PhD Dissertation

International Doctorate School in Information & Communication Technologies

DISI - University of Trento

REQUIREMENTS ENGINEERING FOR SELF-ADAPTIVE SOFTWARE: BRIDGING THE GAP BETWEEN DESIGN-TIME AND RUN-TIME

Nauman Ahmed Qureshi

Advisor:

Prof. Anna Perini

Fondazione Bruno Kessler, Trento

November 2011

[To my family]

Acknowledgments

First of all I am extremely thankful to my supervisor Prof. Anna Perini for her kind guidance, advice and support that kept my spirits and gallantry high during my PhD research. I am honored to be her PhD student. Thank you!!

I also wish to acknowledge and position on record the deep sagacity of gratitude to Prof. John Mylopoulos, Prof. Carlo Ghezzi, Prof. Franco Zambonelli and Dr. Nelly Bencomo for accepting to be in my thesis committee, evaluating my work and providing valuable suggestions and comments.

Special thanks are due to Prof. John Mylopolous for providing me the opportunity to visit his research group at University of Tronto, Canada. During this visit I came across several colleagues to whom I would like to thank especially Neil A. Ernst, Daniele Barone for useful discussions and support during my visit.

I would like to thank Dr. Angelo Susi, Dr. Ivan Jureta, Dr. Norbert Seyff, Prof. Sotirios Liaskos and Dr. Alessandro Marchetto for their valuable suggestions on my work and collaborating with me during my PhD research work. You all have been a great source of inspiration.

I also would like to thank all my fellow colleagues and great friends especially Cu D. Nguyen, Chiara Di Francescomarino, Mirko Morandini, Alberto Siena, Andrea Avancini in the software engineering research group at FBK and many other friends and colleagues Adbul Qader, Tahir, Komminist, Gianluca, Nevena, Lucilla, Marc, Itzel, Sepideh and Raian who kept my spirits high, their suggestions & support provided me the courage to carry out my research.

Last but not the least, I would like to thank my family especially my parents, brothers and sister who have been a great source of motivation for me, their constant encouragement, prayers and devotion towards me and my studies enabled me throughout my PhD research. Thank You!!.

Nauman Ahmed Qureshi

Abstract

Self-Adaptive Software systems (SAS) adapt at run-time in response to changes in user's needs, operating contexts, and resource availability, by requiring minimal to no involvement of system administrators. The ever-greater reliance on software with qualities such as flexibility and easy integrability, and the associated increase of design and maintenance effort, is raising the interest towards research on SAS.

Taking the perspective of Requirements Engineering (RE), we investigate in this thesis how RE for SAS departs from more conventional RE for non-adaptive systems.

The thesis has two objectives. First, to define a systematic approach to support the analyst to engineer requirements for SAS at design-time, which starts at early requirements (elicitation and analysis) and ends with the specification of the system, which will satisfy those requirements. Second, to realize software holding a representation of its requirements at run-time, thus enabling run-time adaptation in a user-oriented, goal-driven manner.

To fulfill the first objective, a conceptual and theoretical framework is proposed. The framework is founded on core ontology for RE with revised elements that are needed to support RE for SAS. On this basis, a practical and systematic methodology at support of the requirements engineer is defined. It exploits a new aggregate type of requirement, called *adaptive requirements*, together with a visual modeling language to code requirements into a design-time artifact (called *Adaptive Requirements Modeling Language, ARML*). Adaptive requirements not only encompass functional and non-functional requirements but also specify properties for control loop functionalities such as monitoring specification, decision criteria and adaptation actions. An experiment is conducted involving human subjects to provide a first assessment on the effectiveness of proposed modeling concepts and approach.

To support the second objective, a *Continuous Adaptive RE (CARE)* framework is proposed. It is based on a service-oriented architecture mainly adopting concepts from service-based applications to support run-time analysis and refinement of requirements by the system itself. The key contribution in achieving this objective is enabling the CARE framework to involve the end-user in the adaptation at run-time, when needed. As a validation of this framework, we perform a research case study by developing a proof of concept application, which rests on CARE's conceptual architecture.

This thesis contributes to the research on requirements engineering for SAS by proposing:

1. a conceptual core ontology with necessary concepts and relations to support the formulation of a dynamic RE problem i.e. finding adaptive requirements specification both at design-time and run-time.

2. a systematic methodology to support the analyst for modeling and operationalizing adaptive requirements at design-time.

3. a framework to perform continuous requirements engineering at run-time by the system itself involving the end-user.

Keywords *Requirements Engineering (RE), Self-Adaptive Software (SAS), Goal-Oriented Requirements Engineering (GORE), Service-Based Applications (SBA)*

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Contemporary Internet-enabled software applications are increasingly immersed in the fabric of our daily life. This surge in using software by the end-users with volatile needs has increased the computational complexity and maintenance problem for such software systems. The increasing reliance on software has led to growing interest in research on *Self-Adaptive Software* systems (SAS) [CGI+08], which have been proposed as a possible solution to overcome such problems.

1.1 Motivation and Problem

Following recent research agendas [CGI+08, DNGM+08, ST09, SBW+10], in this thesis, we define *self-adaptive software systems* as: *software systems that adapt their behavior at run-time in response to changing user's requirements, operating contexts, and resource availability.*

Engineering of *self-adaptive software* is challenging and influences all the phases of software development ranging from requirements engineering, to design, implementation and maintenance. Ongoing research on software engineering for *self-adaptive software* proposes novel design-time methods and techniques to support run-time adaptation mainly exploiting architecture based on MAPE (Monitor, Analyze, Plan, Execute) control loop [KC03]. Common to these approaches is the adoption of component-based [KM07], middleware-oriented [HFS04], service-oriented [DNGM$^+$08], agent-oriented [TCW$^+$04] designs to develop *self-adaptive software systems* while anticipating run-time changes. On the other hand, Requirements Engineering (RE) for *self-adaptive software* is still in its fledgling stages. In RE literature, requirements-driven approaches have motivated the need to monitor system's requirements to validate their compliance at run-time [FF95]. However, in case of *self-adaptive software systems*, RE activities need to be performed at various levels of abstractions, as it is envisioned in [BCZ05].

Along this vision, recent approaches to engineer *self-adaptive software* advocate the role of the requirements along the later stages of the software development to support run-time adaptation [MPP08, DGM09]. Unfortunately, in these approaches it is difficult to accommodate new or changed requirements. In reality, change is inevitable. End-user requirements change with respect to the dynamic environment in which applications are executed and used. RE activities are needed to be performed at design-time with more explicit constructs to specify requirements for SAS, but are also needed at run-time. Existing approaches are limited to cope with this. This manifests a gap between design-time and run-time requirements engineering.

1.2 Research Objectives

In this thesis, we investigate how RE for SAS departs from more conventional RE for non-adaptive systems, focusing on two concrete objectives:

Obj1 To define a systematic methodology at support of the system analyst to engineer the requirements of SAS at design-time, which starts at early requirements (elicitation and analysis) and ends with the specification of the system satisfying the requirements.

Obj2 To enable software holding a representation of its requirements at run-time, and support goal- and user-oriented adaptation at run-time.

We elaborate the above objectives with respect to the following research questions:

RQ1 What concepts and abstractions are needed to formulate the "requirements problem" for SAS? [Obj1]

RQ2 How to systematically engineer requirements for SAS at design-time? [Obj1]

RQ3 How to enable SAS to perform RE at run-time and support goal- and user-oriented adaptation? [Obj2]

To support the discussion at best, a travel domain case study is exploited. *Travel Companion* is a software for travel planning and monitoring, which is an adaptive service-based application. It is able to adapt with respect to changes that occur in the end-user needs, operating context and availability of resources. Other exemplar case studies have been considered during the research, mainly representative of service-based applications e.g. adaptive anti-virus, adaptive Personal Media Server (PMS), adaptive meeting scheduler, service-based laundry collection scenario (Intentional Interoperability), adaptive GPS navigation software, and cleaner agent exemplar case study.

In this thesis description we focus on *Travel Companion* since it shows the main adaptivity properties such as change in context conditions, changes in end-user needs, availability of resources and of solutions (i.e. available services). Below, we give an overview of the case study.

1.3 Case Study: Travel Companion

Travel Companion is a service-based application that is aware of end-users' goals and preferences and by monitoring the usage context (location, device and operating environment) it tries to help them accomplishing their goals. New requirements emerge and that can be operationalized with respect to existing or available services. It relies on third party services that are assumed to be available over the Internet. It provides access to end-users' to get on the fly information as they require through available services (e.g. weather forecast). It also allows them to specify their travels and meeting schedules.

Moreover, Travel Companion manages end-users profile and respective preferences for a particular travel itinerary. It monitors the flight schedules and notifies the end-users' about their travel updates (e.g. flight status changes), meeting reminders (e.g. when, and where meeting is) and in case of situations where the goals and preferences of the end-users' are not met, it looks for alternative solutions (e.g. flight is canceled, look for trains, or rent a car) and tries to involve the end-users' by asking their relevant feedback. It tries to keep track of the available resources (e.g. availability of the Internet, device battery level, social contacts and agenda etc.). It searches for alternatives to operationalize the task (e.g. completing services) that may requires a particular resource (e.g. contacts are needed to inform in case of emergency, agenda is needed to reschedule meetings).

In addition, it provides relevant information (e.g. news, weather updates) that are necessary to keep Travel Companion informed about the external events. At run-time it can act as an analyst by acquiring user input about new or changed requirements and search for available services as alternative ways to operationalize these requirements involving the user. In case of goals and preferences that it is not able to meet, it creates a log for the off line evolution. This case study is exploited to derive proof of concepts on solutions elaborated for the two problems below.

Adaptation at Design-time. Adaptive requirements are elicited and requirements problem is formulated and adapted by finding the adaptive requirements specification. Along this systematic process, monitoring specification and decision criteria are elaborated with the space of alternative tasks that are needed to enable run-time adaptation.

Adaptation at Run-time, which is performed at various level of details. Key facets that trigger adaptation are: contextual information of the end-user and/or the system, resource availability, relevant service availability, and new or emerging needs that an end-user can specify.

1.4 Approach Overview and Contribution

The approach adopted to address the identified objectives, consists in revisiting core concepts, which were introduced in well known foundational work, for defining the software requirements problem. In addition, we propose systematic methods and proof of concept tools to realize solutions solving the problem of requirements engineering for *self-adaptive software systems*, both at design-time and run-time.

- We start with establishing the theoretical and conceptual basis. The idea is that some elements in the definition of the requirements problem for *self-adaptive software systems*, expressed along the classical requirements problem formulation by Zave & Jackson in [ZJ97b] and its recent reformulation [JMF08] at design-time, may change at run-time. This demands for a new requirements problem formulation and a corresponding set of candidate solutions. We propose a formulation of the requirements problem for *self-adaptive software systems* as a dynamic problem. This dynamic requirements problem definition is supported by a revised core ontology for requirements engineering, which includes new concepts and relations.

- To put these concepts into practical use, we propose a new composite requirement type, called *Adaptive Requirements*, that can help analysts to understand requirements for *self-adaptive software systems*. Such requirements not only encompass functional and non-functional requirements but also specify properties for control loop functionalities such as monitoring specification, decision criteria and adaptation actions. We exploit the core ontology to perform requirements engineering for *self-adaptive software systems* and propose a modeling language, called Adaptive Requirements Modeling Language (ARML), along with the guidelines to perform early requirements engineering for *self-adaptive software systems*. Following these guidelines, an analyst at design-time can capture and analyze requirements for the target self-adaptive system thereby formulating the requirements problem and finding the adaptive requirements specification.

- This specification provides input to a framework for performing requirements engineering at run-time. We call it Continuous Adaptive Requirements Engineering (CARE) framework. The framework provides a systematic way to continuously refine requirements specification that is made available to the system. At run-time, the system instantiating CARE acts as an analyst and exploits information gathered through monitoring or explicitly given by the end-user, for deciding and selecting a candidate solution (e.g. using available services) to resolve the requirements problem by itself.

Preliminary evaluations of the proposed ideas are provided. A proof of concept tool for supporting the operationalization of adaptive requirements at design-time has been realized. This prototype tool has been applied to an exemplar case study to evaluate the system performance. A prototype application has been developed that instantiates CARE framework at run-time involving the end-user. An empirical evaluation on the effectiveness of the proposed ARML, through an empirical survey with human subjects, is also presented.

In summary, this thesis contributes to the research in requirements engineering for SAS by proposing:

1. a conceptual core ontology with necessary concepts and relations to support the formulation of a dynamic requirements problem i.e. finding adaptive requirements specification both at design-time and run-time.
2. a systematic approach to support the analyst for modeling and operationalizing adaptive requirement at design-time.
3. a framework to perform continuous requirements engineering at run-time by the system itself involving the end-user.

1.5 Structure

The thesis is organized in chapters with their corresponding dependency links as shown in the Figure 1.1. The aspects covered in each chapter are briefly recalled here below.

Chapter 2, presents the state of the art requirements engineering and design approaches for developing *self-adaptive software systems*. In particular, we distinguish architecture centric approaches including service-oriented, agent-oriented and RE approaches including goal-oriented methods, requirements monitoring and more recent works on RE for *self-adaptive software systems*.

Chapter 3, defines the conceptual and theoretical framework in which we propose a formulation of the requirements problem for *self-adaptive software systems* as a *dynamic RE problem* that a *self-adaptive software* should be engineered to solve. At run-time, a *self-adaptive software* may move from one requirements problem to another thereby finding solutions to satisfy the requirements. To support such a dynamic requirement problem formulation for *self-adaptive software systems*, we revise the core ontology for RE and define new concepts (such as context and resource) and relations (such as

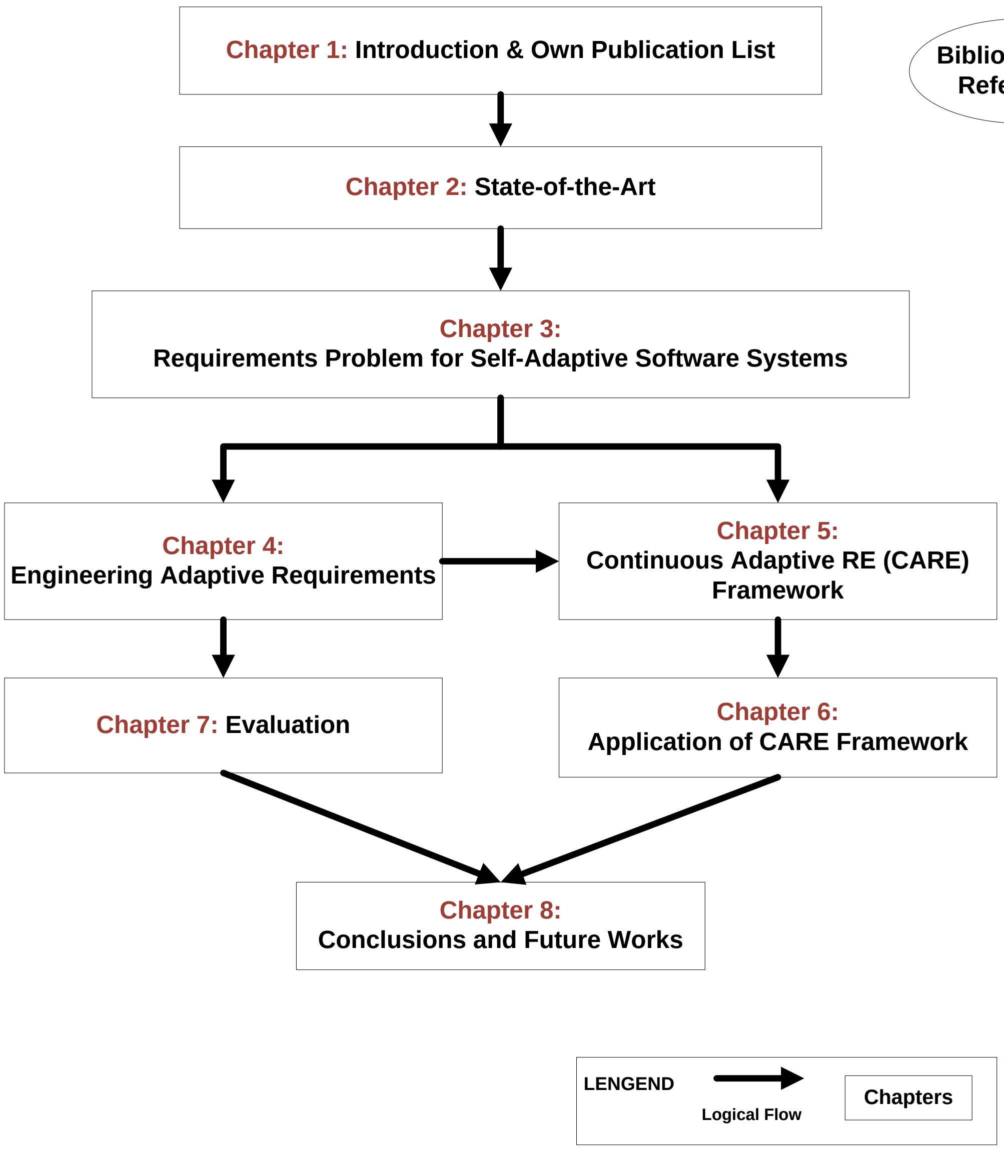

Chapter 1: Introduction & Own Publication List
Chapter 2: State-of-the-Art
Chapter 3: Requirements Problem for Self-Adaptive Software Systems
Chapter 4: Engineering Adaptive Requirements
Chapter 5: Continuous Adaptive RE (CARE) Framework
Chapter 7: Evaluation
Chapter 6: Application of CARE Framework
Chapter 8: Conclusions and Future Works
Bibliography & References
LENGEND
Logical Flow
Chapters

Figure 1.1: Thesis outline

relegate and influence) to support dynamic problem formulation.

Chapter 4, introduces the Continuous Adaptive RE (CARE) framework, that supports continuous RE at run-time involving the end-user. To provide a clear distinction between RE at design-time and RE at run-time, adaptation types are defined in light of CARE, revisiting the four levels of RE proposed by Berry et al. [BCZ05]. Concepts and artifacts needed by the *self-adaptive software* itself at run-time to perform continuous RE involving the end-user are introduced. A conceptual architecture of an application is proposed, which instantiates CARE at run-time.

