Modeling and verification of Functional and Non-Functional Requirements of ambient Self-Adaptive Systems

Manzoor Ahmad, Nicolas Belloir, Jean-Michel Bruel

Modeling and verification of Functional and Non-Functional Requirements of ambient Self-Adaptive Systems

Manzoor Ahmad [a], Nicolas Belloir [a,*], Jean-Michel Bruel [b]

[a] University of Pau and the Pays of the Adour, LIUPPA, 64000 Cedex, France
[b] University of Toulouse, CNRS/IRIT, F-31062 Toulouse Université Cedex, France

A B S T R A C T

Self-Adaptive Systems modify their behavior at run-time in response to changing environmental conditions. For these systems, Non-Functional Requirements play an important role, and one has to identify as early as possible the requirements that are adaptable. We propose an integrated approach for modeling and verifying the requirements of Self-Adaptive Systems using Model Driven Engineering techniques. For this, we use Relax, which is a Requirements Engineering language which introduces flexibility in Non-Functional Requirements. We then use the concepts of Goal-Oriented Requirements Engineering for eliciting and modeling the requirements of Self-Adaptive Systems. For properties verification, we use OMEGA2/IFx profile and toolset. We illustrate our proposed approach by applying it on an academic case study.

1. Introduction

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 Dynamically-Adaptive Systems (DASs) (Whittle et al., 2009). Example applications that require DASs capabilities include automotive systems, telecommunication systems, environmental monitoring, and power grid management systems. In this context, 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. Self Adaptive Systems (SAS) like other systems, have 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 (Whittle et al., 2008). 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.

It is generally accepted that errors in requirements are very costly to fix (Lutz, 1993). 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 DASs are being conceived for applications that require a high degree of assurance (Kasten et al., 2003), 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 DASs, 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 offline. The consequent dynamic adaptation 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 DASs must operate.

Requirements Engineering (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 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. We identify the uncertainty in requirements of these systems and show how to verify it.

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

* Corresponding author. Tel.: +33559407571; fax: +33559407654.
E-mail addresses: manzoor.ahmad@univ-pau.fr (M. Ahmad), nicolas.belloir@univ-pau.fr, nbelloir@gmail.com (N. Belloir), bruel@irit.fr (J.-M. Bruel).

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 (Whittle et al., 2009). In this context, we provide an integrated approach to achieve this objective. We have used two approaches for defining and modeling requirements, i.e., Goal-Oriented Requirements Engineering (GORE) techniques are used to define and model the requirements of SAS (Goldsby et al., 2008; Lapouchnian et al., 2005; Yu et al., 2008; 2004) and SysML is used to specify the system and to provide a link with the requirements.

We propose a model-based requirements modeling and verification process for SAS that takes into account the uncertainty in requirements of these systems. We provide some tools to implement our approach and then apply it on an academic case study. The notion of goals is added to take into account the advantages offered by GORE. Requirements verification is done using a model checking technique.

This paper is organized as follows: In Section 2, we describe the background and the concepts which form the basis of this work, Section 3 shows the state of the art regarding RE for SAS and properties verification of these systems, Section 4 illustrates our proposed approach through an example and the tools that we have developed, Section 5 shows the case study that we used for the validation of our approach, and Section 6 concludes the paper and shows the future work.

2. Background

2.1. RELAX

Relax is an RE language for DASs in which explicit constructs are included to handle uncertainty. For example, the system might wish to temporarily Relax a non-critical requirement in order to ensure that critical requirements can still be met. The need for DASs is typically due to two key sources of uncertainty. First is the uncertainty due to changing environmental conditions, such as sensor failures, noisy networks, malicious threats, and unexpected (human) input; the term *environmental uncertainty* is used to capture this class of uncertainty. A second form of uncertainty is *behavioral uncertainty*, which 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 (Whittle et al., 2009).

2.1.1. 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. Relax also 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. 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 engineers.

Relax takes the form of a structured natural language, including operators designed specifically to capture uncertainty (Whittle et al., 2008); their semantics is also defined. Fig. 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.

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

Fig. 1. Relax operators (Whittle et al., 2009).

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

Fig. 2. Relax grammar (Whittle et al., 2009).

2.1.2. RELAX grammar

The syntax of Relax expressions is defined by the grammar shown in Fig. 2. 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, meaning 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 grammar.

The semantics of Relax expressions is defined in terms of Fuzzy Branching Temporal Logic (FBTL) (Moon et al., 2004). FBTL can describe a branching temporal model with uncertain temporal and logical information. It is the representation of uncertainty in FBTL that makes it suitable as a formalism for Relax.

2.1.3. RELAX process

Fig. 3 shows the Relax process. 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.

First of all, for each *SHALL* statement, we check whether it must always be satisfied or not. Then for each potentially Relax-able requirement, we identify the uncertainty factors. 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. Then we check whether the *SHALL* statement should be Relax-ed to handle uncertainty factors or not. Here we 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.

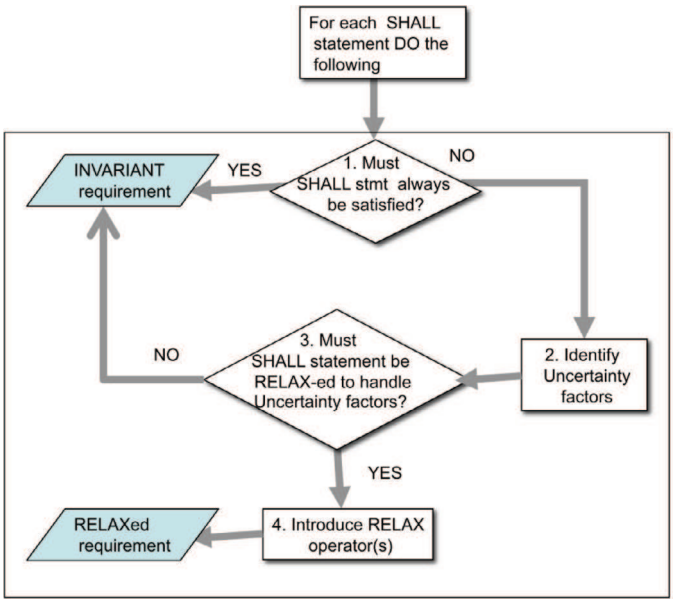

Fig. 3. Relax process (Whittle et al., 2009).

After the application of Relax process on traditional requirements, we obtain invariant and Relax-ed requirements. Relax-ed requirements support 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 runtime (Blair et al., 2009; Cheng et al. , 2009b).

2.2. SysML/KAOS

The SysML/Kaos (Gnaho and Semmak, 2010) model is an extension of the SysML[1] requirements model, with concepts of the Kaos goal model (Lamsweerde, 2009). SysML is an extension of Uml,[2] so 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 metamodel is implemented as a new profile, importing the SysML profile.

2.2.1. 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. 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 constraints, which is used to integrate with other engineering analysis models.

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.

2.2.2. KAOS

Kaos is a goal-oriented 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, i.e., for describing the problem to be solved and the constraints that must be fulfilled by any solution provider. Kaos has been designed: (i) To fit problem descriptions by allowing to define and manipulate concepts relevant to problem description; (ii) To improve the problem analysis process by providing a systematic approach for discovering and structuring requirements; (iii) To clarify the responsibilities of all the project stakeholders; (iv) To let the stakeholders communicate easily and efficiently about the requirements.

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

2.2.3. Why SysML/KAOS?

SysML and Kaos have some advantages and weak points, but these are complementary to each other based on the following points: (i) Requirements description: A textual description in SysML and a description in the form of goals in Kaos; (ii) Relation between requirements: SysML has <<*contain*>> and <<*derive*>> relations; these relations do not have precise semantics, which leads to confusion. Kaos has refinement relations AND/OR; (iii) Traceability relations: <<*satisfy*>> and <<*verify*>> relations in SysML allow to define traceability. Kaos does not have explicit traceability relations; (iv) Tools: A number of tools exist for SysML; most of them are open source. Kaos propose a proprietary tool called Objectiver.[3]

Traditionally, requirements are divided into Functional Requirements (FRs) and Non-Functional Requirements (NFRs). Due to the complexity of systems, NFRs should be processed much earlier than when they are usually handled in most development processes, 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 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. SysML/Kaos is inspired from the work of Chung et al. (1999) and Cysneiros and Leite (2004). The SysML/Kaos model allows both FRs (Laleau et al., 2010) and NFRs (Gnaho and Semmak, 2010) to be modeled.

2.2.4. SysML/KAOS meta-model

Fig. 4 shows the extended meta-model of SysML/Kaos (Gnaho and Semmak, 2010); non-functional concepts are represented as yellow boxes (bottom), the gray boxes (top) represent the SysML concepts. The instantiation of the meta-model allows us to obtain a hierarchy of NFRs in the form of goals. Non-Functional Goals (NFGs) are organized in refinement hierarchies. The meta-class NonFunctionalGoal represents the Non-Functional Goal (NFG), it is specified as a sub-class of the meta-class Goal, which itself is a subclass of the meta-class Requirement of SysML. An NFG represents a quality that the future system must have. The nFGType specifies the type of NFG and the attribute topic represents 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 AbstractNFG or an ElementaryNFG. A goal that cannot be further refined is an ElementaryNFG. The refinement of an AbstractNFG by either abstract or elementary goals is represented by the AssociationClass Refinement. An AbstractNFG may contain several combinations of subgoals (abstract or elementary). The relationship Refinement becomes an AssociationClass between an AbstractNFG and its subgoals. 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 ElementaryNFGs. For that, the SysML/Kaos meta-model considers the concept of the meta-class ContributionGoal. A ContributionGoal captures a possible way to satisfy an ElementaryNFG. The AssociationClass 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* (resp. *negative*) contribution helps positively (resp. negatively) to the satisfaction of an ElementaryNFG. A *direct contribution* describes an explicit contribution to the ElementaryNFG. 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 ContributionGoal has an effect on FGs.

2.3. The OMEGA2 UML/SysML profile and IFx toolset

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. We use OMEGA2/IFx profile and toolset for the properties verification and model simulation of our case study.

2.3.1. The OMEGA2 Profile

OMEGA2 profile (Ober and Dragomir, 2010) 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 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 which provides static analysis, simulation and timed automaton-based model-checking (Clarke et al., 1999) techniques for validation.

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 Diagram (IBD): parts that communicate through ports and connectors. For the SysML Block Definition Diagram (BDD), the following concepts are taken into account: blocks and their relationships (association, aggregation, generalization), interfaces, basic types, signals.

For the system behavior, the OMEGA2 profile takes into account the following concepts: State machines (excluding: history states, entry point, exit point, junction) and Actions; for this, the profile defines a concrete syntax. 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.

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 satisfaction (or not) of safety properties. The main issue in modeling *observers* is the choice of events which trigger their transitions.

The trigger of an *observer* 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.

