our proposed approach. An ideal situation will be to apply it on a real world system, which will help us to assess the validity of our proposed approach against some predefined evaluation criteria. For this, the first step would be to define the evaluation criteria. Then we need to provide enough experience to measure the defined criteria which will provide statistical data about its applicability on one hand and the improvements that can be brought about on the other hand.

The evaluation criteria can be defined, for example in terms of the usability of our proposed approach, correctness of the transformation rules between Relax and SysML/Kaos concepts. We can then model and verify the NFRs of the real world SAS using our approach and compare it with the conventional methods of requirements modeling and verification. This will provide a true assessment of our proposed approach.

Author's Bibliography

International Conferences

1. **Manzoor Ahmad**, Iulia Dragomir, Jean M. Bruel, Iulian Ober and Nicolas Belloir. Early Analysis of Ambient Systems SysML Properties using OMEGA2-IFx, SIMULTECH 29-31 July 2013, Reykhavík Iceland.
2. **Manzoor Ahmad**, Jean M. Bruel, Régine Laleau and Christophe Gnaho. Using Relax, SysML and Kaos for Ambient Systems Requirements Modeling. Procedia Computer Science Volume 10, 2012, Pages 474–481, The 3rd International Conference on Ambient Systems, Networks and Technologies August 27-29, 2012, Niagara Falls, Ontario, Canada.
3. **Manzoor Ahmad**. First Step towards a Domain Specific Language for self-Adaptive Systems. IEEE Proceedings, Tunisia Chapter, DANCE 30 May 2010, Tozeur Tunisia.

National Conferences

1. **Manzoor Ahmad**, Jean M. Bruel, Régine Laleau et Christophe Gnaho. Modélisation des Exigences pour les Systèmes Auto-adaptatifs: Intégration des Techniques Relax/SysML/Kaos. Journées GDR - GPL - CIEL 19 et 20 juin 2012.
2. **Manzoor Ahmad** , Jean M. Bruel, Requirement Language Tooling with Xtext, UBIMOB 2011 : 7es Journées francophones Mobilité et Ubiquité Toulouse, Museum d'Histoire Naturelle 6 juin 2011 - 8 juin 2011.

Workshops

1. Jean M. Bruel, Nicolas Belloir, **Manzoor Ahmad**. SPAS: Un Profil SysML pour les Systèmes Auto-Adaptatifs. Dans: 15ème Colloque National de la Recherche en IUT (CNRIUT), Lille, 08/06/09-10/06/09.
2. **Manzoor Ahmad**, Jean M. Bruel, Antoine Beugnard. From Composition to Connectors. In: 4th International School on Model Driven Development for Distributed Real-time Embedded Systems, Aussois, France, 20/04/09-24/04/09.

Bibliography

[1] Jean R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.

[2] Manzoor Ahmad. First Step Towards a Domain Specific Language for Self-Adaptive Systems. In *10th Annual International Conference on New Technologies of Distributed Systems (NOTERE'10)*, pages 285–290. IEEE, 2010.

[3] Manzoor Ahmad, João Araújo, Nicolas Belloir, Régine Laleau Jean M. Bruel, Christophe Gnaho, and Farrida Semmak. Self-Adaptive Systems Requirements Modelling: four Related Approaches Comparison. In *Comparing *Requirements* Modeling Approaches Workshop (CMA@RE), RE 2013*, Rio de Janeiro Brazil, 2013. IEEE Computer Society Press.

[4] Manzoor Ahmad, Jean M. Bruel, Régine Laleau, and Christophe Gnaho. Using RELAX, SysML and KAOS for Ambient Systems Requirements Modeling. In *The 3rd International Conference on Ambient Systems, Networks and Technologies (ANT '12)*. Elsevier Procedia Computer Science, Volume 10, Pages 474 - 481, 2012.