Chapter 5, defines the concept of Adaptive Requirements for *self-adaptive software systems*, which are requirements that make explicit the feedback loop functions such as monitoring specification, decision criteria and alternative adaptation tasks. To elicit and specify the adaptive requirements, a systematic approach is proposed with guidelines to support the analyst/designer at design-time. In this approach, modeling adaptive requirements helps in identifying monitoring specification that are operationalized using a monitoring framework to validate their usability at run-time, respectively. Exploiting the concepts and relations proposed in the revised core ontology of RE for *self-adaptive software systems*, to formulate the requirements problem at design-time, this chapter proposes visual notations to model adaptive requirements for SAS. An Adaptive Requirements Modeling Language (ARML) for *self-adaptive software systems* is proposed, which is based on a recently proposed abstract requirements modeling language, Techne [JBEM10b]. ARML provides basis for modeling the requirements problem with convenient visual notations.

Chapter 6, illustrates the application of CARE framework to a travel case study introduced earlier. Continuous RE is supported by the proof of concept application that instantiates CARE involving the end-user.

Chapter 7, provides initial evaluation along three aspects. First, it provides preliminary results about measuring scalability and performance of the design-time tool. Second, a qualitative assessment on modeling requirements for *self-adaptive software* is presented. Third, an empirical survey study with subjects is conducted to evaluate the effectiveness of the modeling concepts, visual notations and modeling guidelines for ARML.

Chapter 8, presents the conclusions and summary of contributions. Future research directions which are identified during this thesis work are presented. On some of the ideas initial exploration has been conducted.

1.6 List of Published Papers

Most parts of this thesis have been published in international conferences and workshops. Below, is a list of papers, which is classified as: published along the years, submitted and in preparations or future work.

List of Published Papers

2011

- Nauman A. Qureshi, Ivan Jureta, Anna Perini. *Towards a Requirements Modeling Language for Self-Adaptive Systems.* in *18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'12) [Accepted for publication].*

- Marc Oriol, Nauman A. Qureshi, Xavier Franch, Anna Perini, Jordi Marco. *Requirements Monitoring for Adaptive Service-Based Applications* in *18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'12) [Accepted for publication].*

- Nauman A. Qureshi, Ivan Jureta, and Anna Perini, "Requirements engineering for self-adaptive systems: Core ontology and problem statement", in *23rd International Conference on Advanced Information Systems Engineering (CAiSE'11)*, pp. 33-47, ser. LNCS, vol. 6741. Springer, 2011.

- Nauman A. Qureshi, Norbert Seyff, Anna Perini, "Satisfying User Needs at the Right Time and in the Right Place: A Research Preview", in *17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'11)*, pp. 94-99, vol.6606, Springer (LNCS), March 2011.

- Nauman A. Qureshi, Sotirios Liaskos, Anna Perini, “Reasoning About Adaptive Requirements for Self-Adaptive Systems at Runtime”, in *Second International Workshop on Requirements@Runtime (RRT’11) at RE’11*, pp. 16-22, August 2011.

- Anna Perini, Nauman A. Qureshi, Luca Sabatucci, Alberto Siena, Angelo Susi, “Evolving Requirements in Socio-Technical Systems: Concepts and Practice”, in *30th International Conference on Conceptual Modeling (ER 2011)*, pp. 440-447, LNCS, Springer, Oct. 2011.

2010

- Nauman A. Qureshi, Anna Perini, “Requirements Engineering for Adaptive Service Based Applications”, in *18th IEEE International Requirements Engineering Conference (RE’10)*, pp. 108-111, September 2010.

- Nauman A. Qureshi, Anna Perini, Neil A. Ernst, John Mylopoulos, “Towards a continuous requirements engineering framework for self-adaptive systems”, in *First International Workshop on Requirements@Runtime (RRT’10) at RE’10*, pp. 9-16, September 2010.

- Nauman A. Qureshi, Anna Perini, “Continuous adaptive requirements engineering: An architecture for self-adaptive service-based applications”, in *First International Workshop on Requirements@Runtime (RRT’10) at RE’10*, pp. 17-24, September 2010.

- Nauman A. Qureshi, Cu D Nguyen, Anna Perini, “Analyzing Interoperability Requirements for Adaptive Service-based Applications”, in *Fourth IEEE International Workshop on Requirements Engineering for Services (REFS’10) at COMSPAC’10*, pp. 239-244, July, 2010.

- Alessandro Marchetto, Cu D. Nguyen, Chiara Di Francescomarino, Nauman A. Qureshi, Anna Perini, Paolo Tonella, “A Design Methodology for Real Services”. in *Second International Workshop on Principles of Engineering Service-Oriented Systems (PESOS’10) at ICSE’10*, pp. 15-21, May 2010.

- Chiara Di Francescomarino, Chiara Leonardi, Alessandro Marchetto, Cu D Nguyen, Nauman A Qureshi, Luca Sabatucci, Anna Perini, Angelo Susi, Paolo Tonella, Massimo Zancanaro, “A bit of Persona, a bit of Goal, a bit of Process ... a recipe for Analyzing User Intensive Software System”, in *Fourth International i* Workshop (iStar2010) at CAiSE’10*, pp. 36-40, June 2010.

2009

- Nauman A. Qureshi, Anna Perini, “Engineering Adaptive Requirements”, in *Workshop Software Engineering for Adaptive and Self-Managing Systems (SEAMS’09) at ICSE ’09*, pp. 126-131, May 2009.

2008

- Nauman A. Qureshi, Anna Perini, "Towards Seamless Adaptation: An Agent-Oriented Approach", in *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO '08)*, pp.471-472, 20-24 Oct. 2008.
- Nauman A. Qureshi, Anna Perini, "An Agent-Based Middleware for Adaptive Systems", in *Eighth International Conference on Quality Software (QSIC '08)*, pp.423-428, 12-13 Aug. 2008.

Chapter 2

State of the Art

2.1 Overview

The topic of *self-adaptive software systems* is calling for investigation in several research areas along many facets of software engineering as stated in Cheng et al. [CdLG^{+}09]. Many open issues have been identified and acknowledged by revisiting the current efforts in the field of software engineering (in particular requirements engineering, analysis and modeling, design, engineering and assurance). Currently, the emphasis of research is more on design-time solutions to provide run-time adaptation. Despite ongoing research, requirements engineering for *self-adaptive software systems* has received less attention so far[1]. This results in a gap between design-time and run-time adaptation activities. Self-adaptive software systems need to be aware of their own requirements and able to monitor their end-user's goals and preferences, domain conditions, operating context, available resources and the multiple tasks they are able to perform at run-time. However, engineering such systems is challenging.

In this chapter, we recall state of the art works from software engineering areas where our research develops, namely, architecture-based design approaches (Section 2.2) and requirements engineering approaches (Section 2.3).

[1]This was even more remarkable at the time this thesis started.

For the sake of ease in reading, we categorize design approaches along middleware or service-oriented or software agents based approaches, while for requirements engineering works we distinguish goal-oriented, variability modeling, formal and informal approaches for self-adaptive software.

2.2 Design Approaches for Self-Adaptive Software Systems

So far, considerable work in realizing *self-adaptive software* is architecture driven and based on middleware approaches. Architecture based approaches abstract the adaptation decision from code level to component level supporting if there is consistency between the code and the architectural model. The main idea is to address adaptation needs at design-time defining a set of different architectures, each one appropriate for specific environment conditions.

2.2.1 Component-Based Approaches

In [OGT$^+$99], two complementary processes are introduced to enable *self-adaptive software*, namely, adaptation management i.e. for changing the application over time upon monitoring, evaluating, planning; and evolution management, which focuses on the mechanisms that are used to change the application through system artifacts, also taking into account consistency issue. These two processes can have humans in the loop, or they can be automated, through the use of software agents.

Run-time and design-time adaptation are considered in RAINBOW by Garlan et. al. [CHG$^+$04]. The main idea behind run-time adaptation in RAINBOW[2] is to provide generic externalized adaptation mechanisms based upon a monitoring-reasoning-adapting loop, using probes and gauges. As for design-time adaptation, the authors highlight architecture based adaptation challenges like accurate decompositions and issue of latency at run-time.

[2]http://www.cs.cmu.edu/ able/research/rainbow/

Furthermore, it is well argued in [KM07] that the architectural approaches are promising for developing self-managing systems. This is due to the generality and the level of abstractions, which helps in satisfying the challenges posed by self-* systems. Kramer et al. has discussed their component based reference architecture based on three layers namely - component control layer, change management layer and goal management layer. Subsequently, dealing with challenges like reconfiguration, ensuring application consistency and avoiding undesirable transient behaviors at component layer. At change management layer, a fault-tolerant state management and decentralized configuration management is tackled. Lastly, at higher-level, constraint based planning activity is carried out by aligning "Online" the goals and of the system. This approach provides suitable arguments for architecture based approach but require more comprehensive approaches to be integrated in order to sort out critical system properties. This will help in dealing with the emergent behaviors.

2.2.2 Middleware-Based Approaches

In [HSSF06] product line engineering techniques are used to mask the user and resource variability in an ubiquitous environment by proposing a platform based on MADAM middleware. Its further extension is MUSIC[3], which is an open source platform for building adaptive mobile applications. In this approach a prototype platform is developed as a collection of components that supports a generic adaptation mechanism. However, this approach seems to be simple in terms of adaptation and needs to be improved in terms of performance.

In [HFS04], FAMOUS[4] - an adaptation middleware is used as component framework to dynamically select an architecture variant for applications

[3]http://ist-music.berlios.de/site/platform.html

[4]FAMOUS is a strategic research project founded by the Research Council of Norway. The goal of FAMOUS is to create a framework for building adaptive mobile ubiquitous services.

to adapt in the context of mobile environments by exploiting the reflective architecture, which is a run-time representation of its application framework.

Similar approach is pursued in [GEL$^+$06] but the focus was on using a middleware to manage self-adaptivity by mirror-based reflections and using a QuA[5] middleware to support services by behaviors. In this, a video streaming example was demonstrated supported by QuA middleware for observing QoS parameters using goal-oriented and utility function approaches to quantify the suitability about adaptation. The main idea was to separate the adaptation concerns from the application concerns.

In [FHS$^+$06] emphasis was on separation of adaptation concerns from application concerns by using architectural models generated at run-time leaving the decision on middleware (MADAM) to reason. The application variants and their properties are addressed in terms of coarse-grained and fine-grained variability. Moreover, the focus is on application of architecture models at run-time providing more consistency. The main idea is on separate the adaptation concerns from the application concerns as advocated by D. Garlan et al. [CHG$^+$04].

Most of the work analyzed so far is mainly inspired by middleware approach using architectural models inherently based on closed-loop control. This concept of separating the adaptation concerns from the application concerns seems promising in terms of reusability. The approaches discussed so far, abstract the adaptation decision from code level to component level providing the specification of a set of alternative architectures that better fit with quality factors, and take into account environment conditions, at design-time. However, the link between architecture and requirements for *self-adaptive software* is not fully addressed. Moreover, managing the continuously changing needs of end-users is limited in these works, which makes these approaches more strict and system architect dependent. Whereas, run-time adaptation requires

[5]http://simula.no/research/networks/projects/QuA

more flexible and requirements aware approach.

2.2.3 Agent-Oriented Approaches

Architecture-based approaches employ agents as their core component. According to Perini in [Per09] agent-based approaches are gaining much popularity as due to their natural compatibility for satisfying the visions of autonomic computing by Kephart et al. in [KC03]. Several agent-oriented methodologies (e.g. Tropos [BPG$^+$04] and Gaia [ZJW03]) has been proposed providing abstractions to engineer distributed systems, where an agent can be viewed as a computational system situated in an environment, capable of performing autonomous actions and presenting flexible behaviors to fulfill the goals for which it has been designed.

To achieve its intended purpose, it requires some infrastructural support for the provisioning of its services, called Multi-Agent Systems (MAS), as a flourishing software engineering paradigm to develop distributed systems. In addition, they can be employed to provide flexible infrastructure i.e. to automate the process-oriented tasks, by Oreizy et al. in [OGT$^+$99].

A well grounded discussion can be seen on the adoption of seamless agent middleware technology as part of large software systems by Omicini et al. in [OR04]. This integration of *MAS-as-subsystem* is discussed in light with respect to three dimensions of technology transfer by investigating JADE agent platform. This stimulates the use of agents to provide flexibility not at infrastructure level but also at conceptual level. In [SR07], the suitability of agents has been the key focus, because agents are used as a core artifact to model system dynamics. The approach entails a requirements driven approach by expressing adaptivity and to develop self-organizing MAS.

Tesauro et al. in [TCW$^+$04] discuss the IBM's Unity platform for autonomic computing as an example of how MAS can be used in practice for enabling self-* properties. This architectural model adopts software agents implemented in Java, using the Autonomic Manager Toolkit2. The proposed infrastructure is a composition of agents as autonomic elements including policy elements (policies to be applied), sentinel elements (ensure monitoring) and system elements (ensuring resources), supported by autonomic managers for predicting changes. An exciting part of this work is its application to perform goal-driven self-assembly, that is a self-configuring autonomic element by knowing its high-level goals can contact the registry to get required services by others to fulfill its goals.

Following the AOSE perspective, [PPSM07a] uses the *TROPOS* methodology to model variability in user's needs and preferences, in terms of alternatives for goal achievement at requirements-time. The modeled variability reflects into the design and coding of Belief-Desire-Intention (BDI) agents, which are able to switch from a behavior to a most appropriate one, depending on the perceived environment conditions (including users preferences) at run-time. In this work, goal alternatives represented in the requirements models map to software agents goal models in the design and code artifacts. However, the intentional abstractions are very much close to the agent-oriented approaches to design *self-adaptive software* [MPP08], but at run-time it is not evident that such approaches are adequate enough for managing changes in the requirements during the adaptation process.

2.2.4 Service-Oriented Approaches

Service-oriented architectures (SOA) are based over the concept of decoupled services as their functional unit. SOA as another form of software architecture providing much better facilities for business concerns to abstract the users (consumers) from the internal implementation. The idea of using services in the context of *self-adaptive software* seems promising.

In [DPT07] highlights the use of service-oriented concepts in designing a *self-adaptive software*. The idea rests on the combination of traditional control loop for adaptation and service invocation. They highlights the importance of separation of adaptation, fault taxonomy and standard libraries for loosely coupled application and adaptation modules.

In [IMW07], the authors promote the use of dynamic aspect oriented programming (d-AOP) in SOA platform in order to reuse the existing services and adapting it to the new requirements. The main idea is to leveraging the “Hot-Deployment” feature of the existing Open Gateway Initiative (OSGi) framework published by Eclipse foundation and by integrating the d-AOP to realize the dynamic run-time adaptation by deploying aspects as bundles to provide seamless integration of both by supporting the claim for run-time adaptation.

Analogously, in the context of service engineering, Papazoglou [Pap08] proposes a theoretical approach for structural changes of services, which focuses on service compatibility, compliance, and conformance. This approach builds upon a detailed analysis of the possible sources of change.

This work inspires our research that focuses on the more specific (although ambitious) objective concerning software evolution that requires minimal human intervention. Moreover, in our approach we tend to follow an artifact-centric approach, similar to that proposed in [BGH$^+$07].

Hence, it seems evident that the choice of using services as the motivating artifact for the intended *self-adaptive software*. Based on the concepts like separation of concern, the services themselves are not able to reason about “What, Why and How”. Therefore, they need an explicit mechanism like agents to automate their use at run-time in order to overcome the user’s needs.

Architecture driven approaches provide good abstraction by answering *WHAT to monitor*, *HOW to evaluate* and *WHEN to adapt*. These design-time decisions enable run-time adaptation. Taking the perspective of requirements engineering, these approaches treat requirements at later stages, where the key artifact is the application component, agent, service description or a program code. This treatment leads to strict rules and functions that can aid adaptation as reconfiguration of components, services or agent behaviors. The core issue remains open i.e. how to manage changing requirements at run-time.

2.2.5 Self-Adaptive and Self-Organizing Systems (SASO)

To pursue the goal of building autonomic systems, a unified view of the system is well articulated in terms of a generic framework in [SFRG08]. The presented framework supports both *self-adaptive software* and self-organizing characteristics promoting that a system not only has hierarchical top-down view - as in case of *self-adaptive software* but also is decentralized (in terms of variety of components) by having a bottom-up view - as in case of self-organizing systems. Design-time techniques and architecture supporting run-time infrastructures are the main focus in this work, hence by enabling a system with *self-adaptive software* and self-organizing capabilities. Moreover, supporting the need to have reliable and controllable SASO system, the emergent behavior by respecting the dependable properties such as trustworthiness in case of security, safety and performance. In contrast to traditional approach for building resilient and predictable systems, some engineering requirements are presented. A generic framework is also articulated, which supports the development of trustworthy, resilient, controllable and self-reconfigurable SASO systems.

However, argued framework [SFRG08] is under ongoing investigation by evaluating it on variety of case studies but still it provides some engineering guidelines for building SASO systems. This exploits the concepts of dynamic reconfiguration and keeping systems resilient to threats. Furthermore, this framework advocates the use of enforcement of policies service, which controls the adaptation and organization feature by operating on the meta data. Here, meta data represents the information about components and the environment.

2.2.5.1 Bio-Inspired Self-Adaptive Systems

Recently, there has been a considerable research that is currently being pursued by relating computing systems with biological systems. In [Bru08], a view on building bio-inspired *self-adaptive software* is presented. The main proposal revolves around the notion of modeling the biological system, which can be understood and then using that model by reiterating through it to generate a software design tool. Such tool yields quality of service such as fault-tolerance, robustness, security. In particular, software architecture and architectural styles extends such design tools. In this paper, tiled architectural style is adopted to support self-assembly model supporting the QoS as fault-tolerance.

2.3 Requirements Engineering for Self-Adaptive Software Systems

In particular, requirements engineering has been recognized as the seed activity in the whole software development life-cycle. Requirements engineering embraces sequential activities including elicitation, analysis, specification and validation [NE00]. Requirements are not only changed by the business level, but also emerge at run-time due to changes in the user needs, resource variability and environment. In this, the notion of requirements management is the key activity which prevails along all these activities. Requirements change management is an important activity when dealing with adaptive software.