2.3.2. IFx toolset

OMEGA2 models can be simulated and properties can be verified using the IFx toolset (Bozga et al., 2004). The IFx toolset

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

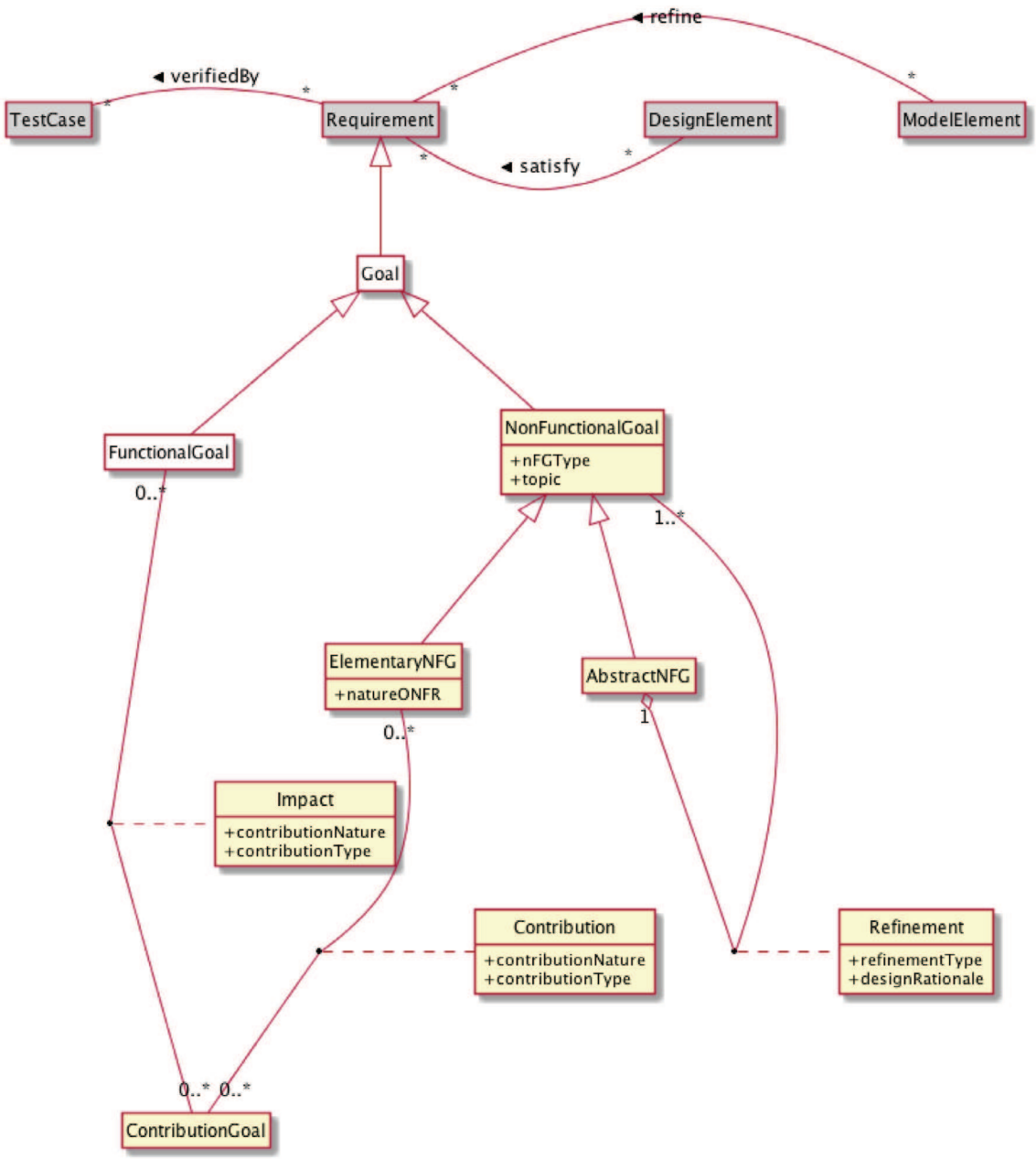

Fig. 4. SysML/Kaos meta model (Gnaho and Semmak, 2010). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

provides verification which ensures 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). The IFx toolset also provides simulation which designates the interactive execution of an OMEGA2 UML/SysML model. 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 toward a simple specification language based on an asynchronous composition of extended timed automata: the IF language,[4] and on the use of simulation and verification tools available for IF. The translation takes an input model in XML Metadata Interchange (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 OMEGA2/IFx approach has been applied for the verification and validation of industry grade models (Dragomir et al., 2012) providing interesting results.

3. State of the art

Different roadmap papers on Software Engineering (SE) for SAS (Cheng et al., 2009b; Rogério de Lemos et al., 2013) discuss the state of the art, its limitations, and identify critical challenges. Cheng et al. (2009b) present a research roadmap for SE of SAS focusing on four views, which are identified as essential: requirements, modeling, engineering, and assurances. The focus is on development

[4] http://www-if.imag.fr/

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 roadmap paper (Rogério de Lemos et al., 2013) 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.

3.1. 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 question arises as to what aspects of the environment the SAS should 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. It is important to identify these properties as early as possible.

Levels of Requirement Engineering for Modeling (LoREM) (Goldsby et al., 2008) is an approach for modeling the requirements of Dynamic-Adaptive Systems (DAS) using i* goal models (Yu, 1997). The i* goal models are used to represent the stakeholder objectives, non-adaptive system behavior (business logic), adaptive behavior, and adaptation mechanism needs of DAS. Each of these i* goal models addresses the three RE concerns (conditions to monitor, decision-making procedure, and possible adaptations) from a specific developers perspective.

Awareness Requirements (AwReqs) (Vítor et al., 2011) 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. AwReqs are represented in a formal language and can be directly monitored by a requirements monitoring framework.

CLAIMS (Welsh and Sawyer, 2010; Welsh et al., 2011) were applied as markers of uncertainty to record the rationale for a decision made with incomplete information in DASs. The work in Ramirez et al. (2012a) 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.

RELAX can be used in goal oriented modeling approaches for specifying and mitigating sources of uncertainty in DASs (Cheng et al., 2009a). AutoRELAX (Ramirez et al., 2012b), 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 also requires an executable specification of the DAS, such as a simulation or a prototype, which applies the set of utility functions to measure how well the DAS satisfies its requirements in response to adverse conditions. 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.

Fuzzy Live Adaptive Goals for Self-Adaptive Systems (FLAGS) (Baresi et al., 2010) is an innovative goal model which deals with the challenges posed by SAS. Goal models have been used for representing systems requirements, and also for tracing them onto their underlying operationalization.

The state of the art regarding RE for SAS shows different approaches from the point of view of its complementarity with RELAX. The different steps in LoREM are interesting but our focus is on RELAX-ed requirements as we want to identify the uncertainty in the requirements of DASs. Regarding AwReqs, in future work, we want to integrate this concept into our approach using Monitor-Analyze-Plan-Execute (MAPE) (Kephart and Chess, 2003) feedback loop that operationalizes the system's adaptability mechanisms. 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. A CLAIM can also be monitored at runtime to prove or disprove its validity (Welsh et al., 2011), thereby triggering adaptation to reach more desirable system configurations if necessary. CLAIMS therefore complement RELAX.

3.2. Properties verification of SAS

For the properties verification of SAS, we use the OMEGA2/IFx profile and toolset which was developed in our team (Ober and Dragomir, 2010). The advantage of the OMEGA2 profile is that it provides the notion of *observers* for specifying and verifying dynamic properties of models. In terms of properties verification, there exists a number of techniques. In the following, we give a description of some of it.

Benghazi et al. (2009) 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 NFRs. It only focuses on safety and timeliness properties, to assure the correct functioning of Ambient Assisted Living (AAL) systems and to show the applicability of this methodology in the context of this kind of system.

Apvrille et al. (2004) introduce a profile named Timed UML and RTLOTOS 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) (Courtiat et al., 2000). With its formal semantics and toolkit, TURTLE enables a priori detection of design errors through a combination of simulation and verification/validation techniques.

In Laleau et al. (2010), the authors propose an extension to SysML with concepts from the goal model of the KAOS method (SysML/KAOS) with rules to derive a formal B (Abrial, 1996) 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.

In MEDISTAM-RT, the focus is on safety and timeliness properties, we do not treat any specific type of properties. We verify those requirements that are of interest for adaptation in SAS. In TURTLE, design errors can be detected through simulation and verification. That is the reason why we plan to explore the complementarity of this approach with our approach. The use of formal methods like B can help avoid the state space explosion problem which is inherent in model checking techniques. We have worked on studying the complementarity of these two approaches and we plan to integrate them in our approach in the future work.

4. Proposed approach

In this section, we introduce the overall view of our proposed approach (Ahmad, 2013). We show our contribution then we describe the overall process of our approach. To illustrate our proposed approach, we use requirements from the barbados Car Crash Crisis Management System (bCMS) case study. At the end, we show the integrated tooling environment that we developed to validate our approach.

4.1. Contribution

To properly define the scope of our contribution, it is necessary to identify the work we have done. Firstly, we have found that although the use of traditional process of SysML/KAOS was interesting for modeling the requirements of SAS, it does not take into account

the notion of uncertainty. On the other hand, RELAX is a process tailored to identify and highlight the uncertainty, but it does not provide tools for its implementation. Finally, the verification techniques used for these models do not take into account the uncertainty posed by these systems. Based on this observation, we contributed toward the definition of an integrated tool-based process. For this, we developed support for RELAX. Then we developed rules to transform requirements addressed by RELAX to SysML/KAOS, using model transformation techniques. Finally, we integrated formal verification techniques i.e., OMEGA2/IFx in the process. To reduce the risk of state space explosion problem (Clarke et al., 2012) when we take into account the whole system using OMEGA2/IFx, we limited its use to verify only adaptable properties. We present in detail the work and the overall process in the next section.

4.2. The proposed approach

In the following, each step of the proposed approach is explained with associated input and output. Fig. 5 shows the overall view of our proposed approach.