[5] Manzoor Ahmad, Iulia Dragomir, Jean M. Bruel, Iulian Ober, and Nicolas Belloir. Early Analysis of Ambient Systems SysML Properties using OMEGA2-IFx. In *3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH'13)*, 2013.

[6] Annie I. Anton. Goal-Based Requirements Analysis. In *Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*. IEEE Computer Society, 1996.

[7] L. Apvrille, J. P. Courtiat, C. Lohr, and P. de Saqui-Sannes. TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit. *IEEE Trans. Softw. Eng.*, 30(7), 2004.

[8] Uwe Aßmann, Nelly Bencomo, Betty H. C. Cheng, and Robert B. France. Models@Run.Time (Dagstuhl Seminar 11481). *Dagstuhl Reports*, 1(11), 2011.

[9] Colin Atkinson and Thomas Kühne. Model-Driven Development: A Metamodeling Foundation. *IEEE Softw.*, 20(5), 2003.

[10] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[11] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. Fuzzy Goals for Requirements-Driven Adaptation. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 125–134, Washington, DC, USA, 2010. IEEE Computer Society.

[12] Jérémy Bascans, Jérémy Walczak, Jérôme Zeghoudi, Manzoor Ahmad, Jacob Geisel, and Jean M. Bruel. COOL RELAX Editor, M2ICE Project, Université de Toulouse le Mirail, 2013.

[13] Benghazi, Kawtar and Visitación Hurtado, María and Rodríguez, María Luisa and Noguera, Manuel. Applying Formal Verification Techniques to Ambient Assisted Living Systems. In *OnTheMove Workshop (OTM '09)*. Springer-Verlag Berlin Heidelberg 2009, 2009.

[14] Daniel M. Berry, Betty H. C. Cheng, and Ji Zhang. The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. In *In 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ*, 2005.

[15] Gordon S. Blair, Nelly Bencomo, and Robert B. France. Models@ Run.Time. *Computer*, 42(10):22–27, 2009.

[16] Sebastien Bornot and Joseph Sifakis. An Algebraic Framework for Urgency. In *Information and Computation (IC '00)*. Elsevier, 2000.

[17] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. The IF Toolset. In *Formal Methods for the Design of Real-Time Systems (FMDRTS '04)*. Springer Berlin/Heidelberg, 2004.

[18] Jean Bézivin. On the Unification Power of Models. *Software Systems Modeling*, 4(2), 2005.

[19] Jean Bézivin and Olivier Gerbé. Towards a Precise Definition of the OMG/MDA Framework. In *Proceedings. 16th Annual International Conference on Automated Software Engineering, 2001. (ASE 2001).*, 2001.

[20] Betty H. C. Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software Engineering for Self-Adaptive Systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[22] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer, 1st edition, 1999.

[23] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, London, 1999.

[24] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model Checking and the State Explosion Problem. In Bertrand Meyer and Martin Nordio, editors, *Tools for Practical Software Verification*, volume 7682 of *Lecture Notes in Computer Science*, pages 1–30. Springer Berlin Heidelberg, 2012.

[25] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. Automated Classification of Non-Functional Requirements. *Requir. Eng.*, 12(2):103–120, May 2007.

[26] J. P. Courtiat, C. A. S. Santos, C. Lohr, and B. Outtaj. Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique. *Comput. Commun.*, 23(12), 2000.

[27] Anne Dardenne, Axel V. Lamsweerde, and Stephen Fickas. Goal-directed Requirements Acquisition. In *SCIENCE OF COMPUTER PROGRAMMING*, 1993.

[28] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Goeschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Marin Litoiu, Antonia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Bradley Schmerl, Dennis B. Smith, João P. Sousa, Gabriel Tamura, Ladan Tahvildari, Norha M. Villegas, Thomas Vogel, Danny Weyns, Kenny Wong, and Jochen Wuttke. Software engineering for self-adaptive systems: A second research roadmap.