Berry et al. in [BCZ05] identify four-level model for engineering dynamic adaptation requirements. It ranges from traditional RE activities, done by system analyst (level 1), to adaptation requirements the system responds to at run-time (level 2) and RE done by the analyst to determine adaptation mechanisms which enable the system to adapt (human-in-the loop adaptation scenario, level 3) and finally adaptation requirements associated to the specific adaptation solutions proposed by software engineering research (level 4).

2.3.1 Goal-Oriented Approaches

Likewise, to analyze adaptation requirements, goals have been acknowledged and used as a useful abstraction in many requirements engineering approaches [vL01b, LLY$^+$06]. The goal-oriented approach is also widely adopted for variability modeling, and has been well acknowledged in (early) requirements engineering for eliciting, specifying, analyzing, and documenting software requirements. The overall idea of goal-oriented modeling and analysis revolves around the notion of *goal*, which allows to capture functional and non-functional aspects of the system-to-be from stakeholder's perspective. At this point, the goal is decomposed into sub goals using AND/OR decompositions. Later a special attention is given to represent the refinements of various viewpoints considering the variability concerns. In this way, it is possible to reason about alternatives based on AND/OR decompositions. High-level goals are said to be satisfied when the sub goals have been fulfilled.

Penserini et al. in [PPSM07a] uses the *TROPOS* methodology to model variability in user's needs and preferences, in terms of alternatives for goal achievement at requirements-time. The modeled variability reflects into the design and coding of Belief-Desire-Intention (BDI) agents, which are able to switch from a behavior to a most appropriate one, depending on the perceived environment conditions (including user preferences) at run-time. In this work, goal alternatives represented in the requirements models map to software agents goal models in the design and code artifacts. This helps answering the question of "HOW", as well as "WHAT and WHY" for run-time adaptation. *self-adaptive software* requirements expressed as goal models can be mapped to BDI architecture resulting into an agent-based design framework to capture the said adaptivity requirements by Morandini et al. in [MPP08], which seems promising step towards an agent-oriented *self-adaptive software* research.

Liaskos et al. in [LLW$^+$05] follows a requirements driven approach to address the problem of variability by configuring software using goal-oriented approach. High-level user preferences are modeled as goal alternatives and by matching them with the intended system configuration. Thus supports reasoning about overlaying goal models to achieve an automatic system configuration. This appears to be a very useful approach to describe the behavior of autonomic elements based on goals.

More recently, goal models are used to derive autonomic element patterns by Zhu et al. in [ZLKM08]. Goal-oriented requirements engineering approach and attribute-based architectural style is used to articulate various autonomic element patterns. These patterns are organized from least to most flexible ones, mainly, characterizing maintainability concerns for *self-adaptive software*. This approach argues the reification of requirements using goal models, which constitutes a framework to reason about maintainability impact.

Along the perspective of goal-oriented requirements engineering, the core requirements ontology is redefined by Jureta et al in [JMF08]. This core ontology is based on goal-oriented concepts, mentalistic notions (belief, desire, intention) called modalities and the speech acts. Mainly, this ontology helps in articulating and understanding the requirements problem in a precise way. But, in order to build *self-adaptive software*, this ontology is not sufficient to deal with requirements change aspect and the adaptive nature of the requirements.

2.3.2 Requirements Monitoring Approaches

Nonetheless, goal-oriented approaches provide basis for refinement of requirements and architecture at run-time. Feather et al. in [FFLP98] use KAOS goal models to monitor requirements violations to reason for run-time behavior of a system and adapt dynamically using pre-defined adaptation tactics (parametric or choosing alternative actions). Moreover, KAOS models are formalized using temporal logic to provide traceability to verify. However, it could be useful, if the system can be made aware of its requirements (i.e.

Goal contributions +/-).

Similarly, Robinson [Rob09] proposed a comprehensive review of requirements monitoring approaches. A monitoring framework named REQMon (new Name EEAT i.e. Event Engineering and Analysis Toolkit) is proposed to monitor web service requirements using KAOS[DvLF93a]. His approach is based on identify potential requirements obstacles in order to monitor them. To do so, for each obstacle, an analyst is required to define a monitor that would retrieve the needed data. A version of OCL (object constraint language) for temporal logic is used to specify monitors.

Salifu et al. in [SYN07a] argue that monitoring is essential for requirements variations / violations and aids in composing switching behaviors in a context. This approach deals with the specification of monitoring problems and switching behaviors of core requirements, considered as invariants, using state charts to analyze the dynamicity and support elicitation. It differs from Feather et al. [FFLP98] approach in terms of monitoring the problems variations in a context besides the core requirements. Furthermore, this approach helps in deeper understanding of contextual variability and its impact on monitoring and switching problems, but it is limited, to scenarios, when considering an adaptive system.

2.3.3 Recent Vision for RE for SAS

Considering the software engineering road map for building *self-adaptive software* by B. Cheng et al. in [CGI$^+$08], there has been considerable emphasis given on various aspects of software engineering mainly requirements, modeling and design, engineering and assurance. This study brings forth many interesting challenges and above all emphasizes more on requirements engineering for *self-adaptive software*. The significant issues include having a requirements specification language, flexible techniques for mapping requirements to architectures, requirements reflection and traceability and lastly online goal refinement. There has been significantly less emphasis given to requirements engineering in general for building *self-adaptive software*.

Recently, a language for specifying requirements for *self-adaptive software* called "Relax" has been proposed by Whittle et al. in [WSB^{+}10]. Relax language guides the analyst at design-time to model the requirements via goal refinements adopting a goal modeling language i.e. KAOS [DvLF93b]. Temporal operators e.g. "as soon as possible", "until" etc. are used to specify requirements which can be relaxed without compromising the original goals. Obstacle analysis is used to identify potential threats to the requirements satisfaction at design-time. Relax uses the traditional RE to specify requirements at early stages and then uses the proposed modifiers (modal verbs) to relax SAS requirements. The main aim is to deal with uncertainty when specifying requirements in a flexible way so that a trade off or satisfaction of the goals can be reached. It helps in capturing the requirements and addressing the question: *how stakeholders requirements can be relaxed?* The proposed language serves as a first step towards requirements specification language for *self-adaptive software*.

It has been emphasized that traditional requirements engineering method and techniques are not enough to deal with the engineering of *self-adaptive software*. In this work, we follow the perspective of requirements engineering for building *self-adaptive software*, enabling not only the analyst to carry out adaptive requirements engineering but empowering the software to reason for adaptations at run-time. This will enable the software to achieve its user's goals and maintain them in case of any changes in the environment or due to resource variability. Requirements engineering is considered as the prime activity in the whole software development life-cycle, which must be performed at run-time by the *self-adaptive software* itself.

2.4 Final Remarks

In summary, existing approaches attempt to anticipate run-time changes at design-time, which limits them to accommodate new or changed requirements of end-users at run-time. Conceptualization used in these approaches vary depending upon the choice of the architecture their solution adopts. Therefore, there is a need for a requirements engineering (RE) framework that provides conceptual foundations by providing concepts that are needed to elicit and specify requirements for *self-adaptive software*, independent of any architecture. Also, to enable *self-adaptive software* to perform continuous requirement engineering activities to manage their requirements and solution to satisfy them at run-time involving the end-user.

In this thesis, we take the perspective of a requirements engineering, which has received less attention so far. Recent vision papers and road maps have identified several challenges for the field of RE for *self-adaptive software*. Among them, the need for SAS to hold representation of requirements at run-time and delaying the design decisions until run-time have been the foremost. Likewise, requirements engineering activities must also focus on methods and techniques that enables the system to perform RE at run-time. For this aim, revisiting conceptual frameworks for the elicitation and specification of requirements for *self-adaptive software* is needed.

Chapter 3

Requirements Problem for Self-Adaptive Software Systems

3.1 Overview

In this chapter we present the core concepts that are necessary to formulate the requirements problem for self-adaptive system. By requirements problem, we mean finding the specification that is composed of tasks and domain assumptions that are suitable in a given context and as per resource availability thus satisfies all the mandatory goals (functional goals and quality constraints) and also as much optional once. This problem formulation forms the basis of our approach that we are going to use throughout the thesis. We also present the interpretation of these concepts and discuss ways in which we exploit them for deriving the requirements specification for self-adaptive systems. To show the purpose of this problem definition, we show in subsequent chapters how this helps in systematically deriving the specification for self-adaptive systems (SAS) at design-time, thus enabling the SAS-to-be to exploit its own specification for monitoring and adaptation reasoning at run-time.

The chapter is organized as follows. In Section 3.2 we briefly recall the general requirements problem definition and its recent revision by using the revisited Core ontology for Requirements Engineering (RE). In Section 3.3 we present the our definition of requirements problem for *self-adaptive software*. In Section 3.4 we introduce new concepts and relations in the existing Core ontology for RE. In Section 3.5 we illustrate with the help of the running example how this requirements problem definition can be seen as a dynamic RE problem at run-time. Finally, in Section 3.6 we discuss the benefits of these proposed concepts in the Core ontology for RE and requirements problem for SAS considering the design-time and run-time. We summarize this chapter in Section 3.7.

3.2 Requirements Problem and Core ontology for RE

Our work builds upon problem definitions and ideas developed in studies pertaining to specific, although related areas, namely engineering of SAS, as well as in works on RE methods and specifically on foundations of RE. Basic concepts are recalled below.

The overall aim of RE is to identify the purpose of the system-to-be and to describe as precisely and completely as possible the properties and behaviors that the system-to-be should exhibit in order to satisfy that purpose. This is also a rough statement of the requirements problem that should be solved when engineering requirements for any system.

Zave & Jackson [ZJ97b] formalized the requirements problem as finding a specification (S) in order to satisfy requirements (R) and does not violate domain assumptions (K), and thereby ensuring that $K, S \vdash R$. This formulation highlights the importance of the specification to be consistent with domain assumptions, and that requirements should be derivable from K and S. It was subsequently argued that there is more to the requirements problem than this formulation states [JMF08]. Namely, a new core ontology for requirements (Core) was suggested along with a new formulation of the requirements problem to recognize that in addition to goals and tasks, different stakeholders have different preferences over requirements, that they are interested in choosing among candidate solutions to the requirements problem, that potentially many candidate solutions exist (as in the case of service-/agent-oriented systems, where different services/agents may compete in offering the same functions), and that requirements are not fixed, but change with new information from the stakeholders or the operational environment. In absence of preferences, it is (i) not clear how candidate solutions to the requirements problem can be compared, (ii) what criteria (should) serve for comparison, and (iii) how these criteria are represented in requirements models.

New concepts suggested in Core led to the revised formulation of the requirements problem: **Given** the elicited domain assumptions, goals, quality constraints, softgoals, tasks, some of which are optional, mandatory, and/or preferred over others, **find** tasks and domain assumptions which satisfy all mandatory goals, quality constraints, and ideally also satisfy at least some of the preferred and/or optional goals and quality constraints. A candidate solution to this problem will be a consistent set of tasks and domain assumptions which satisfy all mandatory goals and quality constraints. Candidate solutions are compared on the basis of which preferred and/or optional goals and quality constraints they satisfy, so as to select one solution among the candidates, and engineer the system-to-be according to the requirements and other information in that solution.

Techne [JBEM10b], an abstract requirements modeling language was recently introduced as as a starting point for the development of new requirements modeling languages that can be used to represent information and perform reasoning needed to solve the requirements problem. Techne is abstract in that it assumes no particular visual syntax (e.g., diagrammatic notation such as those present in i-star [Yu95] and Tropos [PPSM07b]), and it includes only the minimum concepts and relations needed to formalize the requirements problem and the properties of its solutions. Techne is consequently a convenient formalism to formalize the run-time requirements adaptation problem. It is simple and adapted to the concepts, such as goal, task, domain assumption, and relations (e.g. Techne consequence, Preference, Is-mandatory, Is-optional) that remain relevant for the RE of SAS. However, in order to address the challenge of engineering requirements for SAS, Techne does not offer answers to the following questions:

- What are the important aspects to express adaptivity?
- What will happen if requirements evolve at run-time and demand the software to adapt?
- What kind of characterization is required to identify adaptive requirements with respect to other types?
- How to express the adaptive requirements in more flexible way?

To enable a simple and convenient formalism, we argue that requirements and other information is available in propositional form, so that every proposition is nothing but a natural language sentence. We briefly overview below the requirements problem formulation using parts of Techne that we consider are relevant for our problem formulation.

Definition 1 ***Techne language:*** *The language $\mathcal{L}$ is a finite set of expressions, in which every expression $\phi \in \mathcal{L}$ satisfies the following* BNF *specification:*

$$x ::= \boldsymbol{k}(p) \mid \boldsymbol{g}(p) \mid \boldsymbol{q}(p) \mid \boldsymbol{s}(p) \mid \boldsymbol{t}(p) \tag{3.2.1}$$

$$y ::= \bigwedge_{i=1}^{n} x_i \rightarrow x \mid \bigwedge_{i=1}^{n} x_i \rightarrow \perp \tag{3.2.2}$$

$$\phi ::= x \mid \boldsymbol{k}(y) \tag{3.2.3}$$

remark 1 *Uppercase Greek letters Δ, Π, Φ, Ψ denote sets of expressions, i.e., $\Delta \subseteq \mathcal{L}$. Lowercase Greek letters ϕ, ψ, α, β, γ denote individual expressions,* *i.e., members of $\mathcal{L}$. We index or prime symbols as needed. $\rightarrow$ reads "if–then" and $\wedge$ reads "and". Lowercase Latin alphabet letters p, q, r, s denote propositions.*

Labels on propositions and expressions refer to concepts in the core ontology for requirements, and are assigned as follows:

- **Domain assumptions:** $\mathsf{k}(p)$ if p is an instance of the Domain assumption concept in the Core ontology, refers to conditions that are believed to hold (e.g., legal norms, assumptions about how some part of the environment behaves, what properties it has, and so on).
- **Goals:** $\mathsf{g}(p)$ if p is an instance of the Goal concept, i.e., p refers to conditions, the satisfaction of which is desired, binary, and verifiable (e.g., desired functionalities of the system-to-be).
- **Quality constraints:** $\mathsf{q}(p)$ if p is an instance of the Quality constraint concept, that is, refers to conditions that constrain the desired values of non-binary measurable properties or behaviors (e.g., the length of the encryption key, the response time of a software module).
- **Softgoals:** $\mathsf{s}(p)$ if p is an instance of the Softgoal concept, i.e., refers to a vague condition that constrains desired values of (potentially) not directly measurable properties or behaviors (e.g., software should respond quickly, interface should be usable).
- **Tasks:** $\mathsf{t}(p)$ if p is an instance of the Task concept, and thereby refers to behaviors of the system-to-be and/or within its operating environment (e.g., find, transform, produce information, manipulate objects).

The language we have is simply a sorted propositional language in which only conjunction and implication connectives are allowed, and are used in a restricted way, as the BNF specification makes clear. The ontology lets us define a sorting function as follows.

Definition 2 ***Sorting function:*** *Sort* $: \mathcal{L} \longrightarrow \mathcal{O}$, *where* $\mathcal{O}$ *is the set of sort labels, one for each of the following concepts: Strict domain assumption, Defeasible domain assumption, Goal, Quality constraint, Softgoal and Task.*

remark 2 *To simplify notation, we will abbreviate Sort*$(\phi) = \textbf{g}$ *by* $\textbf{g}\phi$*, to say that* ϕ *is a goal. This abuses the notation somewhat, since the* BNF *specification tells us that* $\phi = \textbf{g}(p)$.

In Techne consequence relation is defined as follows.

Definition 3 ***Techne consequence relation:*** *Let* $\Pi \subseteq \mathcal{L}$, $\phi \in \mathcal{L}$, *and* $z \in \{\phi, \perp\}$*, then:*

- $\Pi \mid\!\sim_{\mathcal{T}} \phi$ *if* $\phi \in \Pi$, *or*
- $\Pi \mid\!\sim_{\mathcal{T}} z$ *if* $\forall 1 \leq i \leq n$, $\Pi \mid\!\sim_{\mathcal{T}} \phi_i$ *and* $\textbf{k}(\bigwedge_{i=1}^{n} \phi_i \rightarrow z) \in \Pi$.

remark 3 *The consequence relation* $\mid\!\sim_{\mathcal{T}}$ *is sound w.r.t. standard entailment in propositional logic, but is incomplete in two ways: it only considers deducing positive atoms, and no ordinary proofs based on arguing by contradiction go through, thus being paraconsistent [JBEM10b].*

Techne includes three relations that remain outside the formal Techne language, as they do not participate in the definition of the consequence relation in Techne. These relations are as follows.

Definition 4 ***Preference relation:*** *The preference relation* $\succ \subseteq \mathcal{L} \times \mathcal{L}$ *is an irreflexive binary relation read as follows: if ensuring that* $\phi \in \mathcal{L}$ *holds is strictly more desirable than ensuring that* $\psi \in \mathcal{L}$ *holds, then we say that* ϕ *is strictly preferred to* ψ *and denote this* $\phi \succ \psi$.

remark 4 *We do not require the preference relation to be complete, transitive, or have other specific properties, other than that it is irreflexive. The appropriate choice of these additional properties will depend on, e.g., the automated* *reasoning framework chosen to compute the most preferred requirements given a set of preference relations.*

Definition 5 ***Is-mandatory relation:*** *The is-mandatory relation is a unary relation on a requirement to indicate that the requirement must be satisfied, or equivalently, that every candidate solution must include the mandatory requirement.*

Definition 6 ***Is-optional relation:*** *The is-optional relation is a unary relation on a requirement to indicate that it would be desirable for a candidate solution to include that requirement, but that doing so is not mandatory, or equivalently, that if two candidate solutions differ only in one optional requirement, then the candidate solution which includes that optional requirement is strictly preferred to the candidate solution which does not include that optional requirement.*

The consequence relation leads us to the following conception of the *candidate solution* concept.