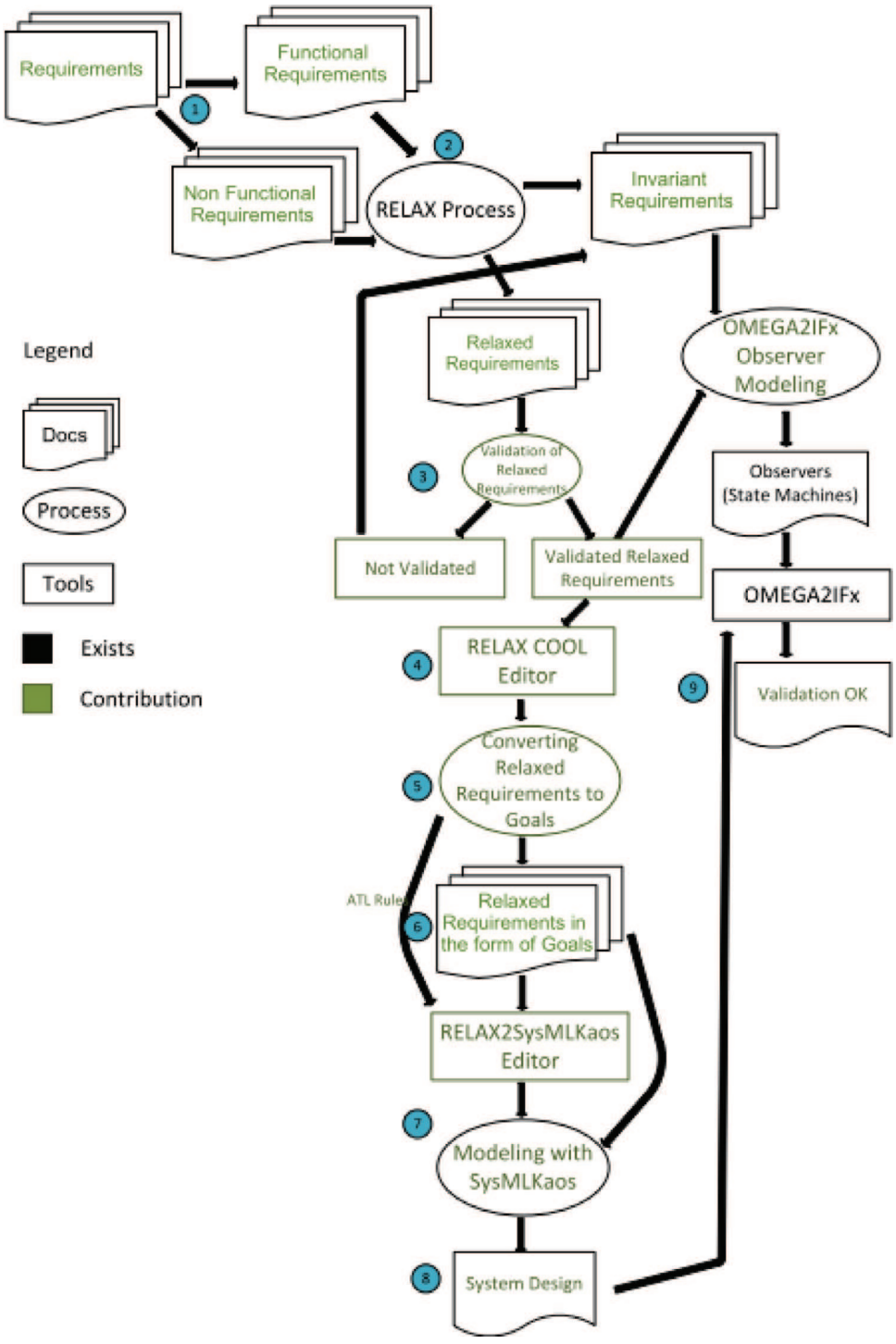

Fig. 5. Overall view of our approach.

1. The overall approach that we propose takes requirements as input. These requirements are elicited in the form of *SHALL* statements by a requirement engineer which are then divided into FRs and NFRs.
2. We apply RELAX process (see Section 2.1.3) 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. Here, we validate the RELAX-ed requirement with the help of an expert i.e., for each RELAX-ed property, we check whether the new expression of the property is acceptable or not. By *acceptable* we mean two things: (i) the RELAX-ed expression is sound (it can be operationalized), and (ii) the boundaries make sense (from the domain expert point of view). If the RELAX-ed expression is acceptable then we proceed with the next step, if it is not acceptable, we propose two options: cancel the RELAX-ation and go back to a *SHALL* invariant or complement the RELAX-ed property with an additional invariant (e.g., a *max* or *min* boundary that constraints the RELAX-ed expression).
4. The resulting RELAX-ed requirements are then formalized 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[5] is used for the development of this editor.
5. 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 (see Section 4.3.1) for the correspondence between RELAX-ed requirements and SYSML/KAOS concepts (Ahmad et al., 2012b). For this purpose, we have developed a tool called RELAX2SYSML/KAOS editor, which is based on Atlas Transformation Language (ATL) transformations. For the time being, the tool helps in mapping the RELAX concepts to SYSML/KAOS concepts but not the inverse.
6. At this step, we have a full list of RELAX-ed requirements with uncertainty factors converted into SYSML/KAOS goal concepts.
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. Once we have the system design, we use the OMEGA2/IFx *observers* to verify the properties of SAS. The input to this step are the OMEGA2/IFx *observers* which are the RELAX-ed and invariant requirements. 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.

4.3. Integration of the approaches

In the following, we present how we defined the convergence between different methods used in our approach.

4.3.1. Relationship between RELAX, SysML/KAOS and SysML

In our integrated approach, we take benefit of SYSML/KAOS while modeling RELAX-ed requirements of SAS. In Fig. 6, we show how several key concepts are taken into account in the selected approaches. The concepts are taken from RELAX and are then compared with the other approaches.

- In SYSML/KAOS, requirements are described in the form of goals; SYSML describes requirements in textual form; RELAX requirements are also in textual form which contains more information in the form of RELAX operators.
- 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*.
- 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*. 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 Dependency/Impact, SYSML/KAOS describes it as an *Impact* of a *Contribution Goal* on a Functional Goal (FG). 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.
- 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,[6] Papyrus,[7] topcased,[8] etc. and for RELAX, we have developed an eclipse-based RELAX COOL editor (Bascans et al., 2013). We have also developed RELAX2SYSML/KAOS editor which does the mapping between RELAX uncertainty factors and SYSML/KAOS goal concepts.

4.3.2. Uncertainty factors/impacts

SYSML/KAOS is a GORE approach that takes into account different kinds of dependencies between *Goals* and *Contribution Goals*. RELAX deals with dependency in terms of the dependency of a RELAX-ed requirement on an invariant requirement but it does not say anything about the dependency of a *Monitor* (*Contribution Goal* in SYSML/KAOS) on *ENV* (*Goal* in SYSML/KAOS). So the injection of SYSML/KAOS in our approach helps in capturing the dependencies between different requirements and also between the monitors and environment. RELAX uses a kind of vocabulary that only captures uncertainty in the requirements of SAS while KAOS helps in allowing the stakeholders communicate easily and efficiently about requirements.

RELAX uncertainty factors, especially *ENV* and *MON*, are particularly important for documenting whether the system has means for monitoring the important aspects of the environment. By collecting these *ENV* and *MON* attributes, we can build up a model of the environment in which the system will operate, as well as a model of how the system monitors its environment. In RELAX, requirements dependencies are delimited by the uncertainty factor *DEP*, as it is important to assess the impact on dependent requirements after RELAX-ing a given requirement. Having said this, SYSML/KAOS can complement RELAX by injecting more information in the form of *positive/negative* and *direct/indirect* impacts (Ahmad et al., 2012a), which models the impact of a *Contribution Goal* on an *Elementary Goal*. 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.

[5] http://www.eclipse.org/Xtext/

[6] http://www.eclipse.org/

[7] http://www.papyrusuml.org

[8] 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

Fig. 6. Relationship b/w SysML/Kaos SysML and Relax.

4.3.3. 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.

4.4. Proposed approach illustration

To illustrate our approach, we use the bCMS [9] case study. Here is an excerpt of the case study.

The bCMS is a distributed crash management system that 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. 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. There is no central database; fire and police stations maintain separate databases and may only access information from the other database through the bCMS system. Each coordination process is hence in charge of adding and updating information in its respective database. Fig. 7 shows the overall view of the bCMS case study.

We have chosen an (illustrative) subset of the bCMS requirements. The requirements are numbered in a shared document.[10]

We have first applied the Relax process on bCMS requirements to get invariant and Relax-ed requirements. For Relax-ed requirements, all the uncertainty factors were identified. Then using the correlation in Fig. 6, we have modeled the bCMS system requirements with the SysML/Kaos approach. In Ahmad et al. (2013a), we have modeled some more requirements of the bCMS case study. Following are some of the Relax-ed requirements that we identified:

- Relax-ed requirements: R4, R8.

Fig. 8 shows the uncertainty factors associated with the *Integrity* R4 (*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.

Fig. 9 shows the uncertainty factors associated with the *Availability* R8 (*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* 0 *minutes AND* $\leq$ 30 *min for every 48 h when no crisis is active.*) Relax-ed requirement.

Fig. 10 shows a low level goal model of the bCMS case study. We have identified a goal *"Ensure the integrity of communications b/w*coordinators[bCMS]"*which is an abstract non-functional goal and*

[9] Available at http://cserg0.site.uottawa.ca/cma2013re/CaseStudy.pdf.

[10] Available at http://goo.gl/uscP5

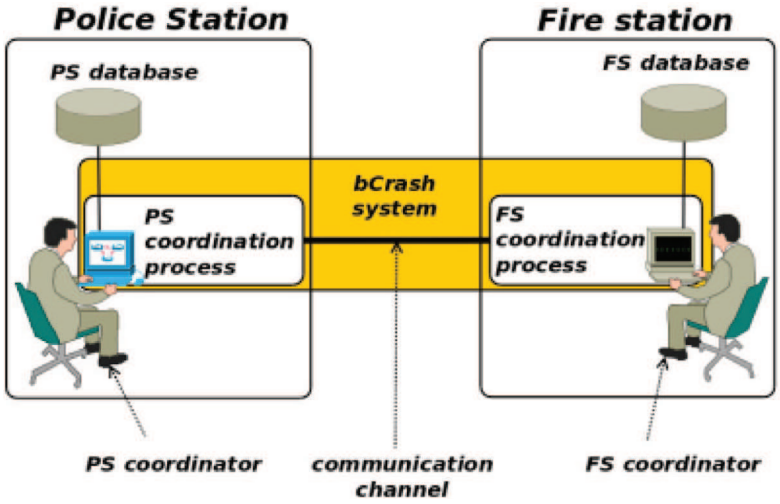

Fig. 7. bCMS case study overall view.

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

Fig. 8. R4 integrity RELAX-ed requirement uncertainty factors.

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

Fig. 9. R8 availability RELAX-ed requirement uncertainty factors.

is AND-refined into two sub-goals using refinement by type: (i) Integrity of communication b/w coordinators[bCMS] and (ii) *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 *Authenticity 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]*. The functional goal R3: *A PSC maintains control over a crisis situation by communicating with the FSC as well as policemen.* This goal is AND-refined into two sub-goals: *To provide coordinated route plan* and *To estimate resources*. The *Contribution Goal Communication Compromiser* has an *indirect* and *negative* impact on the functional goal *To estimate resources*. The property verification part of our proposed approach is illustrated in Section 5.2.1.

4.5. Tools support

In this section, we introduce the tools that implements our proposed approach.