[29] Ana Dias, Vasco Amaral, and João Araújo. Towards a Domain Specific Language for a Goal-Oriented approach based on KAOS. In *RCIS*, pages 409–420, 2009.

[30] Iulia Dragomir, Iulian Ober, and David Lesens. A Case Study in Formal System Engineering with SysML. In *17th International Conference on Engineering of Complex Computer Systems (ICECCS '12)*. IEEE, 2012.

[31] Marvin M Early. Relating Software Requirements to Software Design. *SIGSOFT Softw. Eng. Notes*, 11(3):37–39, July 1986.

[32] Martin Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 1 edition, 2010.

[33] Robert France and Bernhard Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[34] Debanjan Ghosh, Raj Sharman, H. Raghav Rao, and Shambhu Upadhyaya. Self-healing Systems - Survey and Synthesis. *Decis. Support Syst.*, 42(4):2164–2185, January 2007.

[35] Martin Glinz. On Non-Functional Requirements. In *15th IEEE International Requirements Engineering Conference, RE '07*. IEEE, 2007.

[36] Christophe Gnaho and Farida Semmak. Une Extension SysML pour l'ingénierie des Exigences Non-Fonctionnelles Orientée But. In *Ingénierie des Systèmes d'Information*. Lavoisier Paris FRANCE, 2010.

[37] HeatherJ. Goldsby, Pete Sawyer, Nelly Bencomo, Betty H.C. Cheng, and Danny Hughes. Goal-Based Modeling of Dynamically Adaptive System Requirements. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, ECBS '08, Washington, DC, USA, 2008. IEEE Computer Society.

[38] Jeff Gray, Kathleen Fisher, Charles Consel, Gabor Karsai, Marjan Mernik, and Juha-Pekka Tolvanen. DSLs: The Good, The Bad, and The Ugly. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA Companion '08, pages 791–794, New York, NY, USA, 2008. ACM.

[39] The Standish Group. Chaos report 1995. Technical report, The Standish Group, 1995.

[40] The Standish Group. Chaos report 2009. Technical report, The Standish Group, 2009.

[41] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., 2003.

[42] Seong ick Moon, K.H. Lee, and Doheon Lee. Fuzzy branching temporal logic. In *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2004.

[43] IEEE. *IEEE Standard Glossary of Software Engineering Terminology 610.12-1990*. IEEE Press, 1999.

[44] Eclipse Indigo. ATL Guide.

[45] Michael Jackson. *Problem frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[46] Matthias Jarke, X. Tung Bui, and John M. Carroll. Scenario Management: An Interdisciplinary Approach, 1999.

[47] Haruhiko Kaiya, Hisayuki Horai, and Motoshi Saeki. AGORA: Attributed Goal-Oriented Requirements Analysis Method. In *RE*. IEEE Computer Society, 2002.

[48] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1), 2003.

[49] Jörg Kienzle, Nicolas Guelfi, and Sadaf Mustafiz. Crisis Management Systems: A Case Study for Aspect-Oriented Modeling. In *Transactions on Aspect-Oriented Software Development VII*, volume 6210 of *Lecture Notes in Computer Science*, pages 1–22. Springer Berlin Heidelberg, 2010.

[50] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, 1st edition edition, 2003.

[51] Toma Kosar, Pablo E. Martinez López, Pablo A. Barrientos, and Marjan Mernik. A preliminary Study on Various Implementation Approaches of Domain-Specific Language. *Inf. Softw. Technol.*, 50(5):390–405, 2008.

[52] Thomas Kühne. What is a Model? In Jean Bezivin and Reiko Heckel, editors, *Language Engineering for Model-Driven Software Development*, Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[53] Régine Laleau, Farida Semmak, Abderrahman Matoussi, Dorian Petit, Ahmed Hammad, and Bruno Tatibouet. A First Attempt to Combine SysML Requirements Diagrams and B. *Innovations in Systems and Software Engineering*, 6, 2010.