Definition 7 ***Requirements problem in Techne: Given*** *the elicited or otherwise acquired: domain assumptions (in the set* ***K****),tasks in* ***T****,goals in* ***G****, quality constraints in* ***Q****, softgoals in* ***S****, and preference, is-mandatory and is-optional relations in* ***A****,* ***find*** *all* **candidate solutions** *to the requirements problem and compare them using preference and is-optional relations from* ***A*** *to identify the most desirable candidate solution.*

remark 5 ***A*** *is a set that includes all preference relations, all is-optional, and all is-mandatory relations.*

Definition 8 ***Candidate solution:*** *A set of tasks* $\boldsymbol{T}^*$ *and a set of domain assumptions* $\boldsymbol{K}^*$ *are a* **candidate solution** *to the requirements problem if and only if:*

1. *$\boldsymbol{K}^*$ and $\boldsymbol{T}^*$ are not inconsistent,*
2. *$\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\!\sim_{\mathcal{T}} \boldsymbol{G}^*, \boldsymbol{Q}^*$, where $\boldsymbol{G}^* \subseteq \boldsymbol{G}$ and $\boldsymbol{Q}^* \subseteq \boldsymbol{Q}$,*
3. *$\boldsymbol{G}^*$ and $\boldsymbol{Q}^*$ include, respectively, all mandatory goals and quality constraints, and*
4. *all mandatory softgoals are approximated by the consequences of $\boldsymbol{K}^* \cup \boldsymbol{T}^*$, so that $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\!\sim_{\mathcal{T}} \boldsymbol{S}^{\boldsymbol{M}}$, where $\boldsymbol{S}^{\boldsymbol{M}}$ is the set of mandatory softgoals.*

We start below from the Core ontology and problem formulation in Techne, and add concepts specific to the RE for SAS, which leads us to an ontology for requirements in SAS and the formulation of the *requirements problem in context* for SAS. We subsequently show how to formulate the *run-time requirements adaptation problem* as a dynamic problem of changing (e.g. switching, re-configuring, optimizing) the SAS from one requirements problem to another requirements problem, whereby the changing is due to change in requirements, context conditions, and/or resource availability.

3.3 Defining Run-time Requirements Adaptation Problem

Various definitions of SAS have been offered in the literature. We remain aligned with the usual conception, namely, that a SAS is a software system that can alter its behavior in response to the changes that occur dynamically in its operating environment. The operating environment can include anything that is observable by the software itself including context, end-user's input and resource variabilities. Such continuously running systems requires flexibility in managing themselves by exercising autonomic properties called as Self-*[KC03] in response to the changing requirements at run-time.

We start by describing how we identify the kinds of information that pertain to requirements, as well as relations between such information, of which a SAS needs to have an internal representation in order to ensure that it satisfies as best as feasible its users requirements even when there are changes in its operating context and/or in the resources available to it.

To this aim, we formally define a minimal set of concepts and relations needed to formulate the requirements problem, its solutions, the changes in its formulation that arise from changes in the operating context, requirements, and resource availability. We thereby answer the following research questions in the RE for SAS:

1. What concepts and relations capture the requirements-related information that a SAS needs to have an internal representation of in order to satisfy users' requirements, and their change across changing operating contexts and/or resource availability?

2. How do these concepts and relations come together in the formulation of the requirements problem that SAS need to solve?

3. What relationships should be maintained in case of changes that occur dynamically at run-time (unanticipated, or initiated by users)?

This leads us to precisely define the *run-time requirements adaptation problem* as a *Dynamic RE problem* that a SAS should be engineered to solve. We thereby suggest concepts and relations that are necessary to deal with while eliciting and analyzing requirements for SAS and are important to take adaptation decision at run-time by the system itself. To fulfill this aim, we describe to make explicit the dynamic parts in the requirements problem formulation based on the Core ontology[JMF08].

3.4 RE for SAS and its Core Ontology

Building upon the above considerations, we argue that concepts such as user context e.g. profile, location, operational) and resource (e.g. tangible, intangible including money, social-relations) must be considered as first class citizens in the existing Core ontology to engineer requirements for SAS. We add two new concepts, Context and Resource on top of the Core ontology to accommodate the changes that might occur at run-time, which not only demands adaptation (i.e. dynamically changing from one requirements problem to another) but also requires an update to the specification (i.e. refinement of requirements). Moreover, these concepts can enhance the tool set for eliciting and analyzing requirements at run-time.

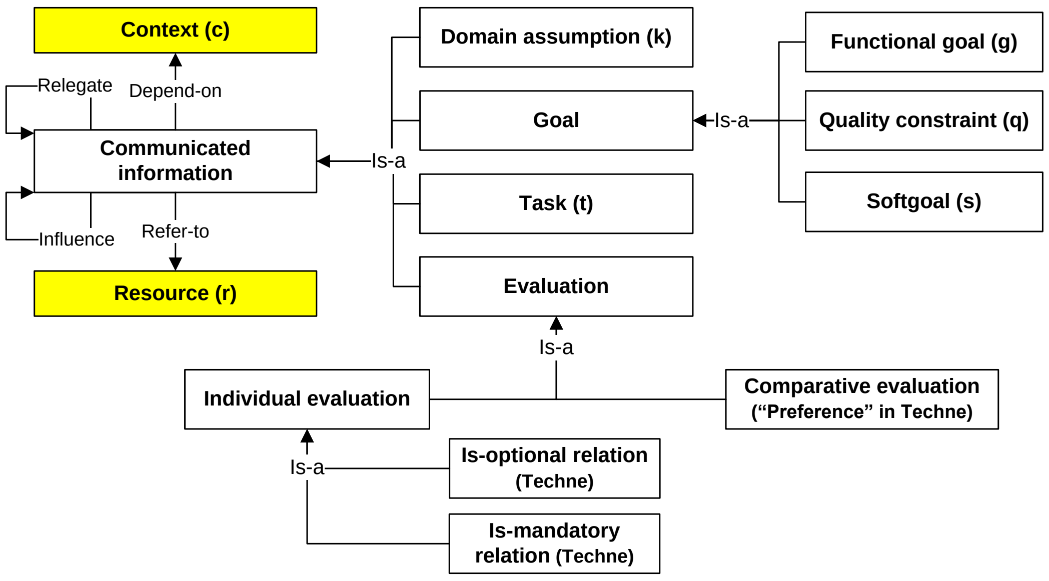


Figure 3.1: New concepts (Context and Resource) and relations (Relegation and Influence) related to the concepts of the Core ontology for RE.

The revised taxonomy of the concepts is proposed taking into account the necessary concepts proposed in Techne [JBEM10c] to support the definition of run-time "**run-time requirements adaptation problem**". In Fig. 3.1, new concepts (Context and Resource) and relations (Relegation and Influence) related to the concepts of the Core ontology for RE are proposed (for details, see [JMF08]). Below, we introduce the concept of requirements database.

Definition 9 ***Requirements database:*** *A requirements database, denoted Δ is the set of all information elicited or otherwise acquired during the* RE *of a* *system-to-be.*

remark 6 *Firstly, $\Delta \subseteq \mathcal{L}$, i.e., every member of Δ is an expression in $\mathcal{L}$. Secondly, since Δ should include all information elicited or otherwise acquired in* RE, *it should include all instances of domain assumptions, goals, softgoals, quality constraints, and tasks that we elicited, found through refinement or otherwise identified during* RE. *Thirdly, one can view Δ as a repository of information that is usually found in what is informally referred to as a "requirements model". We use the concept of requirements database as a machine readable form of "requirements model". Finally, in the notation used in the definition of the requirements problem, note that $\Delta = \boldsymbol{K} \cup \boldsymbol{T} \cup \boldsymbol{G} \cup \boldsymbol{Q} \cup \boldsymbol{S}$.*

remark 7 *Below, we will use the term requirement to abbreviate "member of the requirements database Δ". I.e., we will call every member of Δ a requirement.*

To get to the definition of the *run-time requirements adaptation problem*, we start introducing the Context concept. We are aware of the existing definitions and use of Context in the existing AI and RE literature, for instance [McC93, Dey01, SYN07b, ADG08].

Definition 10 ***Context:*** *An instance C of the* Context *concept is a set of information that is presupposed by the stakeholders to hold when they communicate particular requirements. We say that every requirement depends on one or more contexts to say that the requirement would not be retracted by the stakeholders in every one of these contexts.*

remark 8 *Firstly, we need a language to write this information that is presupposed, and is thereby in the set of information that we call a particular context. We develop that language below. Secondly, the dependence of a requirement on a context means that every requirement is specific to one or more contexts, and thus, requirements need to be annotated by contexts, which begs additional questions on how the engineer comes to determine contexts.*

The term context can be interpreted in various ways, for example, an object, an entity, a surrounding, an environment (operational, external to the system), a set of conditions, a location, a situation. Generally context is a very imprecisely defined concept, it can be defined as *surrounding, circumstances, environment, background, or settings which determine, specify, or clarify the meaning of an event.*[1]

The concept of context has been used in AI and RE literature to characterize information internal or external to the system-to-be. For instance, in AI literature, the term *context has been characterized and formalized to make systems reason on either the circumstances where particular propositions holds or at a certain time instance where the particular proposition might hold* by McCarthy in [McC93]. In RE literature *context and context-aware systems are defined as an abstraction of location, an event, environment or as a set of conditions that may change overtime* in [SYN07b, FS01, FFLP98, ADG09, SS07]. For instance, another most common and well accepted definition of context to date is by Dey in [Dey01] i.e. *Context is any information that can be used to characterize the situation of an entity.* An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Alternatively, *a context **C** refers to any information that helps in characterizing a perceived state of the world.* A context can be characterized as an event or a state of the entity (involving the user) and the operational environment in which the system operates.

Example: The user is traveling, during transit, the context of the user has changed from *traveling* to *stay at airport.*

Example: Studying *indoor* or *outdoor*, with a profile setting *silent* or *airport* or *normal*, all refers to the perceived state of the world.

At this point, we revise the Techne language to allow information that is

[1]http://en.wikipedia.org/wiki/Context

included in contexts. This results in adding one more sort.

Definition 11 ***Language for SAS:*** *The language $\mathcal{L}_{SAS}$ is a finite set of expressions, in which every expression $\phi \in \mathcal{L}_{SAS}$ satisfies the following* BNF *specification:*

$$x ::= \boldsymbol{k}(p) \mid \boldsymbol{g}(p) \mid \boldsymbol{q}(p) \mid \boldsymbol{s}(p) \mid \boldsymbol{t}(p) \tag{3.4.4}$$

$$q ::= \boldsymbol{c}(p) \tag{3.4.5}$$

$$w ::= x \mid q \tag{3.4.6}$$

$$y ::= \bigwedge_{i=1}^{n} w_i \rightarrow w \mid \bigwedge_{i=1}^{n} w_i \rightarrow \bot \tag{3.4.7}$$

$$\phi ::= w \mid \boldsymbol{k}(y) \mid \boldsymbol{c}(y) \tag{3.4.8}$$

remark 9 *We used (indexed/primed p, q, r) as an arbitrary atomic statement, every ϕ an arbitrary complex statement, and every x an arbitrary label to represent Techne labeled propositions i.e. domain assumption (k(p)), a goal (g(p)), etc. to distinguish from these basic labeled propositions the context propositions (i.e., propositions about context), c(p) is added separately in the BNF specification, via q, and every w can either be x or q. Every y represents a complex statement as a formula with conjunction and implication such that y can be either w or $\bot$, where w is some requirement in a context propositions and $\bot$ refers to logical inconsistency. We can then rewrite ϕ as a complex statement consists of either w or $\boldsymbol{k}(y)$ or $\boldsymbol{c}(y)$.*

Definition 12 ***Consequence relation of Techne in context:*** *Let $\Pi \subseteq \mathcal{L}_{SAS}$, $\phi \in \mathcal{L}_{SAS}$, and $z \in \{\phi, \bot\}$, then:*

- $\Pi \mid\!\sim_{\mathcal{CT}} \phi$ *if* $\phi \in \Pi$, *or*
- $\Pi \mid\!\sim_{\mathcal{CT}} z$ *if* $\forall 1 \leq i \leq n$, $\Pi \mid\!\sim_{\mathcal{CT}} \phi_i$ *and* $\boldsymbol{k}(\bigwedge_{i=1}^{n} \phi_i \rightarrow z) \in \Pi$.

remark 10 *The consequence relation $\mathrel{|\!\!\sim}_{\mathcal{C}\mathcal{T}}$ is sound w.r.t. standard entailment in propositional logic. It deduces only positive statement by being paraconsistent, thus all admissible candidate solutions are found via paraconsistent and non-monotonic reasoning. Reasoning is paraconsistent because an inconsistent Δ or C should not allow us to conclude the satisfaction of all requirements therein; it is non-monotonic in that prior conclusions drawn from a Δ or a C may be retracted after new requirements are introduced.*

We also need a function that tells us which contexts a requirement applies to.

Definition 13 ***Contextualization function:*** *Let $\boldsymbol{C}$ be the set of all contexts. $\mathcal{C} : \wp(\mathcal{L}_{\boldsymbol{SAS}}) \longrightarrow \wp(\boldsymbol{C})$ (where $\wp$ returns the powerset) is called the contextualization function that for a given set of formulas returns the set of contexts to which these formulas apply to. By "apply to", we mean that $C \in \mathcal{C}(\phi)$ iff the following conditions are satisfied:*

1. *$C, \phi \not\mathrel{|\!\!\sim}_{\mathcal{T}} \perp$, i.e., ϕ is not inconsistent with context C,*
2. *C is such that $\exists X \subseteq \Delta$ such that $C, X \mathrel{|\!\!\sim}_{\mathcal{T}} \phi$, i.e., the context C together with some requirements X from Δ lets us deduce ϕ.*

remark 11 *Several remarks are in order.*

Firstly, with $\mathcal{L}_{\boldsymbol{SAS}}$, we now have a new sort for expressions that are members of a set that defines a context. Recall that we defined an instance C of Context *as a set of information, so that now $\mathcal{L}_{\boldsymbol{SAS}}$ tells us that one member of that set can either be a proposition p, denoted $\textbf{c}(p)$, or can be a formula with implication, denoted $\textbf{c}(y)$ in the* BNF *specification.*

E.g., if the engineer assumes that the stakeholders wants that her goal $\textbf{g}(p)$ for "arrive at destination" be satisfied both in the context C_1 in which the context proposition "$\textbf{c}(q)$: flight is on time" holds (i.e., $\textbf{c}(q) \in C_1$), and in the context C_2 in which the context proposition "$\textbf{c}(r)$: flight is delayed *but not more than 5 hours" holds (i.e., $\boldsymbol{c}(r) \in C_2$), then $C_1 \in \mathcal{C}(\boldsymbol{g}(p))$ and $C_1 \in \mathcal{C}(\boldsymbol{g}(p))$.*

Secondly, observe that the BNF *specification lets us write formulas in which we combine context propositions and requirements, e.g.:*

$$\boldsymbol{k}(p) \wedge \boldsymbol{c}(q) \to \bot$$

which the requirements engineer can use to state that the domain assumption $\boldsymbol{k}(p)$ that was communicated by the stakeholder does not hold in contexts in which the context proposition $\boldsymbol{c}(q)$ holds. The formula $\boldsymbol{k}(p) \wedge \boldsymbol{c}(q) \to \bot$ itself can be a context formula – we state this by writing $\boldsymbol{c}(\boldsymbol{k}(p) \wedge \boldsymbol{c}(q) \to \bot)$ – in order to capture the idea that it is presupposed that $\boldsymbol{c}(q)$ and $\boldsymbol{k}(p)$ cannot hold together. Observe that if we write $\boldsymbol{c}(\boldsymbol{k}(p) \wedge \boldsymbol{c}(q) \to \bot)$, then by the definition of the contextualization function $\mathcal{C}$, we can see that $\boldsymbol{c}(q) \notin \mathcal{C}(\boldsymbol{k}(p))$, which informally tells us that we cannot ensure $\boldsymbol{k}(p)$ in contexts in which we have $\boldsymbol{c}(q)$.

Since we can combine context formulas and requirements, we can state very useful relations, such as that some requirements conflict with some contexts, by saying that these requirements are inconsistent with some of the context formulas in these contexts.

Thirdly, we could have been more demanding in the definition of the contextualization function $\mathcal{C}$. In the above definition, $\mathcal{C}(\phi)$ will be the set of all information consistent with ϕ, which is usually too much: that set will have information which is not used to derive ϕ, even though it is consistent with ϕ. We leave this definition as it is, however, mainly because it covers the said and other more specific cases and we do not need to focus on minimizing the set $\mathcal{C}(\phi)$. Given that problems of automated reasoning e.g. in case of reasoning over multiple context are outside of the scope in the present discussion.

As an aside, rules that connect requirements and context formulas need not be specified in a definite way by the requirements engineer. It is possible to *learn them by asking feedback to the user. For example, if the system asks the user a question of the form:*

> *Your flight is delayed by 5 hours or more. Do you wish to rebook a flight for the next day?*

This question can be reformulated as a question on which of these two formulas to add to the current context of the user (the context in which we asked the user that question):

$$\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_1) \rightarrow \perp) \tag{3.4.9}$$

$$\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_2) \rightarrow \perp) \tag{3.4.10}$$

where $\mathbf{c}(p)$ is for "flight delayed by more than 5 hours", $\mathbf{g}(q_1)$ is for the goal "keep the booked flight", and $\mathbf{g}(q_2)$ is for the goal "rebook the same flight for the next day". If the user answers "yes", then add formula $\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_1)) \rightarrow \perp$ to the context in which we asked the user that question; if the user answers "no", then we add $\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_2)) \rightarrow \perp$ to the current context.

We now add the Resource concept. The formal language that we use is propositional, we will keep the resource concept out of it.

Definition 14 ***Resource:*** *An instance R of the* Resource *concept is an entity either tangible or intangible referred to by one or more instances of* *Communicated information.*

The concept of resource **R** has been well supported in RE methods such as in goal-oriented approaches [vL01b, Yu95, BPG$^+$04]. Generally resource can be defined as: *"A resource is any physical or virtual entity of limited availability that needs to be consumed to obtain a benefit from it".*[2] It can be either natural resource, human resource or tangible/intangible resource.

[2]http://en.wikipedia.org/wiki/Resource

We refer to tangible/intangible resources in our work e.g. resources that can be consumed or used by the system for the sake of information processing. To be precise, in our case we refer them as physical/tangible entities e.g. mobile phone, ticket itinerary. Whereas, they can also be intangible i.e. user assets i.e. social relations or contacts. Alternatively, *a resource **R** can be characterized as tangible entity in the world or an intangible entity.*

Example: The user is boarding for the flight. At the check-in counter the itinerary represents an informational resource. It can be either printed or kept in a digital format.