4.5.1. RELAX editor

For the generation of RELAX editor, Xtext is used. Xtext is a framework for the development of Domain Specific Languages (DSL) 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. An initial version of the RELAX editor can be found in Ahmad (2010). The RELAX grammar is used as a meta-model for this editor which is generated by Xtext that we call RELAX.ecore. Fig. 11 shows an example of the RELAX file with

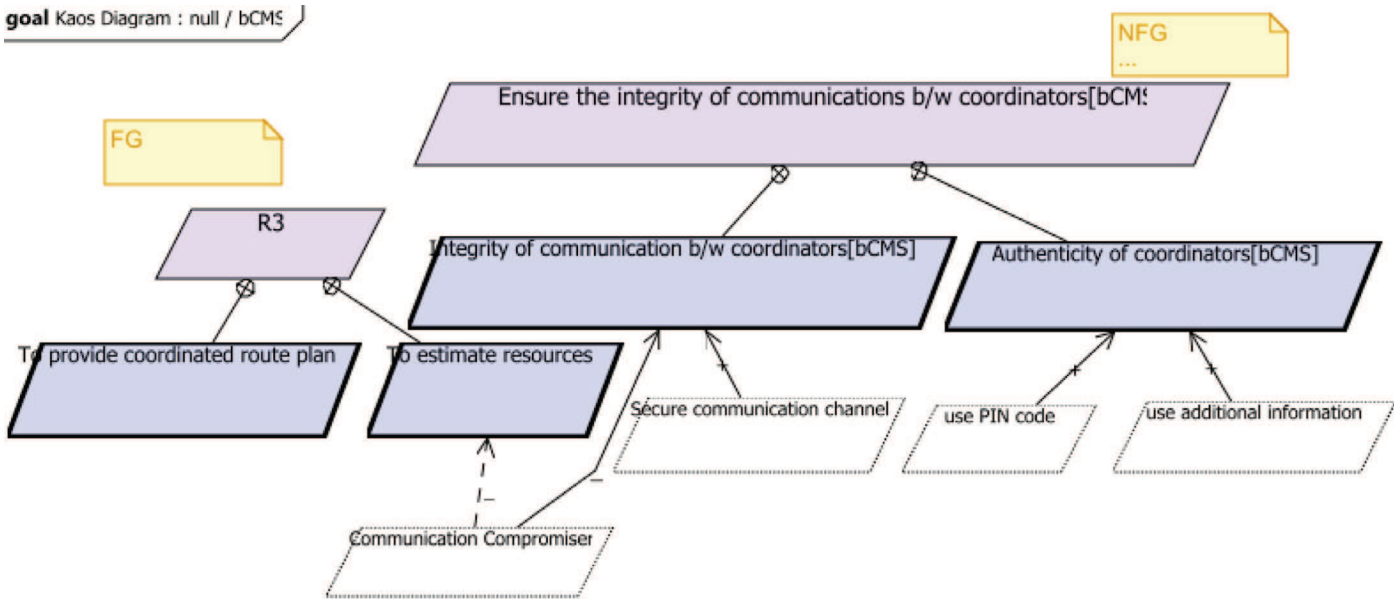

Fig. 10. Low level goal model.

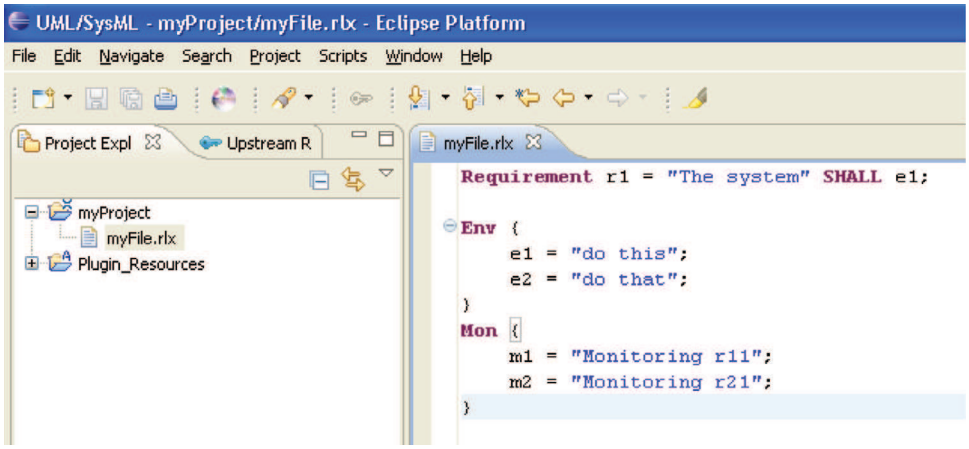

Fig. 11. RELAX file.

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.

4.5.2. RELAX to SysML/Kaos transformation

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.

4.5.3. 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.

4.5.4. Mapping between RELAX and SysML/Kaos elements

Here, we present the relationship between RELAX and SYSML/KAOS elements. The RELAX abstract syntax is defined in the RELAX meta-model. In turn, the SYSML/KAOS abstract syntax is defined in the SYSML/KAOS meta-model.

Fig. 6 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 Fig. 12, an *ENV* is mapped to an *Elementary Goal* and *MON* is mapped to *Contribution Goal*. Fig. 13 shows the generated SYSML/KAOS model after the application of ATL rules. Fig. 14 shows the SYSML/KAOS model opened in the editor.

5. Proof of concepts

In this section, we apply our approach on an academic AAL case study. 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 (Benghazi et al., 2009). 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 (Cleland-Huang et al., 2007). We model the requirements of an

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

Fig. 12. Relaxed requirement to abstract goal mapping.

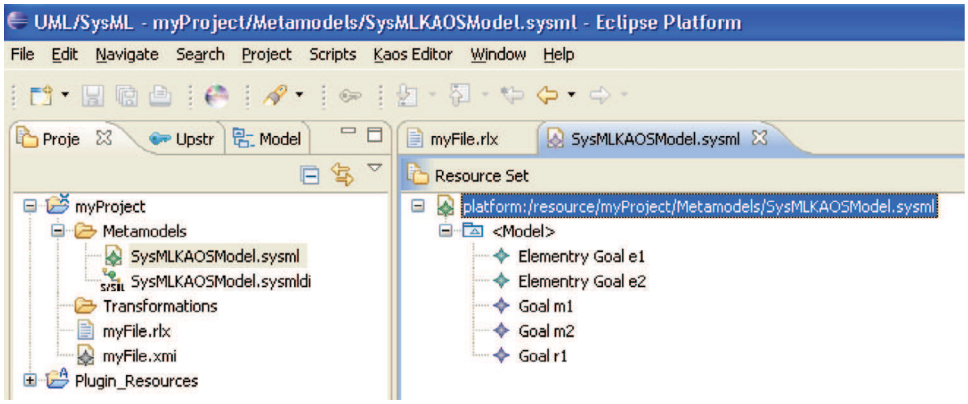

Fig. 13. SysML/Kaos model.

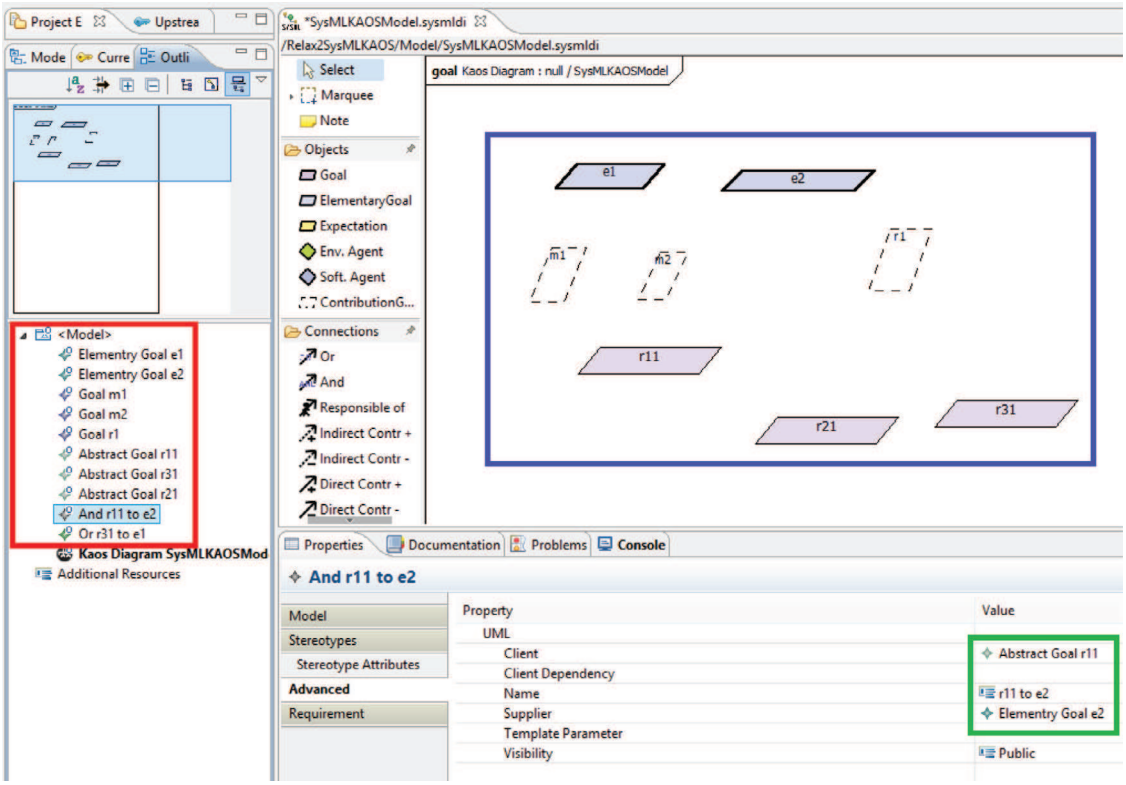

Fig. 14. Generated SysML/Kaos model using ATL transformations.

AAL[11] home which ensures the health of a *Patient* like the one studied by research teams at the IUT of Blagnac.[12] We then show the verification of some of the properties of the AAL system.

5.1. Requirements modeling of the AAL case study

Fig. 15 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 (Whittle et al., 2009).

Fig. 16 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*. Ahmad (2014) shows the application of RELAX process on some of the requirements of the AAL case study.

[11] http://www.iese.fraunhofer.de/fhg/iese/projects/med_projects/aal-lab/index.jsp
[12] http://mi.iut-blagnac.fr/

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.

Fig. 15. AAL case study.

Relax Requirement:

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

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)

Fig. 16. RELAX requirement example.

5.1.1. High level goal model

Fig. 17 shows the high level goal model of the AAL. 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 visits from a technician could be a factor of disturbance for *Mary* and unfeasible due to the large number of AAL houses across the world. 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 *Contribution 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 *Elementary NFGs* using refinement by subject. The sub-goal *Precision [Location Detection]* can be satisfied by a *positive* and *direct* contribution by one of the following *Contribution Goals: combine data from multiple sensors, combine multiple features* and *use redundant features*. The *Contribution Goal combine data from multiple sensors*, contribute indirectly and negatively to the satisfaction of the sub-goal *Performance [AAL System]*.