[54] Axel V. Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*. IEEE Computer Society, 2001.

[55] Axel V. Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 1st edition edition, 2009.

[56] Axel V. Lamsweerde and Emmanuel Letier. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering. In *Radical Innovations of Software and System Engineering, Montery'02 Workshop, Venice(Italy)*. Springer-Verlag, 2003.

[57] Robin Laney, Leonor Barroca, Michael Jackson, and Bashar Nuseibeh. Composing Requirements Using Problem Frames. In *Proceedings of the Requirements Engineering Conference, 12th IEEE International*, RE '04, pages 122–131, Washington, DC, USA, 2004. IEEE Computer Society.

[58] Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Yijun Yu. Towards Requirements-Driven Autonomic Systems Design. In *Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, DEAS '05, pages 1–7, New York, NY, USA, 2005. ACM.

[59] Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, and John Mylopoulos. Requirements-Driven Design of Autonomic Application Software. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, CASCON '06, Riverton, NJ, USA, 2006. IBM Corp.

[60] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[61] Tomaž Lukman and Marjan Mernik. Model-Driven Engineering and its Introduction with Metamodeling Tools. In *9th International PhD Workshop on Systems and Control: Young Generation Viewpoint Izola, Slovenia*, 2008.

[62] Robyn R. Lutz. Targeting Safety-Related Errors during Software Requirements Analysis. *J. Syst. Softw.*, 34(3), 1996.

[63] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electron. Notes Theor. Comput. Sci.*, 152, 2006.

[64] Ahmed Yakout A. Mohamed, Abdel Fatah A. Hegazy, and Ahmed R. Dawood. Aspect Oriented Requirements Engineering. *Computer and Information Science*, 3(4):135–154, 2010.

[65] Rui Monteiro, João Araújo, Vasco Amaral, Miguel Goulão, and Pedro Miguel Beja Patrício. Model-Driven Development for Requirements Engineering: The Case of Goal-Oriented Approaches. In Ricardo Machado João Pascoal Faria, Alberto Silva, editor, *8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012)*, number 8 in Quality of Information and Communications Technology, pages 75–84. IEEE Computer Society, 09 2012.

[66] Pierre A. Muller, Frédéric Fondement, and Benoit Baudry. Modeling Modeling. In Andy Schürr and Bran Selic, editors, *Model Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009.

[67] Peter Naur and Brian Randell. Software Engineering Report, 1968.

[68] Jürgen Nehmer, Martin Becker, Arthur Karshmer, and Rosemarie Lamm. Living Assistance Systems: an Ambient Intelligence Approach. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*. ACM, 2006.

[69] Carlos Nunes, João Araújo, Vasco Amaral, and Carla T. L. L. Silva. A Domain Specific Language for the i* Framework. In *ICEIS (1)*, pages 158–163, 2009.

[70] Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. In *In Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*. ACM Press, 2000.

[71] Iulian Ober and Iulia Dragomir. OMEGA2: A New Version of the Profile and the Tools. In *15th International Conference on Engineering of Complex Computer Systems (ICECCS '10)*. IEEE, 2010.

[72] Iulian Ober and Iulia Dragomir. Unambiguous UML Composite Structures: The OMEGA2 Experience. In *Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543, pages 418–430. Springer, 2011.

[73] Iulian Ober, Susanne Graf, and Ileana Ober. Validating Timed UML Models by Simulation and Verification. In *International Journal on Software Tools for Technology Transfer (STTT '06)*. Springer, Volume 8, Issue 2, pp 128-145, 2006.

[74] Liliana Pasquale. *A Goal-Oriented Methodology for Self-Supervised Service Compositions*. PhD thesis, Politecnico di Milano, Milan, Italy, 2011.

[75] Andres J. Ramirez and Betty H. C. Cheng. Automatic Derivation of Utility Functions for Monitoring Software Requirements. In *Proceedings of the 14th international conference on Model driven engineering languages and systems*, MODELS'11, pages 501–516, Berlin, Heidelberg, 2011. Springer-Verlag.