Example: User can view itinerary on her mobile devices. In this case, itinerary represents a digital information resource, where as mobile device refers to a physical resource. Another case is that user can view her agenda, personal contacts or information about their Money balance. Here, we refer to three assets (intangible resources) i.e. Agenda, Personal Contacts/Social relations and Money. All these can be modified at user's discretion.

In order to introduce resources in the definition of the requirements adaptation problem, we need a function that tells us which resources are referred to by a task, domain assumption, or a context proposition, as these resources will need to be available and used in some way in order to ensure that the relevant domain assumptions and context propositions hold, and that the tasks can be executed.

Definition 15 ***Resource selector function:*** *Let **C** be the set of all contexts. Given a set of tasks, domain assumptions, and/or context propositions, the resource selector function returns the identifiers of resources necessary for the domain assumptions and/or context propositions to hold, and/or tasks to be executed:*

$$\mathcal{R} : \wp(\boldsymbol{T} \cup \boldsymbol{K} \cup \bigcup \boldsymbol{C}) \longrightarrow \wp(\boldsymbol{R}) \tag{3.4.11}$$

remark 12 *The domain of $\mathcal{R}$ are domain assumptions, context propositions,* *and tasks. The reason that goals, softgoals, and quality constraints are absent is that the resources will be mobilized to realize a candidate solution to the requirements problem, and the candidate solution includes only domain assumptions and tasks. Since these domain assumptions and tasks are contextualized, we need to ensure the availability of resources that are needed in the context on which these domain assumptions and tasks depend on.*

*Note also that we have $\bigcup$**C** because **C** is a set of sets, so that we need to get the union of all of the sets in **C**.*

We can now formulate the requirements adaptation problem for SAS.

Definition 16 ***Runtime requirements adaptation problem: Given*** *a candidate solution $S(C_1)$ in the context $C_1 \in$* ***C*** *to the requirements problem RP(C_1) in context $C_1 \in$* ***C****, and a change from context C_1 to $C_2 \neq C_1$,* ***find***

1. *the requirements problem RP(C_2) in context $C_2 \in$* ***C*** *and*
2. *choose among candidate solutions to RP(C_2) a solution $S(C_2)$ in the context C_2 to the requirements problem $RP(C_2)$ in the context $C_2 \in$* ***C****.*

remark 13 *The definition of the requirements adaptation problem reflects the intuition that by changing the context, the requirements problem may change – as requirements can change – and from there, a new solution needs to be found to the requirements problem in the new context.*

We now reformulate the requirements problem so as to highlight the role of context in it, as well as of the resources.

Definition 17 ***Requirements problem RP(C) in context C: Given*** *the elicited or otherwise acquired:*

- *domain assumptions in the set* ***K****,*
- *tasks in* ***T****,*

- *goals in **G**,*
- *quality constraints in **Q**,*
- *softgoals in **S**,*
- *preference, is-mandatory and is-optional relations in **A**,*
- *a context C on which $\boldsymbol{K} \cup \boldsymbol{T} \cup \boldsymbol{G} \cup \boldsymbol{Q} \cup \boldsymbol{S}$ and **A** depend on,*

*the requirements problem in C" by "**find** all candidate solutions $S_1(C), \ldots, S_n(C)$".*

Definition 18 ***Candidate solution** $S(C_1)$ **in the context** C**:** A set of tasks $\boldsymbol{T}^*$ and a set of domain assumptions $\boldsymbol{K}^*$ are a **candidate solution in the context** C to the requirements problem RP(C) in context C if and only if:*

1. *$\boldsymbol{K}^*$ and $\boldsymbol{T}^*$ are not inconsistent,*
2. *$C, \boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{T}} \boldsymbol{G}^*, \boldsymbol{Q}^*$, where $\boldsymbol{G}^* \subseteq \boldsymbol{G}$ and $\boldsymbol{Q}^* \subseteq \boldsymbol{Q}$,*
3. *$\boldsymbol{G}^*$ and $\boldsymbol{Q}^*$ include, respectively, all mandatory goals and quality constraints,*
4. *all mandatory softgoals are approximated by the consequences of $C, \boldsymbol{K}^* \cup \boldsymbol{T}^*$, so that $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{T}} \boldsymbol{S^M}$, where $\boldsymbol{S^M}$ is the set of mandatory softgoals, and*
5. *resources $\mathcal{R}(C \cup \boldsymbol{K}^* \cup \boldsymbol{T}^*)$ needed to realize this candidate solution are available.*

We can define the relegation relation via the inference and preference relations in Techne.

Definition 19 ***Relegation relation:** A relegation relation (Rel) is an $n+1$-ary relation that stands between a requirement $\phi \in \Delta$ and n other sets of requirements $\Pi_1, \Pi_2, \ldots, \Pi_n \subseteq \Delta$ if and only if the following conditions are satisfied:*

1. *$\forall 1 \leq i \leq n$, $\Pi_i \mid\!\sim_{\mathcal{T}} \phi$, i.e., there is an inference relation from every Π_i to ϕ;*

2. *there is a binary relation: $\succ_\phi \subseteq \{\Pi_i \mid 1 \leq i \leq n\} \times \{\Pi_i \mid 1 \leq i \leq n\}$ such that, for any three different $\Pi_i, \Pi_j, \Pi_k \in \{\Pi_i \mid 1 \leq i \leq n\}$:*

   (a) $\succ_\phi$ is irreflexive, i.e., it is not the case that $\Pi_i \succ_\phi \Pi_i$;

   (b) $\succ_\phi$ is transitive, i.e., if $\Pi_i \succ_\phi \Pi_j$ and $\Pi_j \succ_\phi \Pi_k$, then $\Pi_i \succ_\phi \Pi_k$; and

   (c) $\succ_\phi$ is connected, i.e., for any two Π_i and Π_j, either $\Pi_i \succ_\phi \Pi_j$ or $\Pi_j \succ_\phi \Pi_i$.

   whereby $\Pi_i \succ_\phi \Pi_j$ if it is strictly more desirable to satisfy ϕ by ensuring that Π_i holds, than to satisfy ϕ by ensuring that Π_j holds.

remark 14 *The inference relations required by a relegation relations indicate that a relegation relation can only be defined for requirements that we know how to satisfy in different ways. For example, if we have a goal $\boldsymbol{g}(p)$, and we have two ways to satisfy that goal, e.g.:*

$$\Pi_1 = \{\boldsymbol{t}(q_1), \boldsymbol{b}(\boldsymbol{t}(q_1) \rightarrow \boldsymbol{g}(p))\} \tag{3.4.12}$$

$$\Pi_2 = \{\boldsymbol{t}(q_2), \boldsymbol{b}(\boldsymbol{t}(q_2) \rightarrow \boldsymbol{g}(p))\} \tag{3.4.13}$$

then we have satisfied the first condition from the definition of the relegation relation, since $\Pi_1 \mid\!\sim_{\mathcal{T}} \boldsymbol{g}(p)$ and $\Pi_2 \mid\!\sim_{\mathcal{T}} \boldsymbol{g}(p)$.

The second condition in the definition of the relegation relation says that we need to define a preference relation $\succ_{\boldsymbol{g}(p)}$ between different ways of satisfying $\boldsymbol{g}(p)$. Observe that we define $\succ_{\boldsymbol{g}(p)}$ between sets of information, not pieces of information. The Techne preference relation defines preference between individual pieces of information, so we can use preference relations between members of Π_1 and Π_2 to define $\succ_{\boldsymbol{g}(p)}$.

Suppose that $\mathbf{t}(q_1) \succ \mathbf{t}(q_2)$*, i.e., that we prefer to execute task* $\mathbf{t}(q_1)$ *to executing the task* $\mathbf{t}(q_2)$*. We can define* $\succ_{\mathbf{g}(p)} = f(\succ)$*, that is, from the information that the preference relation already includes. Namely, in this example it is appropriate to say that, if* $\mathbf{t}(q_1) \succ \mathbf{t}(q_2)$*, then* $\Pi_1 \succ_{\mathbf{g}(p)} \Pi_2$*. Since we have only* Π_1 *and* Π_2*, it is enough to know that* $\Pi_1 \succ_{\mathbf{g}(p)} \Pi_2$ *to know everything we need to define the relegation relation.*

Namely, the relegation relation $(\mathbf{g}(p), \Pi_1, \Pi_2, \succ_{\mathbf{g}(p)})$ *tells us that, if we cannot satisfy* $\mathbf{g}(p)$ *through* Π_1 *then we will relegate to* Π_2*, i.e., satisfy* $\mathbf{g}(p)$ *through* Π_2*.*

The purpose of Relegation relation **Rel** is twofold. First, it facilitates at design-time to analyze requirements (including Goals, quality constraints, preferences) and relegate their associated conditions (e.g. pre/post, achievement, trigger conditions) by anticipating run-time failures scenarios. Second, it enables SAS at run-time to analyze requirements (including Goals, quality constraints, preferences) in case of changes that can occur dynamically e.g. change in user's context, violation of domain assumption, resource usage or change in user's need or preference, either through sensing the operational environment or explicitly given by the end-user. A **Rel** is applied to counter uncertainties (e.g. unanticipated event) by flexibly relegating some of the requirements, if needed, to achieve critical ones.

We take an example of SAS running on user's mobile phone and monitors user's itinerary during her travel. It observes a prolonged delay in the flight due to Icelandic volcanic eruptions. Satisfying user's high level goal at run-time e.g. "Travel for Business Meeting" equates to the execution of a plans e.g. "Book Flight", "Meeting Scheduled", along with their specific parameters and conditions e.g. "flight origin and destination", "flight schedules", "meeting location", "time of meeting". In this case, SAS must re-evaluate the goal conditions and plans parameters and decide to relegate some conditions or preference criteria to formulate a solution that helps achieving the high level goal of the user. SAS may decide to relegate the goal achievement condition by adding another auxiliary goal "Travel for Business 2 days after" taking into account the available information, i.e. "flight schedules delayed for 2 days", user's current context is "at airport". The instance of the goal i.e. "Travel for Business" is linked with the new goal "Travel for Business 2 days after" using **Rel** by operationalizing it with a similar solution with changed parameters i.e. "Book flight after 2 days". This solution is recommended to the end-user. Upon acceptance, the resource i.e. Travel itinerary will be changed subsequently. Alternatively, end-user could change her itinerary by explicitly providing the input to the SAS. **Rel** does not manifest the original goal of the user to be compromised rather relegate it as per the changes in the operational context.

Qualitative Example of Relegation Relation:

Goal "Meeting scheduled" with a subsequent quality constraint as "Meeting scheduling will be done in 1 day". For instance, at run-time while the user is traveling encounters a flight delay. It does not mean that the goal becomes invalid, rather the conditions to achieve the same goal requires to be relaxed. This means, evaluating the new available information i.e. context conditions i.e. "At airport" and "Flight Delayed", the goal of the user e.g. "Meeting scheduled" cannot be achieved with the existing solution i.e."Flight to Destination", so it must be relaxed.

In this case by applying **Rel**, either the solution needs to be replaced that operationalizes the goal or an instance of the same goal with revised conditions is linked using **Rel** with the original goal e.g. "Meeting scheduled in the Morning" relaxing the quality constraint it has i.e. "Meeting scheduling will be done in < 1 but > 2 days" such that the preference of the end-user is not disturbed i.e. "prefer morning meetings". In this example, the instance of the original goal is not compromised rather relegated to a new goal with different conditions to achieve it. An alternative plan is generated i.e. "Book flight in morning" to operationalize this new auxiliary goal that stands justified to the previous solution i.e. "Book flight", thus changing the parameters in the solution.

Finally, we define the influence relation. Note that it is simple here, since we have no numerical values, so we cannot speak about influence as correlation. We can only that some information influences some other information if the absence of the former makes it impossible for us to satisfy the latter.

Definition 20 ***Influence relation:*** *An influence relation (Inf) is a binary relation from $\psi \in \mathcal{L}_{SAS}$ to $\phi \in \mathcal{L}_{SAS}$, iff either*

1. $\exists \Pi \subseteq \Delta \cup C,\ \Pi \mathrel{|\!\!\sim_{\mathcal{CT}}} \phi$ *and* $\Pi \setminus \psi \mathrel{|\!\!\not\sim_{\mathcal{CT}}} \phi$, *or*
2. $\forall \Pi \subseteq \Delta \cup C,\ \Pi \mathrel{|\!\!\sim_{\mathcal{CT}}} \phi$ *and* $\Pi \setminus \psi \mathrel{|\!\!\not\sim_{\mathcal{CT}}} \phi$.

In the first case above, we say that ψ weakly influences ϕ, denoted $\psi \xrightarrow{wi} \phi$. In the second case above, we say that ψ strongly influences ϕ, denoted $\psi \xrightarrow{si} \phi$.

remark 15 *If $\psi \xrightarrow{si} \phi$, then we have no way to satisfy ϕ if ψ is not satisfied. If $\psi \xrightarrow{wi} \phi$, then some ways of satisfying ϕ cannot be used to do so if ψ is not satisfied.*

The purpose for Influence relation is to assess the impact of change in existing concepts and their relationships. An influence relation (**Inf** hereafter) is needed to analyze the impact of change on the instance of concepts with in Core. This means, if change in the operational environment or in end-user need at run-time causes a change in the instances of the Core, SAS at run-time needs to be able to ascertain the impact of this change on the existing specification before adapting to an alternative solution. This idea of introducing this relation has been initially argued in our recent work on design methodology for real services [MNF$^+$10], where the dynamic semantics of process activities is specified as OCL constraints to analyze the impact of their execution in modifying assets (a type of intangible resource).

Making explicit influence relationship type provides analysis capability to ascertain the impact of change either motivated by the operational environment, system itself, or by the end-user. At design-time, an engineer performs this analysis as per their expertises by anticipating the dynamic changes that could lead a change in run-time behavior. For instance, soft-goal contribution links provides analyst to score the alternatives and to determine to which level that are able to satisfy the goal. This form of analysis is however qualitative. We stand independent of this, and provide influence relation, that later can be formalized using any formal definition to quantify the impact. For example, goal "Payment for Travel Ticket" strongly influence the goal "Generate Itinerary". To formalize this, one can either link softgoal contributions or could add inhibition relation among goal that can be validated once the system is built and deployed. Moreover, this has also favorable consequences at run-time. Enabling SAS with this analysis capability could result in justifying the adaptation and providing meaningful feedback to the end-user by deriving conclusions on user's goal satisfaction.

Qualitative Example of Influence Relation:

Influence relation (**Inf**) determines the process of internal assessment over the dependencies among the concepts in core ontology. For example, if the user books the business flight to attend the meeting and confirms her travel itinerary, any subsequent change has consequences over the existing instances (e.g. goals). The change e.g. change in leg of the flight must not invalidate the satisfaction of goal "Book Itinerary" and on a higher level goal "Travel for business". The SAS needs to ascertain the impact of user's request i.e. "Change flight leg" on the existing goal "Book Itinerary" and on the resource "Travel Itinerary" and asset "Money".

In this case, the new goal "Change Itinerary" must be analyzed with respect to new information i.e. change in operational context, existing goals, resources and their operationalizations. In this scenario, it can be ascertain by the SAS taking into account the dependencies exist in the specification e.g. goal tree. Identifying the potential dependencies, it can associate consequences in achieving this new goal e.g. “flight change is eligible” or not by taking into account the domain assumptions associated with the goal “Book Itinerary” i.e. ticket change implies to a charge of amount 100 Euros. Similar dependencies can be collected and subsequent consequences are ascertain by the SAS to analyze the impact of changed solution. Moreover, in doing this what impact of the new solution that operationalize the new goal will have on the resources i.e. “Travel Itinerary” and asset “Money” e.g. how much compensation has to be paid to change the flight. Similarly, in case of removal of an instance the impact of this change on the specification can be derived. **Inf** is, however, critical to ascertain the quality of the adaptation process and evaluating the satisfaction of the user’s needs.

3.5 Run-time Requirements Adaptation Problem: Illustration

We now revisit the above definitions and using scenarios from travel exemplar case study to illustrate how SAS, instantiating CARE and running on user’s mobile phone, resolves the “run-time requirements adaptation problem” at run-time. Given the requirements problem in [JMF08, JBEM10b]

$$\mathbf{K}^*, \mathbf{T}^* \mid\!\sim_{?} \mathbf{G}^*, \mathbf{Q}, \mathbf{S}^{\mathbf{M}}$$

Where $\mathbf{K}^*$ is the *believed content* of the specification, $\mathbf{G}^*$, $\mathbf{Q}$, $\mathbf{S}^{\mathbf{M}}$ are *desired contents* in the specification as communicated by the stakeholders, such that to arrive to a specification $\mathbf{T}^*$, which brings about the state of the world in which $\mathbf{G}^*$ are satisfied without violating $\mathbf{K}^*$, and by assuring $\mathbf{Q}$. The reason for having a non-monotonic consequence relation is that the acquisition of new knowledge at run-time requires revision of solution space, so that inference is

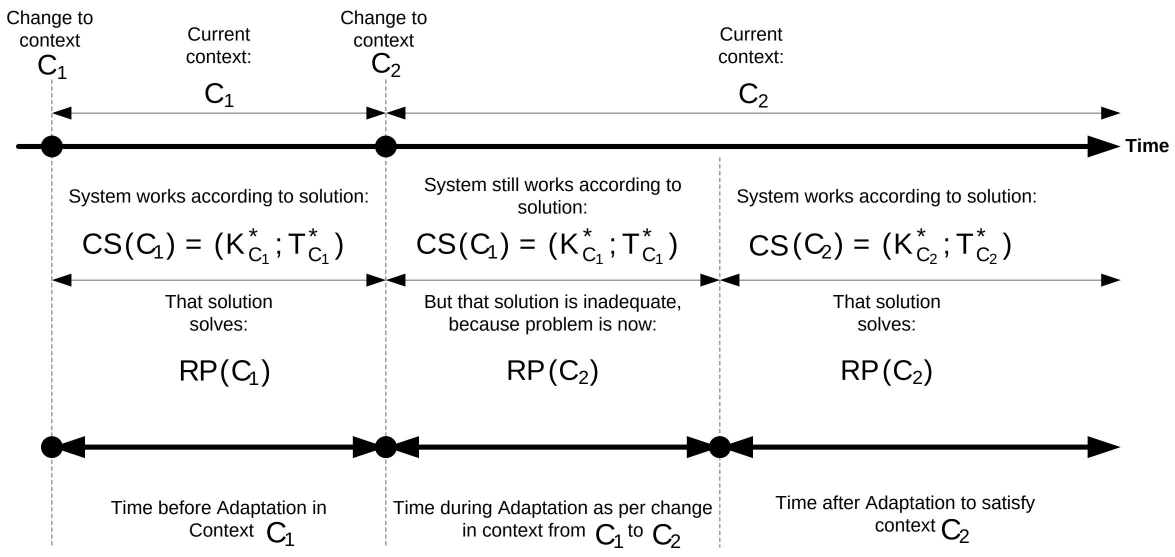

Change to context C_1
Current context: C_1
Change to context C_2
Current context: C_2
Time
System works according to solution:
$CS(C_1) = (K^*_{C_1}; T^*_{C_1})$
System still works according to solution:
$CS(C_1) = (K^*_{C_1}; T^*_{C_1})$
System works according to solution:
$CS(C_2) = (K^*_{C_2}; T^*_{C_2})$
That solution solves:
$RP(C_1)$
But that solution is inadequate, because problem is now:
$RP(C_2)$
That solution solves:
$RP(C_2)$
Time before Adaptation in Context C_1
Time during Adaptation as per change in context from C_1 to C_2
Time after Adaptation to satisfy context C_2

Figure 3.2: System Adaptation Sequence in Time

defeasible and existing specification $\mathbf{T}^*$ (solutions) must be revised as per new information.