5.1.2. Low level goal model

Fig. 18 shows the security goal model of AAL. In order to further extract new goals from the AAL system, we identify another goal, *Security [fridge data]*, which is an *Abstract NFG* that can be AND-refined into three sub-goals using refinement by type: *Confidentiality [fridge data]*, *Integrity [fridge data]* and *Availability [fridge data]*. Similarly, the sub-goal *Availability [fridge data]* can be refined into two sub-goals using refinement by subject: *Availability [Storing RFID information]* and *Availability [Sensors data]*. The *Contribution Goal having high-end sensors* contributes directly and positively to the goal *Availability [Sensors data]*, and may contribute indirectly and positively to *Integrity [fridge data]*.

5.2. 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 (Nehmer et al., 2006). In this section, we show how we used OMEGA2/IFx (Verimag and Irit, 2011) for the properties verification and model simulation of AAL system.

5.2.1. Modeling the AAL system with OMEGA2 profile

We start by taking into account the structural part of the AAL system. Those parts are considered that are concerned with the daily calorie 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 calorie consumption of the *Patient* as the *Patient* needs not to exceed it after a certain threshold.

Fig. 19 shows the main Internal Block Diagram (IBD). The communication between different blocks takes place through ports. In

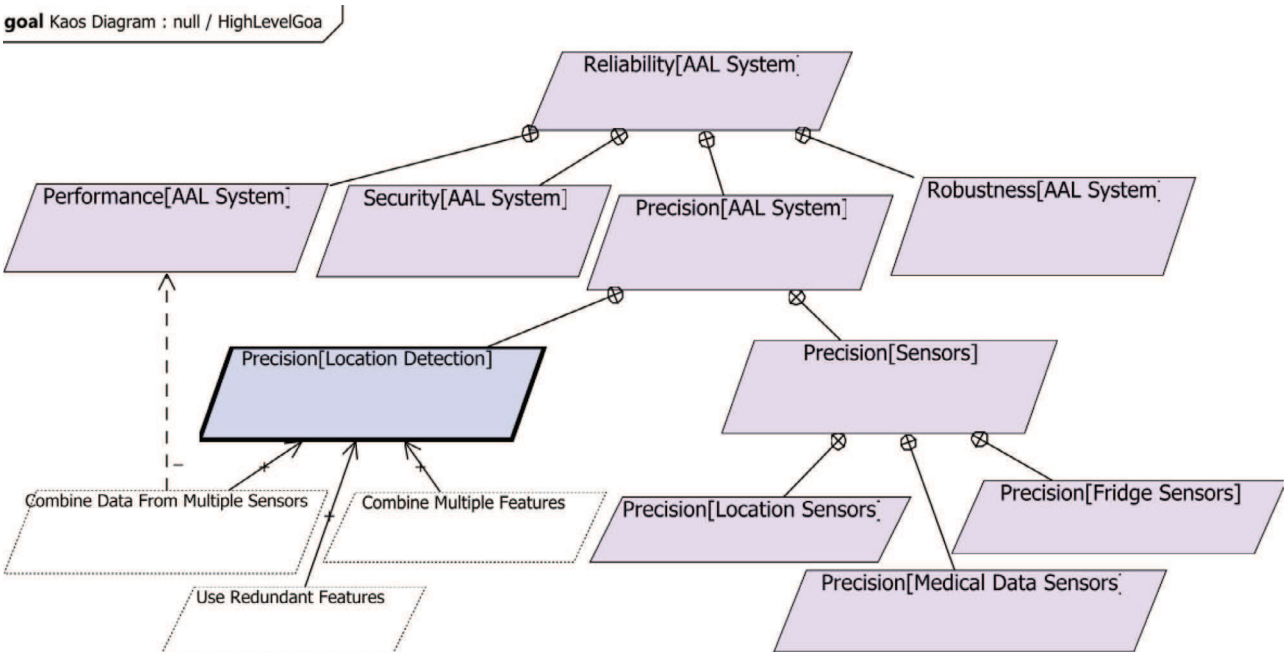

Fig. 17. High level goal model.

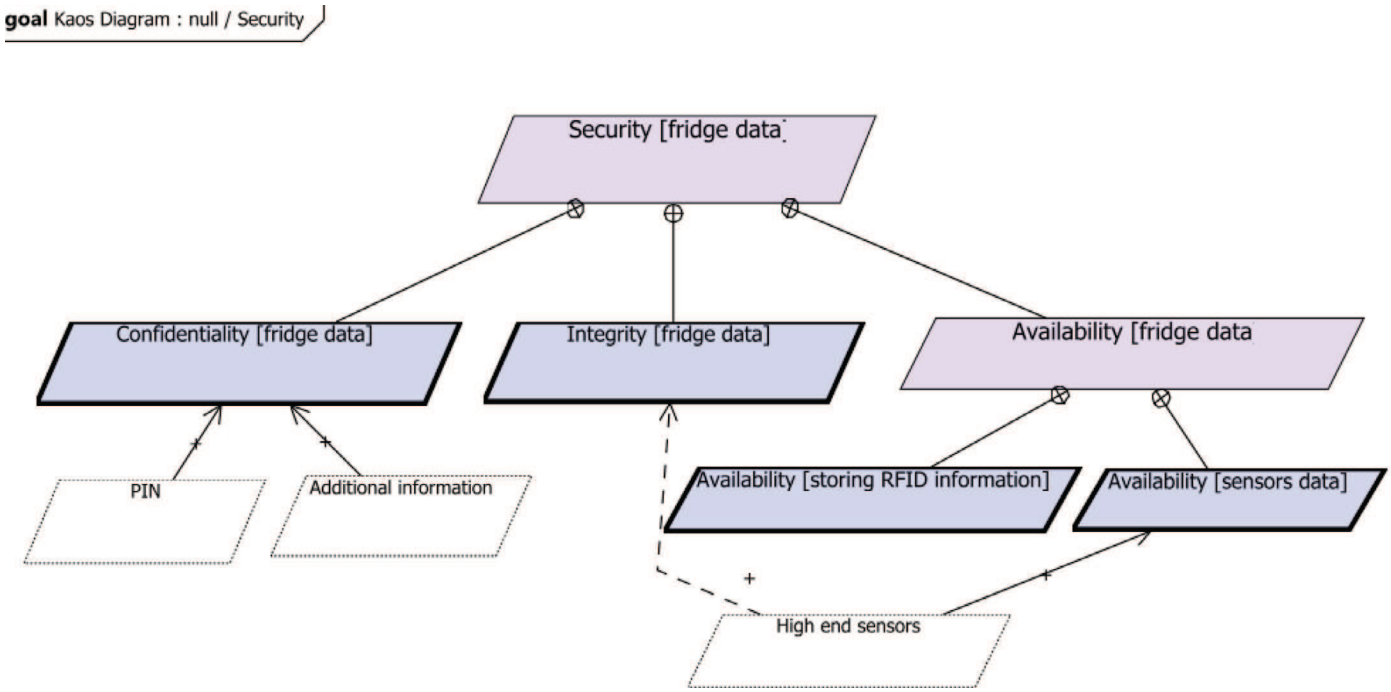

Fig. 18. Security goal model.

Fig. 19, 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 important parts of the AAL system are *Patient* and *Fridge*. A *Fridge* in turn is composed of *Display, Alarm, Controller*, and *Food* blocks. Fig. 20 shows the IBD for the *Fridge* block. Each of the four blocks behaviors is modeled in a separate State Machine Diagram (SMD). The *Food* block contains information about the *Food* items in the *Fridge*, the calories contained in each item, the total number of calories the *Patient* has accumulated and the calorie 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* calorie level surpasses a certain threshold.

Fig. 21 shows the SMDfor the *Patient* block. Here, the exchange of information between *Patient* and *Fridge* takes place. The number and quantity of each item present in the *Fridge* is identified. If a certain product still present in the *Fridge* is chosen by the *Patient* then the information is communicated with the *Fridge* and the list is updated. Otherwise the *Fridge* is empty and the *Patient* will wait 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.

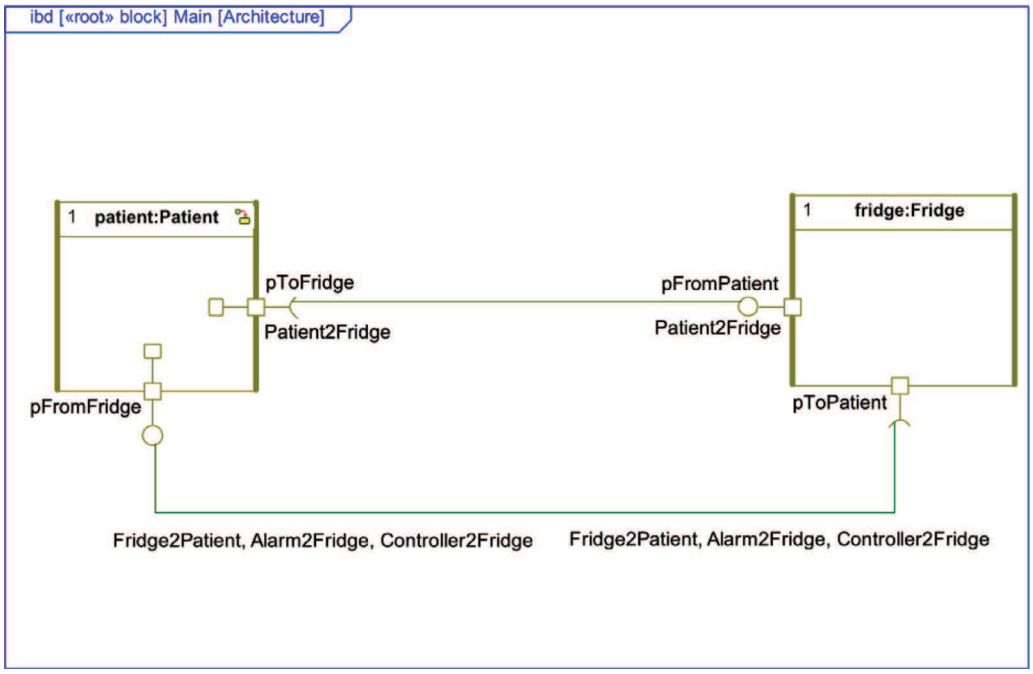

Fig. 19. Main Internal Block Diagram.