[76] Andres J. Ramirez, Betty H. C. Cheng, Nelly Bencomo, and Pete Sawyer. Relaxing Claims: Coping with Uncertainty While Evaluating Assumptions at Run Time. In RobertB. France, Jürgen Kazmeier, Ruth Breu, and Colin Atkinson, editors, *Model Driven Engineering Languages and Systems*, volume 7590 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012.

[77] Andres J. Ramirez, Erik M. Fredericks, Adam C. Jensen, and Betty H. C. Cheng. Automatically RELAXing a Goal Model to Cope with Uncertainty. In *Proceedings of the 4th international conference on Search Based Software Engineering*, SSBSE'12, pages 198–212, Berlin, Heidelberg, 2012. Springer-Verlag.

[78] Awais Rashid and Ruzanna Chitchyan. Aspect-oriented Requirements Engineering: A Roadmap. In *Proceedings of the 13th international workshop on Early Aspects*. ACM, 2008.

[79] Colette Rolland, Carine Souveyet, and Camille B. Achour. Guiding goal modelling using scenarios. In *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, volume 24. IEEE, 1998.

[80] Vítor E. S. Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness Requirements for Adaptive Systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self- Managing Systems*, SEAMS '11, pages 60–69, New York, NY, USA, 2011. ACM.

[81] Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh. Specifying Monitoring and Switching Problems in Context. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 211–220, 2007.

[82] Ed Seidewitz. What Models Mean. *Software, IEEE*, 20(5), 2003.

[83] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Softw.*, 20(5):19–25, September 2003.

[84] Bran Selic and Jim Rumbaugh. *Using UML for Modeling Complex Real-Time Systems*, 1998.

[85] Eric P. Kasten Seyed M. Sadjadi, Philip K. McKinley. Architecture and Operation of an Adaptable Communication Substrate. In *Proceedings. The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems FTDCS'03*, 2003.

[86] Ian Sommerville. *Software Engineering*. Addison-Wesley, 9th edition edition, 2010.

[87] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. Wiley, 1st edition edition, 1997.

[88] Alistair Sutcliffe. Scenario-based Requirements Engineering. In *11th IEEE International Requirements Engineering Conference, 2003.* IEEE, 2003.

[89] Gerald Tesauro and Jeffrey O. Kephart. Utility Functions in Autonomic Systems. In *Proceedings of the First International Conference on Autonomic Computing*, ICAC '04, pages 70–77, Washington, DC, USA, 2004. IEEE Computer Society.

[90] Arie V. Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: an Annotated Bibliography. *SIGPLAN Not.*, 35:26–36, 2000.

[91] Verimag and Irit. *OMEGA2-IFx for UML/SysML v2.0, Profile and Toolset, User Manual Document v1.1*, 2011.

[92] Kristopher Welsh and Pete Sawyer. Requirements Tracing to Support Change in Dynamically Adaptive Systems. In *Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ '09, pages 59–73, Berlin, Heidelberg, 2009. Springer-Verlag.

[93] Kristopher Welsh and Pete Sawyer. Understanding the Scope of Uncertainty in Dynamically Adaptive Systems. In *Requirements Engineering: Foundation for Software Quality*. Springer Berlin Heidelberg, 2010.

[94] Kristopher Welsh, Pete Sawyer, and Nelly Bencomo. Towards Requirements Aware Systems: Run-Time Resolution of Design-Time Assumptions. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011.

[95] Jon Whittle, Pete Sawyer, Nelly Bencomo, and Betty H. C. Cheng. A Language for Self-Adaptive System Requirements. In *International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements, 2008. SOCCER '08.*, 2008.

[96] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean M. Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In *Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE*, RE '09, pages 79–88, Washington, DC, USA, 2009. IEEE Computer Society.