For example, user arrives at the airport to avail her flight from Italy to Canada via Paris for a business meeting. While at the airport after the boarding, user want to connect to the Internet using her mobile phone to check emails and flight details before checking in for the plane. Moreover, user wants to be informed about any flight delay.

Taking the above example, we now present SAS adaptation sequence at run-time in case of change in context C along the time $T = t^1, ...t^n$ as shown in the Fig.3.2. Let CS be a set of candidate solution, thereby determining the run-time requirements adaptation problem as a combination of instances of the tasks $\mathbf{T}^*$) and domain assumptions $\mathbf{K}^*$ such that $\mathbf{G}^*, \mathbf{Q}^*$ and $\mathbf{S^M}$ are satisfied. In case of changes in the context $C = C_1, ...C_n$ overtime for which CS needs to be re-evaluated by the system and R is required to be used or identified in a given context C to realize CS. By re-evaluation we mean that system at run-time exploits its monitored information, evaluate all the possible alternative CS or search for new ones (i.e. exploiting available services) that can satisfy the run-time adaptation problem in response to changes in the C therefore adapting to the candidate solution CS. At this SAS may perform at sub-optimal level and can exploit automated reasoning techniques such as planning could be applied to sort and select the candidate solutions with respect to given preferences. We present three scenarios to illustrate how the SAS can adapt at run-time by resolving the run-time requirements adaptation problem.

Before Adaptation: Lets consider that at time t^1, to satisfy the user's goals $\mathbf{G}^*$ are *to connect to the Internet for checking details of itinerary* and *inform about the flight delays* with the quality constraint **Q** is *to have the Internet connectivity not less than 256Kbps*, SAS is exercising its candidate solution CS i.e. set of tasks $\mathbf{T}^*$, e.g. *search for available connection*, *enable Wifi*, *get itinerary details* and *show flight itinerary*, in the current context i.e. C_1 is *at the airport*, and the domain assumption i.e. K^*_{t1} e.g. *Internet must be available at the airport*, is not violated. This implies that, $CS(C_1)$ = (K^*_{C1}, T^*_{C1}) and $CS(C_1)$ satisfies the run-time requirements adaptation problem i.e. $RP(C_1)$. We can rewrite this as:

$$C_1, \mathbf{K}^*_{C_1}, \mathbf{T}^*_{C_1} \mid\!\sim_{\mathcal{CT}} \mathbf{G}^*_{C_1}, \mathbf{Q}^*_{C_1}$$

where $\mathcal{R}(C_1 \cup \mathbf{K}^*_{C_1} \cup \mathbf{T}^*_{C_1})$ identifies the set of resources **R** available e.g. (Airport WiFi hotspot, Mobile Phone of the user) to perform $\mathbf{T}^*_{C_1}$.

During Adaptation: SAS while executing $CS(C_1)$, monitors a change in context i.e. *the airport WiFi connection becomes unavailable* at time t^2. Due to this change in context from C_1 to C_2, the existing candidate solution $CS(C_1)$ might be valid but is not adequate to satisfy the current context C_2. As a consequence, the SAS needs to re-evaluate its candidate solutions CS by searching in its solution base i.e. Δ or looking for solutions that can be realized through available services. For instance, a new candidate solution $CS(C_2)$ could be e.g. *connect to the Internet using data services either 3G or*

Edge on mobile phone R; or *recommending user to move to the area where the signal strength is stronger*; or *avail the Internet on the free booths*. At this stage, SAS may use relegation relation to infer, if the $\mathbf{G}^*$ with a $\mathbf{Q}$ is *to have the Internet connectivity not less than 256Kbps* can be relegated. After re-evaluating the possibilities, SAS finds $CS(C_2)$ i.e. set of tasks $\mathbf{T}^*$ e.g. *enable 3G or Edge service* and *connect to the Internet* with a refined $\mathbf{Q}$ i.e. *Internet connectivity greater than 256Kbps* for the user. At this stage the influence relation is also used to ascertain the influence of $CS(C_2)$ on user's goals and preference e.g. *Hi-speed Internet is preferred than no Internet connection*. SAS can derive conclusions that adapting to $CS(C_2)$ will not affect K^*_{C2} i.e. *Any flight information must be communicated to the customer* and goal $\mathbf{G}^*$ i.e. *to connect to the Internet to view itinerary* and *inform about the flight delays* will be satisfied. Therefore, $CS(C_2) = (K^*_{C2}, T^*_{C2})$; satisfying the run-time requirements adaptation problem i.e. $RP(C_2)$. We can rewrite this as:

$$C_2, \mathbf{K}^*_{C_2}, \mathbf{T}^*_{C_2} \mathrel{|\!\!\sim_{\mathcal{CT}}} \mathbf{G}^*_{C_2}, \mathbf{Q}^*_{C_2}$$

where $\mathcal{R}(C_2 \cup \mathbf{K}^*_{C_2} \cup \mathbf{T}^*_{C_2})$ identifies the set of resources $\mathbf{R}$ available e.g. (Access 3G or Edge data services, Mobile Phone of the user) to perform $\mathbf{T}^*_{C_2}$.

In Fig.3.2, the initial solution $CS(C_1)$ becomes invalid due to change in the user's context (e.g. Home to Airport) or change in the environment (e.g. Swear rain caused all the flights to be delayed), i.e. $\neg\, CS(C_1)$ (e.g. flight service becomes unavailable), and the system requires to re-evaluate other instances of the specification. At this stage, relegation relation can be exploited to analyze the alternative ways to satisfy the user's goals, thus the new information is observed to relegate the requirement to another with an alternative solution ($CS(C_1)$ with different set of $\mathbf{T}^*$) to achieve the same goal.

During Adaptation (Alternative Scenario): At time t^2, during the adaptation phase the SAS re-evaluates its CS solution based (or requirements database i.e. Δ), to find a solution set that satisfies the user's goals. A candidate solution could be $CS(C_2)$: *to provide a directed map to the user for* *shopping at airport;* or $CS(C_3)$: *a possibility to stay at courtesy lounge of the airline*. In other case, **S**^{4}: *user flight from Paris to Amsterdam can be changed/canceled and a new itinerary is created using Thaly's train online reservation*. The system re-evaluates the most candidate solution by ranking them taking into account the user's preferences i.e. **A**. For example, $CS(C_4) \succ CS(C_3) \succ CS(C_2)$. At this stage the influence relation is used to ascertain the influence on changing the itinerary (R) on goals and/or on the set of quality constraints or preferences. For instance, SAS can derive conclusions that adapting to $CS(C_4)$ leads to change in R (i.e. Itinerary) in case of C_2(i.e. user at Airport), where $(K^*_{C2}$ (i.e. Any Euro-flight ticket can be transferable to any Euro-rail economy or first class ticket) does not violates and affects of $CS(C_4)$ to be the candidate solution for the achievement of **G*** (i.e. Travel for Business) and **Q*** quality constraint (i.e. reach destination not later than 3 hours) and **S*** (preference i.e. Hi-speed train is preferred than waiting at airport).

After Adaptation: At time t^3, SAS adapts to the candidate solution $CS(C_2)$ taking into account the context C_2 and available resources R i.e. Access 3G or Edge data services, Mobile Phone of the user, thus not violating the K^*_{C2}. Adaptation is performed dynamically at run-time by changing (e.g. switching, re-configuring, optimizing) SAS from one requirements problem to another i.e. $RP(C_1)$ to $RP(C_2)$, in response to changes in the context, user's needs or resource variabilities.

3.6 Benefits & Related Work

Engineering of SAS aims to resolve run-time requirements adaptation problem by finding a candidate solution in response to changing context, resource variabilities and emerging end-user's needs. Intuitively, resolving such problem amounts to an optimization problem, where admissible set of candidate solutions (e.g. realized through available services) need to be stored in the requirements database such that at run-time they can be made available at run-time when SAS identifies potential changes in the external environment. This enables SAS to first, monitor its available information characterized as the core elements such that it is able to reason on them taking into account the contextual changes and existing requirements.

The introduced concepts (context and resource) and relations (relegate and influence) are general enough to help the analyst at design-time to formulate competing set of requirements with supported candidate solutions in the requirements database, thereby we could call them configurations or set of alternative specifications.

Our point is not to show that we do something more efficiently than others, but to give a *general* framework in which we can represent and understand the key concepts, relations, and problems to be solved in the RE of SAS. For instance, in comparison with RELAX [WSB$^+$10], which was introduced as a formalism for the specification of requirements in a way that allows their relaxation (deidealization) at run-time, our Relegate relation is a more general relation than the RELAX operators, since we do not commit to fuzzy logic: we only ask for a way to represent alternatives and to compare them. RELAX is a particular way to relax requirements (a particular way to implement the Relegate relation), that obtains a straightforward interpretation in the language we used here, and thereby, the RELAX operators cannot capture information other than that which we can capture in the formalism presented. There are other ways to handle uncertainty and relaxation of requirements, and our aim is to remain independent of particular approaches. RELAX adopts a particular approach, and our argument is that we should first understand the general problem, and then focus on developing particular requirements modeling languages to handle it.

At run-time, after development of the intended SAS-to-be, it starts exercising the monitoring functions and decision making modules that are built and defined using requirements specification, SAS could reason on its own information at run-time. However, practical methods that SAS requires to perform non-monotonic reasoning at run-time to find candidate solutions are needed to support such kind of reasoning. It is obvious that using our general framework or defining the requirements problem for SAS provides more information to the SAS itself to reason on and to take into account while deciding for an adaptation action at run-time. There are various ways in which to search for candidate solutions e.g. search based RE and AI planning techniques, which are also promising in this regard. For instance, planning can be applied to sort candidate solutions, once the context conditions are stable and resources are identified.

3.7 Final Remarks

The problem formulation we proposed makes no assumptions and imposes no constraints on how the information used in the problem formulation is acquired. We thereby recognize that not all information can be collected during requirements engineering, or at design time, but that this will depend on the technologies used to implement the system. For example, the information about the context, the formulas in C may – if the implementation technology allows – be obtained by recognizing patterns in the data that arrives through sensors, then matching patterns of data to templates of proposition or implications. We stayed in the propositional case, since this was enough to define the main concepts and relations, and subsequently use them to formulate the run-time requirements adaptation problem. The actual system will operate using perhaps more elaborate, first-order formalisms to represent information, so as to make that information useful for planning algorithms applied to identify candidate solutions. However, regardless of the formalism used, the system still needs to be designed to ensure the general conditions and relations that the problem formulation states: e.g., that the system needs an internal representation of information pertaining to contexts, domain assumptions, tasks, goals, and so on, that goals and quality constraints are satisfied through consistent combinations of C, K and T, among others. In other words, we may change the formalism, but the principles on what information we will be dealing with, and what relations it should stand in, when engineering SAS does not change: we will still have goals to satisfy, preferences to take into account, domain assumptions, and context conditions that we should not violate.

Chapter 4

Continuous Adaptive Requirements Engineering (CARE) Framework

4.1 Overview

In this chapter we introduce the Continuous Adaptive Requirements Engineering (CARE) Framework. The CARE framework is proposed to support continuous requirements engineering by the system itself involving the end-user at run-time. The rationale behind CARE is that changes occur continuously in the operational environment in which the system and end-user is part of. The only way to understand what changes are acceptable in a system is with respect to its requirements, and more specifically, end-user's intentions. For this reason, CARE prescribes continuous refinement of requirements at run-time to support the dynamic RE problem defined in the previous chapter. We provide a clear distinction between RE performed at design-time involving the human user i.e. analyst/designer, and RE performed at run-time by the SAS itself involving the human user i.e. end-user (if needed). Actually, CARE prescribes adaptation types that are defined with respect to the level's of RE for SAS envisioned by Berry et al. in [BCZ05], which provides a way to reason about adaptation at design-time and run-time. A conceptual architecture is defined to support this method in CARE. The architecture is general to the extent that it is not intertwined with any specific technology or technique.

In fact, CARE is thought for software-intensive systems specifically that are expected to operate continuously, such as service based applications (SBA) [DNGM$^+$08]. SBA are one particular kind of SAS as they operate in an open environment (i.e. the Internet) and relies on the available services (built by third party). These services are used to operationalize not candidate solution at design-time (i.e. during the specification of requirements) but also to furnish end-user needs at run-time (i.e. when new service is preferred over another to resolve a new requirements problem). In this context, SBA need to adapt continuously in response to dynamic changes either mediated by the environment or through end-user needs. Thus, CARE is proposed to bridge the gap between RE activities that are performed at design-time and at run-time.

The rest of the chapter is organized as follows. In Section 4.2, we present key underlying adaptation types that are defined in line with the four levels of RE for SAS by Berry et al. in [BCZ05]. In Section 4.3, we distinguish between RE at design-time and RE at run-time clarifying the role of the human user, nature of the requirements and type of analysis. In Section 4.4, we define the conceptual architecture based on SBA discussing the CARE framework at run-time. In Section 4.5 we compare our proposed framework with the existing proposed frameworks to highlight the potential benefits of CARE. Finally, in Section 4.6 we summarize the key contributions of CARE framework that extend the state of the art.

4.2 RE levels and Adaptation Types

We elaborate the distinction between RE activities at design-time and at run-time with the help of adaptation types. These types are defined based on the levels of RE proposed by Berry et al. in [BCZ05]. Traditionally, RE activities are performed by the analyst/designer at design-time taking into account the end-user's needs. In these activities, design decisions are taken anticipating the run-time. This limits the scope of the SAS under construction, when it is deployed in real environment. Changes in the operating environment are expected to occur continuously, which may or may not happen in an unanticipated and uncertain way. Thus, SAS while running must be able to cope with these changes and adapt to the new candidate solution that resolves the dynamic requirements problem. To fulfill this aim, CARE prescribes adaptation types that are defined enabling SAS to perform RE activities by itself involving the end-user (if needed) at run-time.

The classification discussed in [BCZ05] proposes four level of RE for SAS (called dynamically adaptive systems in that paper). Consider (after [BCZ05]) a system realized in (alternative) programs $S \in \mathcal{S}$ for a given set of (anticipated) domains $D \in \mathcal{D}$. Level 1 RE is performed by the designers to produce a set of programs $\mathcal{S}$, anticipating a set of possible domains $\mathcal{D}$. Level 2 RE is performed by the system itself, at run-time, to select between programs as the operating environment changes. Level 3 RE is done to update the requirements themselves in response to unanticipated changes. Finally, Level 4 RE refers to research on RE in DAS (as proposed in this chapter). Aside: the difference between autonomic or SAS and requirements-aware systems [SBW$^+$10] is that the former does not consider requirements explicitly, while the latter always makes reference to a consistent and contemporaneous requirements model.

We believe there are three different tasks involved in requirements-aware systems: one, how to change the current S to a new S', with respect to the system requirements – i.e., the mechanics behind switching programs; two, how to select which new S' to choose, with respect to the requirements – i.e., the decision problem; and three, corresponding to Level 3, how to update the requirements themselves when there is no available $S \in \mathcal{S}$ which responds to the new D – the requirements problem.

Based on this we introduce our classification of adaptation types that CARE prescribes (i.e. an adaptive SBA, which instantiates CARE) to use for adaptation reasoning at run-time. For simplicity sake, we use SAS to refer to adaptive SBA.

Type 1 consists of changes which are anticipated at design-time and for which alternatives candidate solutions already exist in the specification, and the best alternative is obvious. This type conforms to Level 1 in [BCZ05].

Example: *During the World Cup, Twitter's servers become unresponsive. The manager service automatically offloads jobs to an Amazon content distribution network.* Type 1 events are familiar and anticipated. This is the classical autonomic computing scenario.

Type 2 consists of changes in the environment e.g. context, preferences etc. The SAS must monitor such changes; after evaluating the change (e.g. variation in operating or user context, or availability of the resources, or change in end-user preference) it adapts to the most feasible alternative solution (which might not be in existing specification) using different available services. Here, the SAS mainly exploits the alternative candidate solutions by being aware of its user's goals and preference. Alternatively, it looks for similar solutions, that can satisfy the change per se. This type relates to Level 2 in [BCZ05].

Example: *A SAS knows the user wants a four-star hotel near the beach (monitor e.g. preferences), but cannot optimize both variables simultaneously. It tries to book the cheaper three-star hotel near the beach, which also satisfies the user's cost preference, or it proposes to her a four-star near the city center.*

Type 3 consists of changes which are unanticipated by the designer, due to partial or uncertain knowledge about the environment. The events are also unfamiliar to the end-user. However, the system can query the end-user for new requirements and attempt to replan dynamically to accommodate them. The SAS relies on a shared ontology to infer similarity and techniques to find the appropriate service satisfying the new or emerged requirement. This type relates to the Level 3 in [BCZ05].

The main difference with respect to Level 3 in [BCZ05] is that our framework involves the end-user in specifying new requirements (where *end-user* is seen as distinct from the *analyst/designer*). The system itself analyzes the end-user's context variation as a refinement of existing requirements or an entirely new requirement. Once an appropriate solution is found, this new requirement is updated in the original specification at run-time. When there is no solution found, the system opts for Type 4. In Berry et. al.'s classification [BCZ05], Level-3 RE is performed before Level-2, but the order can be changed. In our case, we perform both Level-2 and Level-3 concurrently unless we have to do offline maintenance or evolution (our Type-4).