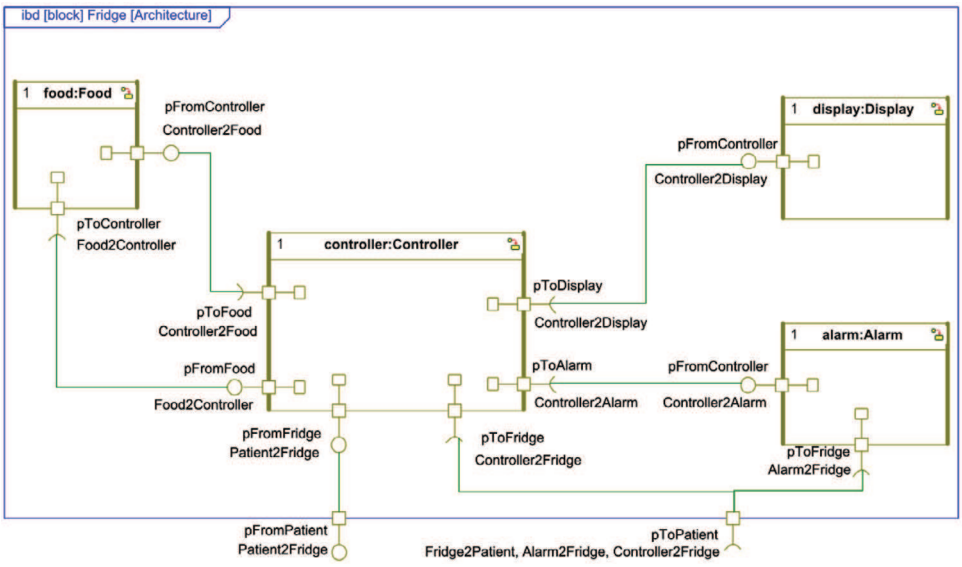

Fig. 20. Fridge Internal Block Diagram.

5.2.2. Properties verification of the AAL system

Below are the properties to be verified (Ahmad et al., 2013b).

Property 1: The Fridge SHALL detect and communicate information with AS MANY Food packages AS POSSIBLE. A RELAX-ed version of this requirement with all the uncertainty factors is shown in Fig. 16.

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 as many *Food* items present in the *Fridge* as possible. Fig. 22 shows the SMD of the *Property 1*. The trigger for this property is an *observer* transition which is a *match* clause specifying the type of event (e.g., send), some related information (e.g., eat) and *observer* variable (e.g., p) that may send related information. 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*, then the next step is to calculate the number of items consumed. 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 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 or equal to -1 or greater than or equal to 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.

We now consider the invariant requirement. *Property 2: The Alarm SHALL be raised instantaneously if the total number of calories surpasses the maximum calories allowed for the Patient.*Fig. 23 shows the SMDfor

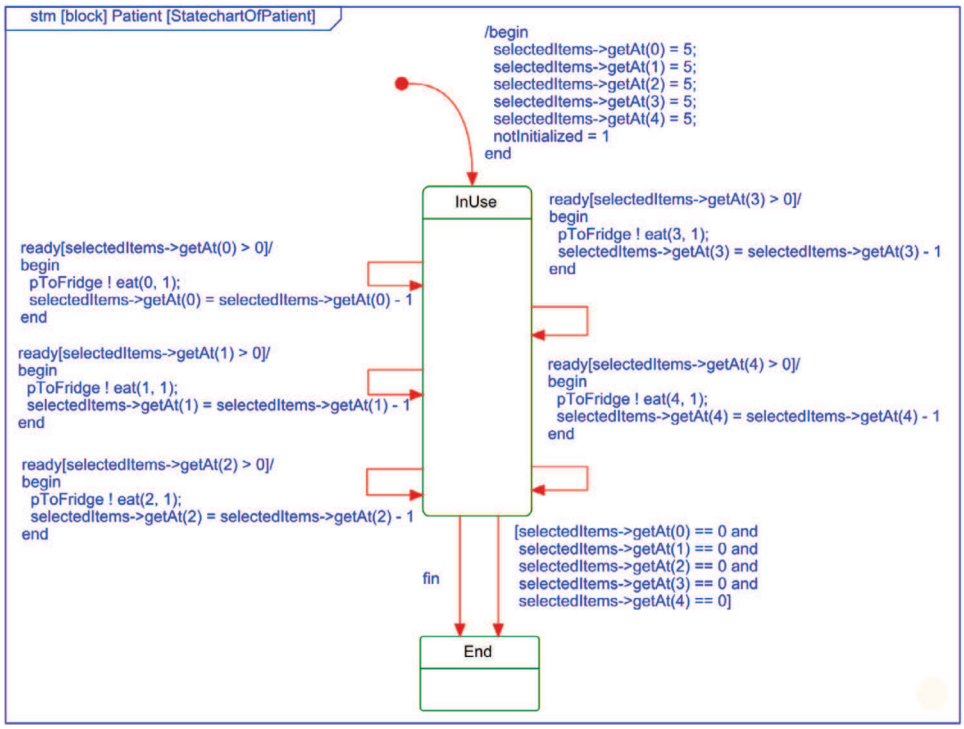

Fig. 21. Patient State Machine Diagram.

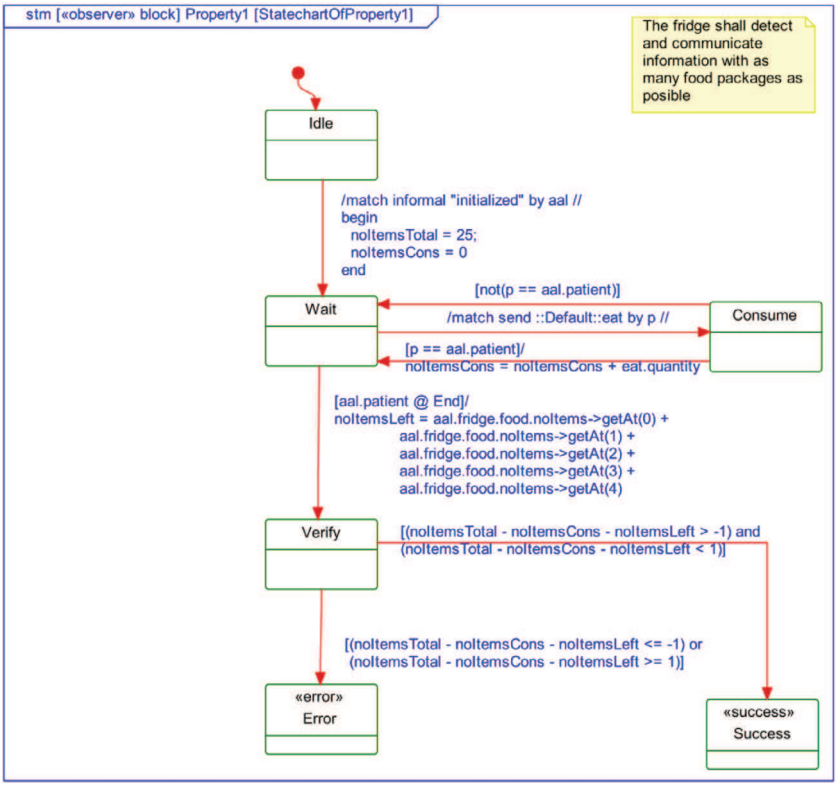

Fig. 22. Property1 State Machine Diagram.

property 2. 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. 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.

5.2.3. *Verification results*

Until now, the AAL system is modeled along with the properties to be verified on the model. We now show how to verify these

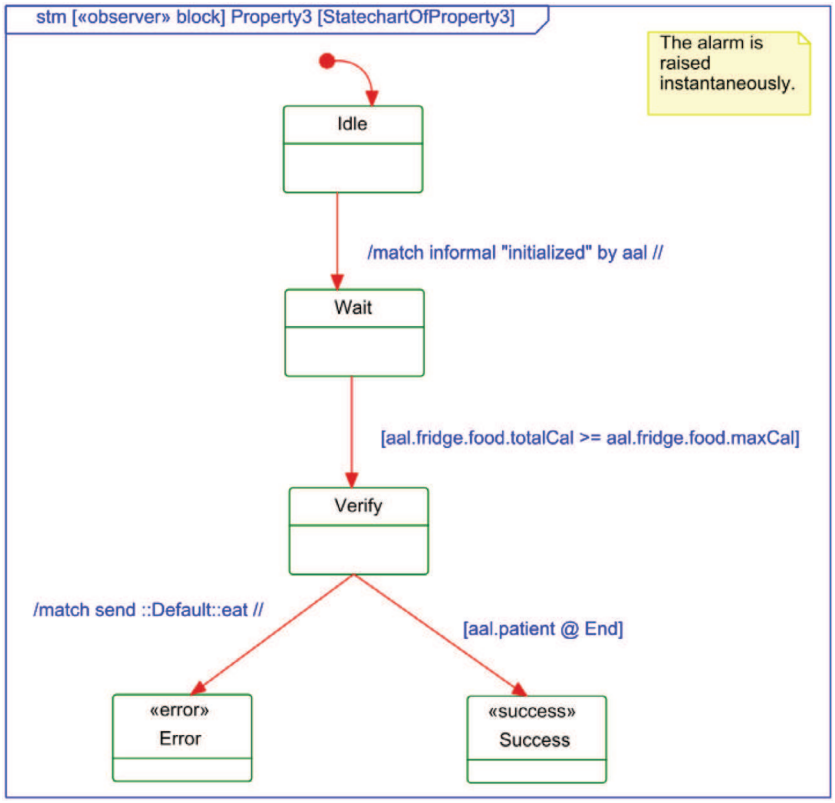

Fig. 23. Property2 State Machine Diagram.

properties using the IFx toolset. The AAL2 model is first exported into *AAL2.xmi* and then using the IFx toolset the *AAL2.xmi* is compiled into *AAL2.if* (Fig. 25). The *AAL2.if* is compiled into an executable file i.e., *AAL2.x* (Fig. 26). While verifying the AAL model, the model checker has found several error scenarios (Fig. 27). Any of the error scenarios 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. We check the states of the *observers* in order to identify which property has not been satisfied. In this case, *Property 2* fails. While checking the state of the entire system for this property, we discover that the *<<error>>* state contained the maximum 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* does not function properly which is strictly linked to its *Food* process. One can observe in the SMDof the *Food* block (Fig. 24) that the *Alarm* is raised only if the total number of consumed calories is strictly superior to the maximum allowed; a condition which does not satisfy the request that the *Alarm* is raised as soon as possible. The correction consists of raising the *Alarm* also in case the total number of consumed calories is equal to the maximum allowed threshold. Once this error is corrected in the SMDof the *Food* block, the verification succeeds.

6. Conclusion and future work

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 is to propose an integrated approach for modeling and verifying the requirements of SAS using Model Driven Engineering (MDE) techniques. It takes requirements as input and then by applying various processes and tools, we integrate the notion of uncertainty 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.