[97] Tjerk Wolterink. The Future of Software Engineering: Model Driven Engineering, 2010.

[98] Eric S. K. Yu. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*. IEEE Computer Society, 1997.

[99] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and Julio C. S. P. Leite. From Goals to High-Variability Software Design. In *Proceedings of the 17th international conference on Foundations of intelligent systems*, ISMIS'08, pages 1–16, Berlin, Heidelberg, 2008. Springer-Verlag.

[100] Yijun Yu, Julio C.S.P. Leite, and John Mylopoulos. From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In *Proceedings of the Requirements Engineering Conference, 12th IEEE International*, RE '04, pages 38–47, Washington, DC, USA, 2004. IEEE Computer Society.

[101] K Yue. What Does It Mean to Say that a Specification is Complete? In *International Workshop on Software Specification and Design (IWSSD'87)*, 1987.

[102] Lofti Zadeh. Fuzzy Sets. *Information and Control*, 8(3):338–353, 1965.

[103] Ji Zhang and Betty H. C. Cheng. Model-based Development of Dynamically Adaptive Software. In *Proceedings of the 28th international conference on Software engineering (ICSE'08)*. ACM, 2006.

Acronyms

AGORA Attributed Goal-Oriented Requirements Analysis Method

AAL Ambient Assisted Living

AHRI Aware Home Research Initiative

AMN Abstract Machine Notation

AORE Aspect Oriented Requirements Engineering

ATL Atlas Transformation Language

AwReqs Awareness Requirements

BDD Block Definition Diagram

DSL Domain Specific Languages

DSLs Domain Specific Languages

DAS Dynamic Adaptive Systems

DSL Domain Specific Language

FLAGS Fuzzy Live Adaptive Goals for Self-Adaptive Systems

bCMS barbados Car Crash Crisis Management System

FSC Fire Station Coordinator

FBTL Fuzzy Branching Temporal Logic

FRs Functional Requirements

FR Functional Requirements

FGs Functional Goals

FG Functional Goal

GBRAM Goal Based Requirements Analysis Method

GORE Goal Oriented Requirements Engineering

GUI Graphical User Interface

GMF Graphical Modeling Framework

IBD Internal Block Diagram

IDE Integrated Development Environment

KAOS Knowledge Acquistion in autOmated Specification
LoREM Levels of Requirement Engineering for Modeling
LSC Live sequence charts
LSCs Live Sequence Charts
LTL Linear Temporal logic
MAPE Monitor-Analyze-Plan-Execute
MDD Model Driven Development
MDE Model Driven Engineering
MDA Model Driven Architecture
MOF Meta Object Facility
UML Unified Modeling Language
NFGs Non Functional Goals
NFG Non Functional Goal
NFRs Non Functional Requirements
NFR Non Functional Requirement
OCL Object Constrained Language
OMG Object Management Group
PIM Platform Independent Models
PSC Police Station Coordinator
QVT Query View Transformation
RTLOTOS Real Time Language Of Temporal Ordering Specifications
REAssuRE REcording of Assumtions in RE
RE Requirements Engineering
RC Requirement Chunk
SysML System Modeling Language
SE Software Engineering
SAS Self Adaptive Systems
SMD State Machine Diagram
SD Strategic Dependency
SR Strategic Rationale
TCTL Timed Computation Tree Logic
TURTLE Timed UML and RT-LOTOS Environment
XML Extensible Markup Language
XMI XML Metadata Interchange

Thesis Author

Modeling and Verification of Functional and Non Functional Requirements of Ambient, Self Adaptive Systems

Thesis Supervisors:
Jean-Michel BRUEL — Professor, University of Toulouse, France
Nicolas BELLOIR — Maître de Conférences, UPPA Pau, France
PhD defended october 07, 2013 at IUT de Blagnac