Example: *Julia reaches Oslo, but all hotels are full, and the system specification does not include the alternative of searching for home-stays. At this point, the system prompts Julia to say that there is no suitable itinerary, and asks for possible work-around. Julia informs the system that a home-stay (a sub-concept of* `Accommodation`*) is possible, and the system attempts to find a service which can handle this (e.g, using swoogle.umbc.edu).*

In the above example, user specifying an option of "Home-Stay" is considered as a new requirement for the system to operationalize. Such requirement is an alternative to users' high level goal "Find Accommodation". In service-oriented RE (SORE) [TJWW07], user requirements are used to discover services either at design-time or at run-time to operationalize them. The requirements analysis is conducted with a pre design-time knowledge about the existing available services. Requirements model requires a continuous revision as new services are discovered to operationalize them, differently as in case of object oriented analysis and design.

Type 4 consists of Type 3 events for which there is *no* possible addition or simple refinement. A new specification and the relative new SAS functionalities are generated. In our view of requirements-aware systems, this is the worst-case scenario.

We term Type 4 changes *system evolution*, as distinct from *system adaptation* (Types 1-3). As a further investigation on Type 4 changes, which is described in the future work, we have investigated a framework for dealing with evolving requirements. A proposed framework is presented in [PQS$^+$11]. This is roughly analogous to biological systems, where adaptation reflects the ability of an individual to cope with environmental variance, while evolution is a widespread genetic change to prolonged stimuli. One should be careful not to over-extend this metaphor, however, as biological systems are not designed nor does evolution progress towards 'better' designs.

4.3 RE at Design-Time Vs RE at Run-Time

The two RE processes supported by CARE both at design-time and run-time with their interdependencies respectively are shown in Fig. 4.1. At design-time, the analyst/designer exploits the stakeholders needs and performs analysis to formulate the requirements problem i.e. finding the specification that can resolve the requirements. At run-time, we propose that RE must be performed by the SAS itself involving the end-user to cope with the continuous changes. The conceptual difference between these processes are based on: (1) the involvement of human user, (2) role of the SAS itself and (3) the nature of the requirements that are kept "alive" both at design-time and at run-time.

4.3.1 RE at Design-Time

While designing SAS, the role of the analyst/designer (human) is critical to perform RE activities starting from the elicitation of stakeholder's intentions (or from market studies in case of a market-driven development), and later to specify the requirements of SAS (see Fig. 4.1, upper part). Additional input

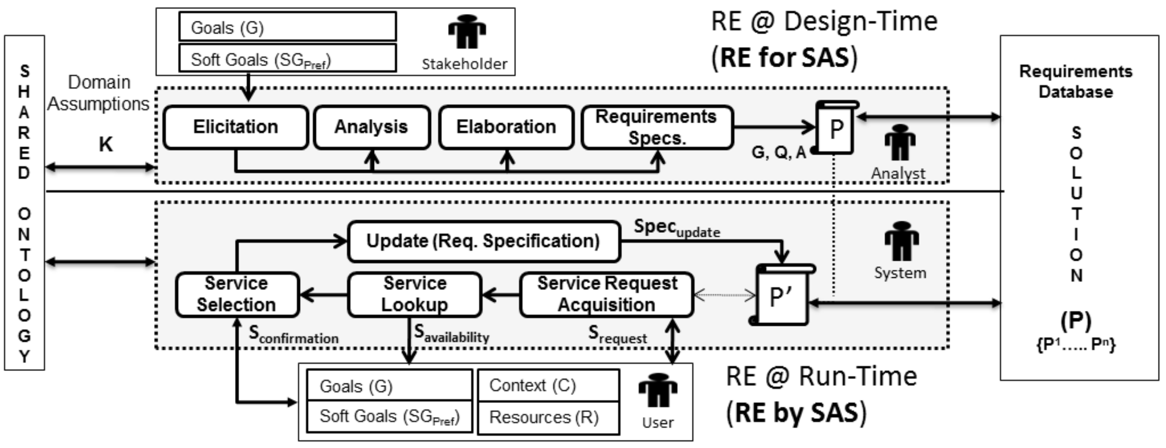


Figure 4.1: RE process at Design-time and Run-time

to the RE process at this stage is domain knowledge or domain assumptions, represented as **K** in Fig. 4.1, upper part. The core ontology of RE for SAS (introduced in previous chapter) is instantiated using the domain knowledge (e.g. Travel domain). Below we present the nature of the requirements at design-time.

Requirements at Design-Time

Requirements are expressed as Goals and quality constraints that may be mandatory or optional, and preference relations can be defined over them (A). A requirement problem is represented as a goal model with labeled nodes and edges, which captures instances of the concepts and relationships of the core ontology of requirements for SAS. A solution consists of a collection of tasks and domain assumptions that are valid in a context, such that, when the tasks are executed they entail the satisfaction of all mandatory goals and maximize satisfying the optional ones.

Different solutions can be associated to a requirements database (Δ). They may be classified in terms of preference and optionality requirements. We use $\mathcal{L}_{\mathbf{SAS}}$ (defined in Chapter 3) to support expressing stakeholder goals and preferences, combined with a solution repository.

Design-Time RE process

An analyst follows a systematic approach to model requirements and specify requirements of SAS using the concepts and relations introduced in $\mathcal{L}_{\mathbf{SAS}}$ language. The core concepts of $\mathcal{L}_{\mathbf{SAS}}$ are described in the previous chapter. Below we describe the conceptual process to highlight the key aspects of CARE's design-time activities as shown in the Fig. 4.1 upper part.

Elicitation: is performed using our $\mathcal{L}_{\mathbf{SAS}}$ language constructs (defined in the previous chapter), which represents the core concepts defined in the core ontology of requirements for SAS. The analyst captures the relevant information from the stakeholders. Following our requirement problem definition, design-time requirements are elicited and formalized into a set of goals, quality constraints and preferences (optional requirements) and domain assumptions that determine the stakeholder needs.

Analysis: is performed to analyze the requirements. Here the relations defined in the core ontology of requirements for SAS are used. Mandatory and optional goal and quality constraints are refined into tasks that operationalize the high-level goals using inference relation. This results into a goal graph where multiple goal refinements are analyzed with respect to their satisfaction criteria. Information about the softgoals is added to analyze competing requirements that can influence or be influenced by each other.

Elaboration: is performed to further extend the requirements model with more precise information. With this premise, the requirements in requirements problem are used to describe system's properties. They include not only functional or non-functional requirements but also monitoring specification, evaluation criteria and adaptive actions specified as tasks. In fact, tasks are specified to ensure that the system's behavior meets them when operating in a dynamic environment. The dynamic information pertaining to the system or end-user is added using the concepts: context and resources. Relations among requirements are refined to avoid any conflicts. The domain knowledge is used to elaborate the terms used to specify context and resources. This done when the core ontology is instantiated by the domain information.

Specification: is performed by the analyst by finding sets of tasks and domain assumptions which can make *mandatory* requirements hold. Design-time requirements are represented in a goal model after performing the goal analysis and elaboration resulting into a specification of requirements. A specification P (also known as a 'solution' to the requirements problem) is found and stored in the requirements database Δ. Such specification can be used by the systems at run-time as a *Live Artifact* to reason for adaptation and to re-plan, if necessary. Each requirement encompasses maximum possible alternatives that are considered at design-time, i.e. it defines Type 1 adaptation. Also an incremental revision can be applied, enables managing Type 3 adaptation situations.

Example

Here we use the example from the Travel companion case study introduced in Chapter1. The mandatory goals of the end-user are Travel on Business and Business Successful, where as Provide Assistance Service for the SAS. Below we present the kind of RE performed by the analyst to structure the requirements for the SAS-to-be.

Given the elicited information or otherwise acquired, an analyst structures this information as goals e.g. *Travel Companion SBA*. The resulting goal-model reveals the main goal of supporting its end-user *Provide Service Assistance* is AND-decomposed (using inference relation) into the subgoals *Service Request Created*, *Service Data Collected*, *Service Selection Made*, *Update Specification*, *Service Delivery Ensured*, which have been in turn be further decomposed. For instance, the goal *Service Request Created* has been OR-decomposed (using inference relation) into two alternative subgoals: *Service Request Created by User* and *Service Request Created by SBA*. This captures the fact that a new requirement at run-time can be either gather from the user or by the SBA itself, due to either a context change or an event.

The goal *Service Request Created by User* has been further OR-decomposed (using inference relation) into two sub goals: *Preferences Changed* and *Required Service Searched*, both contributing positively to the soft goal *Convenience*. This makes clear that the new requirement can be constructed using the change in users' preferences or by looking up for a required service. Similarly, the goal *Service Request Created by SBA* is OR-decomposed (using inference relation) into two sub goals: *Due to Change in Context* and *Due to Change in Events*, both contributing positively to the soft goal *Timely Information*. Here the new requirement is generated by the information coming from the change in the context or by the occurrence of any unanticipated event.

To lookup for the service based on the new requirement, the goal *Service Data Collected* is AND-decomposed (using inference relation) into two subgoals: *Candidate Service Selected* and *Check Available Services* with a positive contribution to the soft goal *Efficiency*. The service availability is conditional, as the required services may not be available when the look up is performed. The service selection can be performed automatically, depending upon the context or it can be selected by the user, thus the goal *Service Selection Made* is OR-decomposed into two subgoals: *Automatic Selection* and *Manual Selection* contributing positively to the soft goal *Timely Information*.

Finally, upon selection of an appropriate service, goal *Update Specification*, AND-decomposed (using inference relation) into two subgoals: *Refine Exist. Req.* and *Add New Req.* subsequently contributing positively to soft goal *Flexibility*. The update is either refining the existing requirement of adding new service request into the initial specification. The goal *Service Delivery Ensured* is OR-decomposed into two subgoals: *Service Composed* and *Service Orchestrated*, both contributing positively to the soft goal *Availability*. As the composition or orchestration is performed the service delivery must ensure the availability or the existing services, thus making the SBA more robust.

4.3.2 RE at Run-Time

At this stage the main difference is the role of the analyst, which is played by the SAS itself. The key stakeholders are end-user and the SAS itself, as depicted in the Fig. 4.1, lower part. At run-time, SAS performed the RE activities to cope with the dynamic changes (e.g. change in end-user needs and preference, context variations and availability of resources). The process is dynamic as it takes place at run-time. Below we clarify the nature of the requirements at run-time.

Requirements at Run-Time

At run-time, new requirements are captured or acquired, given by the user or otherwise gathered as an information by the SAS while monitoring the environment or the end-user. Requirements at run-time are defined as ***service requests***. The dynamic aspect at run-time are captured by the system through monitoring or new requests may be raised by the end-users. Service request includes: functional goals of the user, quality constraints or preferences, context information (user or operational), resource information and other facets as shown in Fig. 4.2. The concept of *Service Requests* has been introduced to manage "requirements at run-time". To use this, we define a machine-readable format to express such requests in XML. Fig 4.2 shows such a Run-time Requirements Artifact (RRA, hereafter). RRA can be expressed by the user (switching to Type 3 adaptation) or it can be generated by the system taking into account the dynamic changes in the environment through monitoring (switching to Type 2 adaptation).

The RRA is evaluated at run-time to look for available solutions (e.g. competing services) to the requirements problem from the Δ. In case, there are not solution in Δ, SAS looks up for a relevant service that can best satisfy the intended RRA. This RRA is stored and used to update the initial specification, if new requirements are identified or new competing services

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <Request xmlns="http://www.example.org/Request">
     <RequestDescription>
        <Source></Source>
        <Func_Goal></Func_Goal>
        < Alternative Func_Plan></Alternative Func_Plan>
       <Context></Context>
       <Quality_Attribute></Quality_Attribute>
       <Preference></Preference>
       <Stimulus></Stimulus>
       <Date></Date>
       <Time></Time>
       <Response></Response>
       <Resource></Resource>
     </RequestDescription>
</Request>
```

Figure 4.2: Run-time Requirement Artifact (RRA)

are selected.

The key benefit of translating and acquiring a end-user's requests (RRA) as requirement is two-fold. First, the end-user is involved in the RE process enabling the SAS to (1) get new requirements (new services to make use of, existing services to drop); (2) refining existing requirements, which just requires an alternative solution to be added with the existing one to satisfy them; (3) to accommodate user's preferences over time by refining the selection of services for quality reasons. Secondly, the SAS (e.g. Travel Companion) does RE on its own for the user. It provides an optimal solution to the user based on her preference e.g., minimum cost and convenience, context (profile, location, operational) and resource variability e.g. mobile device; or changes that might occur due to unanticipated events e.g. flight delays.

Run-Time RE process

We now describe the CARE's key RE activities that are performed by SAS at run-time. It includes: Service Request Acquisition, Service Lookup, Service

Selection and Update Specification along with their respective input/output operations, i.e. $S_{request}$, $S_{availability}$, $S_{confirmation}$ and $Spec_{update}$ as shown in lower part of the Fig. 4.1. We instantiate an specification instance i.e. P' of P from Δ, which is an input to the CARE activities.

Service Request Acquisition ($S_{request}$): is performed to acquire service request either as a result of monitoring users' context and/or preferences or expressed as a search query in natural language (using keywords or a whole phrase) by end-user. Once the service request (RRA) is acquired from the user, the existing specification P' is compared with the given request to identify the operation to be performed (switch to Type 3). Specific operations are defined to enable the SAS to reason for refining its requirements. For example, add a new goal; add a new task; substitute, or relegate an existing goal or task that satisfies the existing goal – these operations are defined in the following Chapter 6. Note, that each operation has consequences such as looking up a new service or reconfiguring the existing ones. Alternatively SAS generates a service request on its own as a result of monitoring, if any change in context conditions or violation of a goal achievement condition is detected (switch to Type 2). In this case, SAS opt to take corrective actions or propose set of actions to the user in case of error interpreting her input (switch to Type 3). A run-time requirement artifact (RRA) is filled to lookup relevant services that operationalize the given request.

Example: *In the scenario, the main goal of the user is Find Accommodation. Due to a flight delay in reaching Amsterdam, the hotel booking has been canceled. The SAS propose some alternative accommodations which are not satisfactory. The user then generates a RRA Stay at friend's Place by providing her friend's address. This option was not perceived during designtime and is treated as new requirement, which requires an update to the original goal specification.*

At run-time, SAS looks for more than just a single optimal solution to a requirements problem. It instead presents a set of solutions, which helps managing uncertainty. Possible alternatives candidate solutions are specified for the SAS, which enables it to try them in case of possible changes at run-time. Actually, uncertainty is resolved by monitoring and switching tasks. In the above example, we describe a case where the user changed her accommodation goals to stay with a friend. This new requirement is added to the model as a RRA at run-time, and the corresponding implication for the specification is either explicitly given ("I will call her") or obtained from a service registry ("Amsterdam White Pages"). The solution with the task Reserve Online is removed and returned to the repository, and the new task added to the specification P'.

Service Lookup ($S_{availability}$): This activity is initiated after analyzing the RRA (switch to Type 3) or as a result of monitoring the performance of an existing service (switch to Type 2). Lookup is performed either in the existing pool of services of the SAS or using a web service search, e.g. Woogle or Seekda. The lookup operation tries to match/compare user's keywords and service descriptions to locate available services by showing a list of possible services to the user or choose the most relevant competing service taking into account the users' preferences.

Example: *Arriving at Amsterdam airport, the SAS proposes alternative hotels nearby taking into account the current location context of the end-user and using Google places it shows possible hotels with respect to her cost preferences. Alternatively, it displays hotels using a service e.g. Booking.com*

Service Selection ($S_{confirmation}$): The SAS actively involves the end-user where necessary (switch to Type 3). The RRA is populated with a resulting list of services, which is shown to the user for selection and confirmation. In case SAS is proposing a service based on user's preferences e.g. *LowCost* or based on the competing advantage of an existing service performance e.g. *Response Time* (switch to Type 2), the best possible service is shown to the user for the confirmation. At this step, the user confirmation is used to refine the current request (RRA) i.e. adding details about the selected services that operationalizes the service request (RRA).

Example: *The system attempts to update the itinerary to accommodate flight cancellations. It proposes to the user a bus trip from Austria to Amsterdam. The user reject this and selects a train via Paris instead.*

Update Specification ($Spec_{update}$): Once the RRA has been completely filled using the users' input (switch to Type 3) or monitored information (switch to Type 2), this activity is triggered as a consequence. An update to the initial specification P' is ensured either through refinement of existing or addition of new requirements.

Example *The alternative sub goal is added Stay at friend's Place, which is operationalized by looking up services e.g. Google maps, and transportation service, motivated by the user's preference e.g., taxi is more convenient than bus and metro. This changes the cost preference. In this case, an alternative to the goal FindAccommodation is added, a change in preference e.g. Taxi $\succ$ Bus $\succ$ Metro is accommodated considering Convenience $\succ$ Cost. This then implies new services might be selected.*

To summarize, CARE framework's run-time process is performed by the SAS that instantiates it. Requirements model is reappraised and reconfiguration to SAS behavior is reasoned based on the adaptation types involving the user, where needed.

Illustration of Example

At design-time, analyzing the end-user's goals: Travel on Business and Business Successful. A combination of tasks and domain assumptions is a *candidate solution* for the problem of satisfying the mandatory goals. There are possibly many candidate solutions. For example, the set S_1: {Receive (Tickets) via Email, Book Plane, Reserve Online} is a candidate solution. An alternative is to replace Receive (Tickets) via Email with Receive

(Tickets) by Mail to produce S_2.

To sort candidates, we use a decision procedure for ranking them, taking into account the number of preferences satisfied and the number of options included. A candidate solution with strictly more of either satisfied preferences or included options is a preferred solution. Returning to the solution S_1, we see that since Receive via Email is preferred to Receive via Mail, S_1 dominates S_2. S_1 has one satisfied preference and one optional goal achieved (Convenience) while S_2 has none.

Preferences and options are incomparable, so this is a multi-criteria decision problem, and there may be a Pareto-front of equally acceptable solutions (e.g., solutions which satisfy one preference and two options, or two preferences and one option, respectively, are incomparable). One such solution is chosen by the system, and used as our initial specification P. The remaining candidate solutions are stored in a requirements repository (Δ). The idea is that as our SAS detects changes in the situation, other solutions (alternatives) become applicable, and another might be more suitable. Consider the case where a user is offline. In this case, the task Receive via Email is no longer possible, and the system will have to switch to another task to achieve the goal Obtain Tickets.

At run-time, the solution repository is searched by SAS (e.g. Travel Companion), and a solution which matches the current situation is selected (P). The system monitors the environment for changes (such as a delayed flight), and then makes system adaptations according to which Type the change impacts. In Types 1 and 2 adaptation, the existing solutions in the repository suffice for adaptation or in some cases (Type 2 adaptation) it can look for available services as per the service request (RRA). In Type 3 adaptation, the user can add requirements, including preferences, and SAS exploiting Δ re-evaluates the set of solutions incrementally. In Type 4 adaptation, the system is taken offline and we begin at the CARE's 'design-time' phase, so to speak.

There are two components to the run-time reasoning. In one, we use retrieval methods to find and retrieve requirements solutions from the repository. In the other, the system works with the user to identify new requirements or modifications to the model, which are then fed back into the model reasoner.

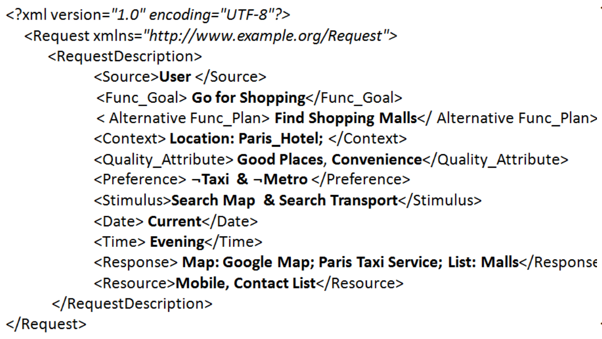
For instance, while the SAS is in operation, it detects that the user's flight was canceled. The system attempts to update the itinerary to accommodate flight cancellations. It cannot automatically shift plans (Type 1), since there is no obvious choice. It proposes to the user a train trip via Austria to Amsterdam. Since a (Type 2) event occurred, it attempts to replan using an existing alternative. Since the optional goal Ease of Travel is not satisfied, and yet the train will increase the price of the trip (π_2), the user reject this. As there are no remaining alternatives, the system shifts to (Type 3), and prompts the user for a run-time requirement. The user suggests booking a bus. Choosing to go by bus satisfies the Minimize Cost goal, although Ease of travel is not satisfied.

It should be noted that this example is similar to examples in the AI planning domain. Planning is a lower-level operation using goal operationalizations, devising a plan to satisfy them. Our RE reasoning is more high level, and involves design decisions about the system to be, maintaining a separation between the problem space and the solution space. In a low-level implementation, of course, both AI planning and RE reasoning are both search problems.

4.4 Conceptual Architecture of CARE

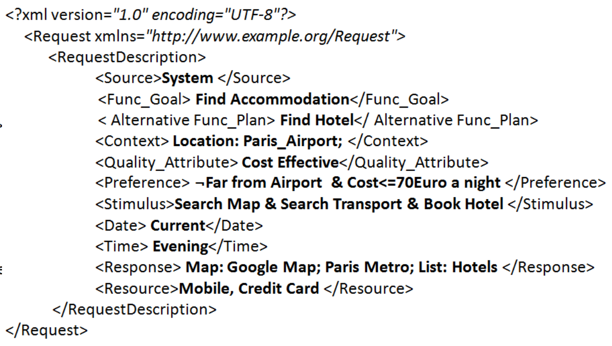
In Fig. 4.5, we present the architecture of a SAS, which realizes our *CARE* framework. We describe the key aspects and activities of the architecture using Travel companion case study examples as described below:

Example 1 *In Fig. 4.4, the system while monitoring users' itinerary detects that the flight has been delayed for 1 day in Paris. Having no other alternative*