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 against other requirements in situations where the resources are constrained or priority must be given to requirements. 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. We then use SysML/Kaos which is an extension of the SysML requirements model with concepts of the Kaos goal model. Here, invariant requirements are captured by the concept of FGs whereas Relax-ed requirements are captured by the concept of NFGs. 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 into 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. We have developed a tool called Relax2SysML/Kaos editor which is capable of modeling the Relax-ed requirements in the form SysML/Kaos goal concepts. We provide a mechanism to verify some adaptable and invariant properties of the SAS using formal method technique OMEGA2/IFx. In order to validate our proposed approach, we have applied it to an academic Ambient Assisted Living case study.

Our work resides within the framework of self-adaptation, but we do not treat the development of self-adaptation mechanisms. We help SAS developers by providing a mechanism for identifying the uncertainty associated with the requirements of these systems. Fig. 28 shows a table with the pros and cons of our proposed approach.

In terms of the future work, we have applied our approach to an academic case study. The next step is to apply it to a real industrial

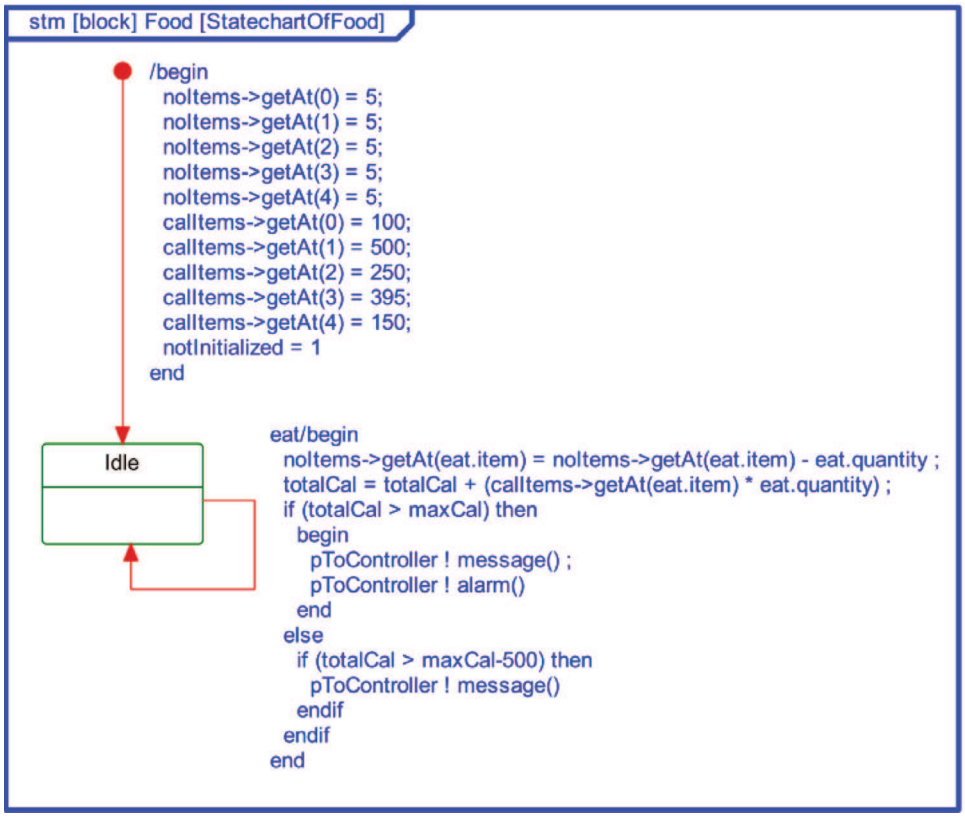

Fig. 24. Food State Machine Diagram.

```
[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 ~]$
```

Fig. 25. XMI to IF compilation.

```
[ahmad@topcased ~]$ if2gen -tdbm AAL2.if
Compiled IF spec...
m4 -I/home/dragomir/if2/src/code -D_CTYPE_=h AAL2.m4 > AAL2.h
m4 -I/home/dragomir/if2/src/code -D_CTYPE_=C AAL2.m4 > AAL2.C
g++ -c -o AAL2.o -Wall -Wno-deprecated -O3 -DABSTRACT -I/home/dragomir/if2/src/simulator-t -I/home/dragomir/if2/src/simulator AAL2.C
AAL2.C: In member function 'void if_Default_Property1_instance::_Idle_1_fire(IfMessage*)':
AAL2.C:24425: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_Default_Property1_instance::_Idle_1a_fire(IfMessage*)':
AAL2.C:24442: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_Default_Property2_instance::_Idle_1_fire(IfMessage*)':
AAL2.C:26060: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_Default_Property2_instance::_Idle_1a_fire(IfMessage*)':
AAL2.C:26077: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_Default_Property3_instance::_Idle_1_fire(IfMessage*)':
AAL2.C:26926: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_Default_Property3_instance::_Idle_1a_fire(IfMessage*)':
AAL2.C:26943: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_u2i__assumptions_instance::_s_1_fire(IfMessage*)':
AAL2.C:27853: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_u2i__assumptions_instance::_s_1a_fire(IfMessage*)':
AAL2.C:27871: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_u2i__assertions_instance::_ne_1_fire(IfMessage*)':
AAL2.C:28120: warning: deprecated conversion from string constant to 'char*'
AAL2.C: In member function 'void if_u2i__assertions_instance::_ne_1a_fire(IfMessage*)':
AAL2.C:28137: warning: deprecated conversion from string constant to 'char*'
Compiled C++ code...
g++ -o AAL2.x AAL2.o /home/dragomir/xrc/libxerces-c.so -L/home/dragomir/if2/bin/x86_64 -lsimulator -lexplorator
Done.
[ahmad@topcased ~]$
```

Fig. 26. IF to executable file compilation.

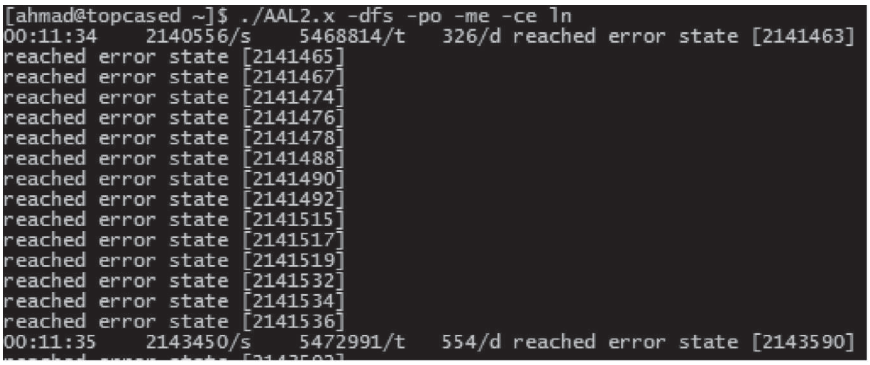

Fig. 27. Model checker results in error scenarios.

	Pros & Cons of our Approach	Reasons
+	Proof of concepts	The proposed approach is validated on two case studies
+	Tools	Tools were developed to Implement the proposed approach
+	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 research work was not part of an industrial project so we did not have the occasion to do some true empirical studies, we need to apply it on a real world case study to show the correctness of the transformation rules between Relax and SysML/Kaos concepts
-	No Demonstration of ROI	We provide no information regarding the Return On Investment by using our proposed approach
-	No adaptation Mechanism	We take into account the uncertainty in requirements of Self Adaptive Systems but we do not talk about the underlying adaptation mechanisms although we mention some techniques in the state of the art section to compare it with RELAX

Fig. 28. Pros and cons of our proposed approach.

case study, which will confront it to more rigorous and varied evaluation criteria such as its usability and its performance.

In order to validate the RELAX-ed requirement, we check whether the new expression of the property is acceptable or not. If it is acceptable then we proceed with the next step, in case if it is not acceptable, we propose two options: i.e., to cancel the RELAX-ation and go back to a *SHALL* invariant or complement the RELAX-ed property with an additional invariant (e.g., a *max* or *min* boundary that constraints the RELAX-ed expression). We would like to explore the validation step of the RELAX-ed requirement in more detail, so that to show how we can introduce the boundary values in the RELAX-ed expression.

We plan to investigate the adaptation mechanism techniques so that we can incorporate it in our proposed approach. Our approach takes into account the uncertainty in requirements of SAS, we model it using SYSML/KAOS and then we verify it but we do not talk about the underlying adaptation mechanisms.

For the time being, our RELAX2SYSML/KAOS tool is capable of mapping the RELAX concepts to SYSML/KAOS concepts but not the inverse. A natural follow up of our work is to investigate how we could make it a two-way process, 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 would help in taking into account the information modeled in SYSML/KAOS that we cannot capture in RELAX.

The verification of RELAX-ed requirements in our proposed approach is done using OMEGA2/IFX. To take into account the complexity of large systems, we can do the validation of their requirements at execution time. A promising approach to managing complexity in run-time environments is to develop adaptation mechanisms that leverage software models, referred to as Models@run.time (Blair et al., 2009). 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 (Aßmann et al., 2011).

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.

The use of model checking techniques used by OMEGA2/IFX exposes us to the problem of state space explosion 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 hope to tackle this problem using formal methods like B. There are already some works done for the mapping between SYSML/KAOS and B in this regard. In Laleau et al. (2010), a method is defined for bridging the gap between the requirements analysis level (Extended SysML) and the formal specification level (B). 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.

References

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

Ahmad, Manzoor, 2010. First step towards a domain specific language for self-adaptive systems. In: 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE'10). IEEE, pp. 285–290.

Ahmad, Manzoor, 2013. Modeling and Verification of Functional and Non Functional Requirements of Ambient, Self Adaptive Systems (Ph.D. thesis). Mathématique Informatique Télécommunications, University of Toulouse Mirail, France.

Ahmad, Manzoor, Araújo, João, Belloir, Nicolas, Laleau, Régine, Bruel, Jean-Michel, Gnaho, Christophe, Semmak, Farrida, 2013a. Self-adaptive systems requirements modelling: Four related approaches comparison. In: Comparing ∗Requirements∗ Modeling Approaches Workshop (CMA@RE) RE'13. IEEE Computer Society Press, Rio de Janeiro Brazil, pp. 37–42.

Ahmad, Manzoor, Bruel, Jean-Michel, 2014. A comparative study of RELAX and SysML/Kaos. Technical Report. Institut de Recherche en Informatique de Toulouse, University Toulouse II Le Mirail, France.

Ahmad, Manzoor, Bruel, Jean-Michel, Laleau, Régine, Gnaho, Christophe, 2012a. Modélisation des Exigences pour les Systmes Auto-adaptatifs: Intégration des Techniques Relax/SysML/Kaos. In: Journées GDR - GPL - CIEL http://gpl2012.irisa.fr/sites/default/files/CIEL2012-Ahmad-paper22/index.pdf.

Ahmad, Manzoor, Bruel, Jean-Michel, Laleau, Régine, Gnaho, Christophe, 2012b. 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, pp. 474–481.