Abstract

The overall contribution of this thesis is to propose an integrated approach for modeling and verifying the requirements of Self Adaptive Systems using Model Driven Engineering techniques. Model Driven Engineering is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem level abstractions to software implementations. By using these techniques, we have bridged this gap through the use of models that describe complex systems at multiple levels of abstraction and through automated support for transforming and analyzing these models. We take requirements as input and divide it into Functional and Non Functional Requirements. We then use a process to identify those requirements that are adaptable and those that cannot be changed. We then introduce the concepts of Goal Oriented Requirements Engineering for modeling the requirements of Self Adaptive Systems, where Non Functional Requirements are expressed in the form of goals which is much more rich and complete in defining relations between requirements. We have identified some problems in the conventional methods of requirements modeling and properties verification using existing techniques, which do not take into account the adaptability features associated with Self Adaptive Systems. Our proposed approach takes into account these adaptable requirements and we provide various tools and processes that we developed for the requirements modeling and verification of Self Adaptive Systems. We validate our proposed approach by applying it on two different case studies in the domain of Self Adaptive Systems.

Keywords: Software Engineering, Requirements Engineering, Model Driven Engineering, Self Adaptive Systems, Ambient Systems, RELAX, RELAX-ed Requirements, Non Functional Requirements Modeling, Properties Verification.

Discipline: Informatique

Institut de Recherche en Informatique de Toulouse — UMR 5505
Université de Toulouse Le Mirail, 5 Allée Antonio Machado 31100 Toulouse, France

Thesis Author

Modélisation et Vérification des Exigences Fonctionnelles et Non Fonctionnelles des Systèmes Ambiants, Auto-Adaptatifs

Directeurs de thèse :
Jean-Michel BRUEL — Professeur, Université de Toulouse, France
Nicolas BELLOIR — Maître de Conférences, UPPA Pau, France
Thèse soutenue le 07 octobre, 2013 à l'IUT de Blagnac

Résumé

Le contexte de ce travail de recherche se situe dans le domaine de Génie Logiciel, et plus spécifiquement vise les systèmes auto-adaptatifs. Le travail de recherche vise les tous premiers stades du cycle de vie du développement logiciel : la phase de spécification des exigences. Nous nous concentrons sur la définition et la modélisation des exigences ainsi que de leur vérification. La contribution globale de cette thèse est de proposer une approche intégrée pour la modélisation et la vérification des exigences des SAS à l'aide de techniques d'ingénierie des modèles. Nous prenons les exigences en entrée de notre processus et les divisons en exigences fonctionnelles et non fonctionnelles. Ensuite, nous appliquons un processus pour identifier les exigences qui sont adaptables et celles qui sont invariantes. Les progrès récents dans les techniques basées sur les buts en Ingénierie des Exigences nous ont poussé à intégrer ces techniques dans notre approche. En (Goal Oriented Requirements Engineering, GORE), les exigences non fonctionnelles sont exprimées sous la forme de buts, ce qui est beaucoup plus riche et complet dans la définition des relations entre les exigences. Ici, les exigences invariantes sont capturées par le concept de buts fonctionnels et les exigences adaptables sont capturées par le concept des buts non fonctionnels. Nous avons identifié quelques problèmes dans les méthodes classiques de modélisation des exigences et la vérification des propriétés. Ces approches ne tiennent pas compte les caractéristiques d'adaptabilité associées avec les systèmes auto-adaptatifs. Afin de valider notre approche, nous avons modélisé les exigences de deux études de cas et vérifié les exigences d'un étude de cas.

Mots-clés : Génie Logiciel, Ingénierie des Exigences, Ingénierie de Modèles, Transformation de Modèles, Systèmes Auto-Adaptatifs, Systèmes Ambiant, RELAX, Exigences Non Fonctionnelles, Vérification des Propriétés.

Discipline: Informatique

Institut de Recherche en Informatique de Toulouse — UMR 5505
Université de Toulouse Le Mirail, 5 Allée Antonio Machado 31100 Toulouse, France