Ahmad, Manzoor, Dragomir, Iulia, Bruel, Jean-Michel, Ober, Iulian, Belloir, Nicolas, 2013b. Early analysis of ambient systems sysml properties using omega2-ifx. In: 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH'13) SciTePress.

Apvrille, Ludovic, Courtiat, Jean P., Lohr, Christophe, de Saqui-Sannes, Pierre, 2004. TURTLE: A real-time UML profile supported by a formal validation toolkit. IEEE Trans. Softw. Eng. 30 (7), 473–487.

Aßmann, Uwe, Bencomo, Nelly, Cheng, Betty H.C., France, Robert B., 2011. Models@Run.Time (Dagstuhl Seminar 11481). DagstuhlReports 1 (11) http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=11481.

Baresi, Luciano, Pasquale, Liliana, Spoletini, Paola, 2010. Fuzzy goals for requirements-driven adaptation. In: Proceedings of the 2010 18th IEEE International Requirements Engineering Conference. IEEE Computer Society, Washington, DC, USA, pp. 125–134.

Bascans, Jérémy, Walczak, Jérémy, Zeghoudi, Jérôme, Ahmad, Manzoor, Geisel, Jacob, Bruel, Jean-Michel, 2013. COOL RELAX Editor, M2ICE Project, Université de Toulouse le Mirail.

Benghazi, Kawtar, Visitación Hurtado, María, Rodríguez, María Luisa, Noguera, Manuel, 2009. Applying formal verification techniques to ambient assisted living systems. In: OnTheMove Workshop (OTM '09). Springer-Verlag, Berlin/Heidelberg, pp. 381–390.

Blair, Gordon S., Bencomo, Nelly, France, Robert B., 2009. Models@ Run.Time. Computer 42 (10), 22–27.

Bozga, Marius, Graf, Susanne, Ober, Ileana, Ober, Iulian, Sifakis, Joseph, 2004. The IF toolset. In: Formal Methods for the Design of Real-Time Systems (FMDRTS '04). Springer-Verlag, Berlin/Heidelberg, pp. 237–267.

Cheng, Betty H.C., Sawyer, Pete, Bencomo, Nelly, Whittle, Jon, 2009a. 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). Springer-Verlag, Berlin/Heidelberg, pp. 468–483.

Cheng, Betty H.C., de Lemos, Rogério, Giese, Holger, Inverardi, Paola, Magee, Jeff, Andersson, Jesper, et al., 2009b. Software engineering for self-adaptive systems: A research roadmap. In: Software Engineering for Self-Adaptive Systems. Springer-Verlag, Berlin, Heidelberg, pp. 1–26.

Chung, Lawrence, Nixon, Brian A., Yu, Eric, Mylopoulos, John, 1999. Non-Functional Requirements in Software Engineering, 1st Springer-Verlag.

Clarke, Edmund M., Grumberg, Orna, Peled, Doron, 1999. Model Checking. MIT Press, London.

Clarke, Edmund M., Klieber, William, Novek, Milo, Zuliani, Paolo, 2012. Model checking and the state explosion problem. In: Meyer, Bertrand, Nordio, Martin (Eds.), Tools for Practical Software Verification. In: Lecture Notes in Computer Science, 7682. Springer-Verlag, Berlin Heidelberg, pp. 1–30.

Cleland-Huang, Jane, Settimi, Raffaella, Zou, Xuchang, Solc, Peter, 2007. Automated classification of non-functional requirements. Requir. Eng. 12 (2), 103–120.

Courtiat, Jean P., Santos, Celso A.S., Lohr, Christophe, Outtaj, B., 2000. Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. Comput. Commun. 23 (12), 1104–1123.

Cysneiros, Luiz Marcio, Leite, Julio Cesar Sampaio do Prado, 2004. Non functional requirements: From elicitation to conceptual models. IEEE Trans. Softw. Eng. 30, 328–350.

de Lemos, Rogério, Giese, Holger, Müller, A. Hausi, Shaw, Mary, Andersson, Jesper, Litoiu, Marin, et al., 2013. Software engineering for self-adaptive systems: A second research roadmap. In: de Lemos, R., Giese, H., Mller, H., Shaw, M. (Eds.), Software Engineering for Self-Adaptive Systems II. In: Lecture Notes in Computer Science, 7475. Springer-Verlag, Berlin Heidelberg, pp. 1–32. doi:10.1007/978-3-642-35813-5_1.

Dragomir, Iulia, Ober, Iulian, Lesens, David, 2012. A case study in formal system engineering with SysML. In: 17th International Conference on Engineering of Complex Computer Systems (ICECCS'12). IEEE, pp. 189–198.

Gnaho, Christophe, Semmak, Farida, 2010. Une Extension SysML pour l'ingénierie des Exigences Non-Fonctionnelles Orientée But. In: Ingénierie des Systèmes d'Information. Lavoisier Paris FRANCE, pp. 277–292.

Goldsby, Heather J., Sawyer, Pete, Bencomo, Nelly, Cheng, Betty H.C., Hughes, Danny, 2008. 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. IEEE Computer Society, Washington, DC, USA, pp. 36–45.

Kasten, Eric P., Sadjadi, Seyed M., McKinley, Philip K., 2003. Architecture and operation of an adaptable communication substrate. In: Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03) IEEE http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1204293&queryText=Architecture+and+operation+of+an+adaptable+communication+substrate&newsearch=true&searchField=Search_All.

Kephart, Jeffrey O., Chess, David M., 2003. The Vision of Autonomic Computing. Computer 36 (1).

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

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

Lapouchnian, Alexei, Liaskos, Sotirios, Mylopoulos, John, Yu, Yijun, 2005. Towards Requirements-Driven Autonomic Systems Design. In: Proceedings of the 2005 workshop on Design and evolution of autonomic application software. ACM, New York, NY, USA, pp. 1–7.

Lutz, Robyn R., 1993. Targeting safety-related errors during software requirements analysis. J. Syst. Softw. 34 (3), 223–230.

Moon, Seong ick, Lee, K.H., Lee, Doheon, 2004. Fuzzy branching temporal logic. IEEE Trans. Syst. Man Cybernet. B: Cybernet. 34 (2).

Nehmer, Jürgen, Becker, Martin, Karshmer, Arthur, Lamm, Rosemarie, 2006. Living assistance systems: An ambient intelligence approach. In: Proceedings of the 28th International Conference on Software Engineering (ICSE'06). ACM, pp. 43–50.

Ober, Iulian, Dragomir, Iulia, 2010. OMEGA2: A new version of the profile and the tools. In: 15th International Conference on Engineering of Complex Computer Systems (ICECCS '10). IEEE, pp. 373–378.

Ramirez, Andres J., Cheng, Betty H.C., Bencomo, Nelly, Sawyer, Pete, 2012a. Relaxing claims: Coping with uncertainty while evaluating assumptions at run time. In: France, Robert B., Kazmeier, Jrgen, Breu, Ruth, Atkinson, Colin (Eds.), Model Driven Engineering Languages and Systems. In: Lecture Notes in Computer Science, 7590. Springer-Verlag, Berlin/Heidelberg, pp. 53–69.

Ramirez, Andres J., Fredericks, Erik M., Jensen, Adam C., Cheng, Betty H.C., 2012b. Automatically RELAXing a goal model to cope with uncertainty. In: Proceedings of the 4th International conference on Search Based Software Engineering. Springer-Verlag, Berlin/Heidelberg, pp. 198–212.

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

Vítor, E., Souza, S., Lapouchnian, Alexei, Robinson, William N., Mylopoulos, John, 2011. Awareness requirements for adaptive systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. ACM, New York, NY, USA, pp. 60–69.

Welsh, Kristopher, Sawyer, Pete, 2010. Understanding the scope of uncertainty in dynamically adaptive systems. In: Requirements Engineering: Foundation for Software Quality. Springer-Verlag, Berlin Heidelberg, pp. 2–16.

Welsh, Kristopher, Sawyer, Pete, Bencomo, Nelly, 2011. 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, pp. 560–563.

Whittle, Jon, Sawyer, Pete, Bencomo, Nelly, Cheng, Betty H.C., 2008. A language for self-adaptive system requirements. In: International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER'08) IEEE http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4797489&filter%3DAND%28p_IS_Number%3A4797485%29.

Whittle, Jon, Sawyer, Pete, Bencomo, Nelly, Cheng, Betty H.C., Bruel, J.-M., 2009. RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE. IEEE Computer Society, Washington, DC, USA, pp. 79–88.

Yu, Eric S.K., 1997. Towards modeling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering. IEEE Computer Society, pp. 226–235.

Yu, Yijun, Lapouchnian, Alexei, Liaskos, Sotirios, Mylopoulos, John, Leite, Julio C.S.P., 2008. From goals to high-variability software design. In: Proceedings of the 17th International Conference on Foundations of Intelligent Systems. Springer-Verlag, Berlin, Heidelberg, pp. 1–16.

Yu, Yijun, Leite, Julio C.S.P., Mylopoulos, John, 2004. From goals to aspects: Discovering Aspects from requirements goal models. In: Proceedings of the 12th IEEE International Requirements Engineering Conference. IEEE Computer Society, Washington, DC, USA, pp. 38–47.

Manzoor Ahmad received his Ph.D. from the University of Toulouse Mirail in October 2013. He is working as research/teacher assistant at the University of Pau for the academic year 2013-2014. Currently member of the LIUPPA (Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour). Previous Member of the MACAO team (Modèles, Aspects, Composants pour des Architectures à Objets) of the IRIT (Institut de Recherche en Informatique de Toulouse) CNRS laboratory. His research areas include requirements engineering, model driven engineering, goal based requirements engineering, computer networks and use of formal methods for the properties verification of self-adaptive systems.

Nicolas Belloir is an associate professor at the computer department of the University of Pau, France, since 2005. It is member of the MOVIES research team at the LIUPPA (Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour). He received a Ms. D. in computer science from the University of Toulouse in 1999. He worked as software engineer between 1999 and 2001 in Transiciel Company on embedded and real-time systems. He received its Ph. D. in computer science from the University of Pau in 2004. His research interest deals with semi-formal modeling language (UML, SysML, DSML …), model driven engineering, requirement engineering, and component-based software engineering.

Jean-Michel Bruel received his Ph.D. from the University Paul Sabatier (Toulouse) in December 1996. From September 1997 to August 2008, he was associate professor at the University of Pau. Member of the LIUPPA (Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour) from 2000 to 2008. Currently member of the MACAO team (Modèles, Aspects, Composants pour des Architectures à Objets) of the IRIT (Institut de Recherche en Informatique de Toulouse) CNRS laboratory. His research areas include development of distributed, component-based applications, methods integration, and on the use of formal methods in the Component-Based Software Engineering context. He has defended his "Habilitation à Diriger des Recherches" in December 2006 and obtained in 2008 a full professor position at the University of Toulouse. He has been head of the Computer Science department of the Technical Institute of Blagnac from 2009 to 2012.