```
<?xml version="1.0" encoding="UTF-8"?>
  <Request xmlns="http://www.example.org/Request">
     <RequestDescription>
           <Source>User </Source>
           <Func_Goal> Go for Shopping</Func_Goal>
           < Alternative Func_Plan> Find Shopping Malls</ Alternative Func_Plan>
           <Context> Location: Paris_Hotel; </Context>
           <Quality_Attribute> Good Places, Convenience</Quality_Attribute>
           <Preference> ¬Taxi & ¬Metro </Preference>
           <Stimulus>Search Map & Search Transport</Stimulus>
           <Date> Current</Date>
           <Time> Evening</Time>
           <Response> Map: Google Map; Paris Taxi Service; List: Malls</Response
           <Resource>Mobile, Contact List</Resource>
     </RequestDescription>
</Request>
```

Figure 4.3: User Specified RRA.



```
<?xml version="1.0" encoding="UTF-8"?>
  <Request xmlns="http://www.example.org/Request">
     <RequestDescription>
           <Source>System </Source>
           <Func_Goal> Find Accommodation</Func_Goal>
           < Alternative Func_Plan> Find Hotel</ Alternative Func_Plan>
           <Context> Location: Paris_Airport; </Context>
           <Quality_Attribute> Cost Effective</Quality_Attribute>
           <Preference> ¬Far from Airport & Cost<=70Euro a night </Preference>
           <Stimulus>Search Map & Search Transport & Book Hotel </Stimulus>
           <Date> Current</Date>
           <Time> Evening</Time>
           <Response> Map: Google Map; Paris Metro; List: Hotels </Response>
           <Resource>Mobile, Credit Card </Resource>
     </RequestDescription>
</Request>
```

Figure 4.4: System Generated RRA.

to choose in its existing P', it looks for another possible solution using existing repository of specification (i.e. P) to select another P', which might satisfies users' goals. It generates an RRA with a goal Find Accommodation *along with its operational plan* Find Hotel *and operationalize it with the candidate services (e.g. Booking.com) taking into account users' preferences ($\neg$ Far from Airport, Cost$\Leftarrow$70 Euros) with a quality constraint (e.g. Cost effective) and context (Paris Airport) and resources (e.g. credit card) information. Since, user opts to book a hotel and this RRA is updated in the initial specification P'.*

Example 2 *Subsequently, taking into account users' context (location) information, it proposes list of options e.g. places to eat, visit museums or watch movie along with alternative ways to commute based on users' preferences (i.e. Cost or Convenience). User rejecting these options, propose to* Go for Shopping *and input* Find Shopping Malls. *System, upon acquiring this new requirements, tries to operationalize it searching and providing the user with a list of Malls along with possible routes to reach them using Map service (e.g. Google Maps). It also propose transport service taking users' preferences i.e. Bus $\succ$ Taxi $\succ$ Metro, along with quality constraints i.e. Good Places, as shown in Fig. 4.3.*

Companion SBA includes a *User Agent*; *Control Agents* i.e. monitor, evaluator, adaptation agent; *Reasoner Agent*; *Service Monitor Agent* and *Ser-*

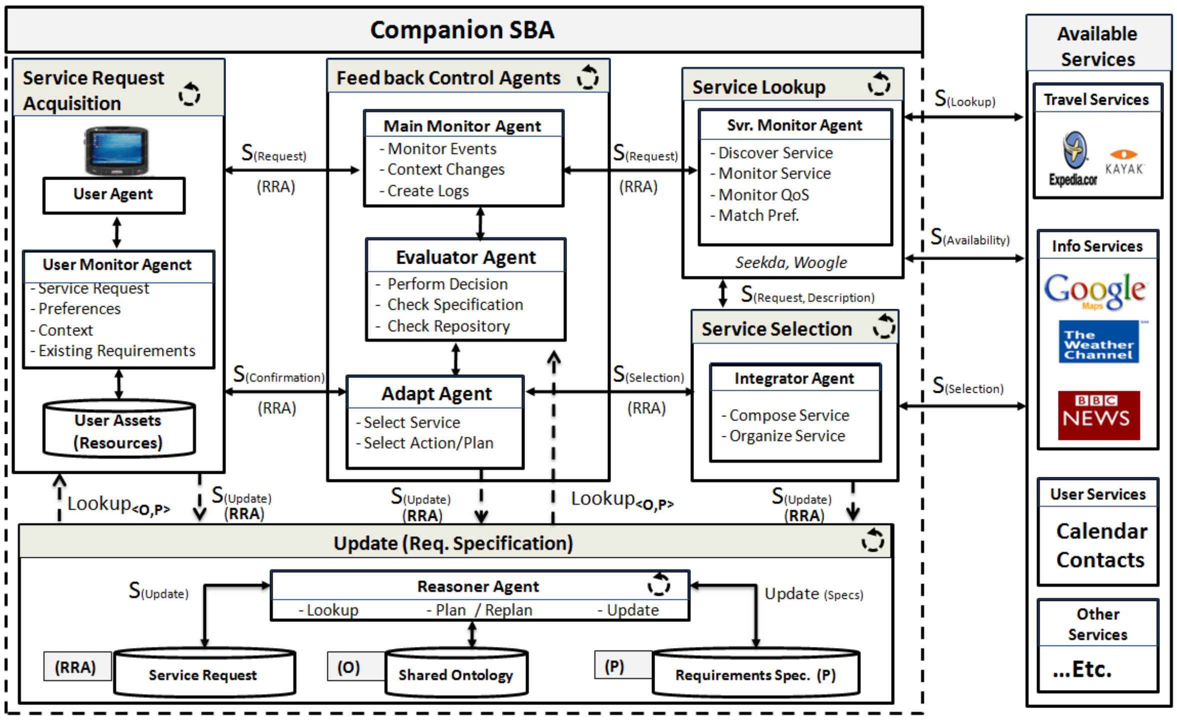


Figure 4.5: Conceptual Architecture of CARE

vice Integrator Agent that helps performing the CARE run-time RE activities i.e. *Service Request Acquisition*, *Service Lookup*, *Service Selection* and *Update Specification*. A set of available services (e.g. Maps, Calender, Travel, Weather and News Services) are used to accommodate end-users' needs and preferences. Service can be added, removed or modified with new ones.

Each activity is represented by a square box with a loop symbol meaning that each activity is performed continuously. Agents supporting these activities are represented with square boxes inside relevant activity. Solid arrow head lines represents coordination among activities along with input/output operations specified over them. Dotted single arrow head lines represent Lookup and update operations over repositories as shown in Fig. 4.5. The underlying implementation tier is service-oriented, rest we are agnostic about how the underlying services are implemented.

4.4.1 Service Request Acquisition

is performed by two agents: 1) *User Agent*, whose role is to provide an interface to its user enabling her to express service requests ($S_{request}$) as RRA. RRA can be either expressed by the user or Companion itself. This agent also helps the user to manage her travel itineraries, set preferences, manage personal contacts and to add services by searching them e.g. Weather, News etc.; 2) *User Monitor Agent* is focused on detecting changes in the existing requirements (i.e. goals, preferences), user context and change or new service requests.

This activity is mainly responsible for eliciting the new requirements or refining existing requirements in the form of service requests (i.e. RRA). The existing specification P' in the specification repository is compared with the given request (i.e. RRA) by performing $Lookup_{<O,P>}$ operation, where O represents shared ontology and P represents specification repository. This operation compares the RRA (XML fields) with the existing specification P' (represented in XMI format) by pattern matching using regular expressions and identifies a corresponding action from the ontology to populate the RRA.

Companion SBA upon monitoring the contextual changes (e.g. changing in user profile, location) in the environment (e.g. events) may identify a violation of a goal or an unexpected performance of a service leading to generate a *service request* (RRA) on its own and opt to take corrective actions or proposes actions to the end-user in case of error interpreting her input. Note, that each operation has consequences e.g. looking up a new service or reconfiguring the existing ones.

Subsequently, RRA is passed as ($S_{request}$) (RRA) to the Feedback control agents for identifying suitable operationalization by looking for alternative or new services that can satisfy the users' intentions. Later, an update to the service request repository tagged with (RRA) is made using operation i.e. $S_{Update<RRA>}$. At this stage RRA is updated in the repository with the identified actions and specified new/changed requirements.

Example: *The main goal of the user is* Travel For Business. *We refer to the Example 2, where user provide input as* Go for Shopping *and* Find Shopping Malls, *Companion SBA operationalize it by searching for shopping malls by monitoring users' context (i.e. Paris Airport) evaluating the change in itinerary (i.e. Flight Delay in Paris) and providing a list of malls to visit asking user for preferences to commute (i.e. Bus is preferred over taxi and metro), by being aware of her softgoal* Minimize Cost. *These options were not perceived during designtime and are treated as new requirements (additional goals/tasks), which are operationalized by selecting available services. Working in conjunction with the end-user, Companion SBA search for optimal solutions (e.g. Maps Service: Google Maps, Transport Service: Paris Bus) and simultaneously RRA is generated acquiring users' input and in parallel system fill it based on monitored information.*

4.4.2 Service Lookup

is performed once the RRA is created. Feed back Control Agents - Main Monitor Agent collects input acquired from User Agent, feeds to Service Monitor Agent to perform this activity. RRA is filled with users' input and monitored information e.g. context (location) and events - flight delay. Service Monitor Agent performing this activity, discover services by looking up using e.g. Seekda or Woogle web service search engines. In case of requirements refinement, competing services are replaced with related ones, where as in case of requirements addition new services are searched by performing S_{Lookup} operation using search engines. Correspondingly, operation $S_{Availability}$ is returned with list of services along with service signatures e.g. Map Service - Google Maps.

Current service description and discovery standards i.e. WSDL and UDDI rely on structured format (e.g. XML) to describe services (i.e. service signatures). The search is performed based on the keywords in RRA. Service Monitoring Agent, matches keywords in RRA with the Service descriptions to help discovering the services, collecting information about existing service QoS by matching given users' preferences.

Example: *Arriving at Paris airport, the Companion SBA monitors a delay in flights using Weather Service, and Lufthansa Itinerary Update Service. It then proposes the user to book a hotel using Booking.com by asking her to specify preferences i.e 3 Star, ¬ Far from Airport, Cost⇐70 Euros) with a quality constraint (e.g. Cost effective), see Example 1*

4.4.3 Service Selection

is performed by Integrator Agent based on an input received from Service Lookup activity as $S_{Request,Description}$ and from Adapt Agent as $S_{Selection}$. Former holds the candidate services along with their service signatures (e.g. WSDL) and latter provides the criteria for selection of service. This activity relies on *Integrator Agent*, which helps in orchestrating or composing the new services. It selects the candidate service that best operationalize the users' service request (RRA). Correspondingly, the Adapt Agent seeks confirmation from the user as $S_{Confirmation}$. Once the confirmation is given, the RRA is filled with the selected service, which completes it. Later RRA is updated by performing $S_{Update<RRA>}$ operation.

Example: *The system attempts to update the itinerary to accommodate flight cancellations. It proposes the user to stay at Hotel, providing a list of Hotels through Booking.com, providing routes to hotels using Google Maps and propose commute with Bus or Taxi by using Airport Shuttle Service or Airport Limo-Taxi Service to Hotel. User confirm by selecting the proposed hotel as per her preference i.e. 3 Star hotel, ¬ Far from Airport, Cost⇐70 Euros, and selects the Airport Shuttle as per her cost preference*

4.4.4 Update Specification

activity ensures an update to the repositories as shown in Fig.4.5. A Reasoner Agent performs update operations e.g. lookup, plan, re-plan and update to existing repositories e.g. Requirements specification P, shared ontology O and service request repository RRA. The reasoning rules and details about refinement operations we have identified e.g. Add_{Goal}, Add_{Task}, $Substitute_{Goal}$, $Substitute_{Task}$, $Relegate_{Goal}$, $Relegate_{Task}$, $Delete_{Goal}$, $Delete_{Task}$; provides support for continuous management (refinement) of requirements at run-time.

Example *The new alternative goal is added* Go for Shopping *along with its operational plan* Find Shopping Malls*, which is operationalized by looking up services e.g. Map, and Transportation service, based on user's given preferences e.g., Bus is convenient than metro and cheaper than taxi. In this case, an alternative to the goal Travel for Business is added, new set of preference e.g. Bus $\succ$ Metro $\succ$ Taxi is accommodated considering Cost $\succ$ Convenience and a quality constraint i.e. Good Places. Correspondingly, set of services operationalizing this goal are also updated.*

4.4.5 Feedback Control Agents

are responsible for being aware of end-user's goals and preferences and new service requests $S_{Request}$(RRA) through monitoring, evaluating such changes and adapting to a suitable alternative that can satisfy them.

- **Main Monitor Agent** is responsible for getting feed back from *User Monitor Agent* and *Service Monitor Agent* as $S_{Request}$(RRA). It collects information about the external events, changes in end-user's context, goals and preferences and provides a timely information to the Evaluator Agent by logging the changes.
- **Evaluator Agent** evaluates the events, perform $Lookup_{<O,P>}$ on the

repositories if needed and decide about the appropriate behavior (i.e. changing the service based on improved QoS criteria), thus letting the Adapt Agent to select and enforce the candidate plan to invoke the service that operationalizes the RRA. The candidate refinement operations to refine the specification artifact e.g. Add_{Goal}, Add_{Task}, $Substitute_{Goal}$, $Substitute_{Task}$, $Relegate_{Goal}$, $Relegate_{Task}$, $Delete_{Goal}$, $Delete_{Task}$; along with service selection criteria are decided by this agent.

- **Adapt Agent** takes input as the decision criteria for the service selection. It shares the criteria with the Integrator Agent as $S_{Selection}$(RRA) and subsequently seek confirmation from the user for candidate service as $S_{Confirmation}$(RRA). Later, this agent also performs an update $S_{Update<RRA>}$. Moreover, the candidate operation to update the specification is also selected based on given input from Evaluator Agent and subsequently added in the (RRA).

4.5 Related Work

In this chapter we presented a novel CARE framework for continuous refinement of requirements at run-time by the SAS itself involving the end-user. A conceptual architecture for SAS is described that realizes CARE's RE activities at run-time. A classification of adaptation types is presented that is prescribed by the CARE framework to reason for adaptation at run-time.

Various approaches has recently presented frameworks for realizing self-adaptive systems. For instance, Morandini et al. [MPP08] presented a goal-oriented framework for developing SAS. In this framework Tropos methodology has been extended to incorporate different extended models for goal type, environment and failure. Concepts associated with these models are also proposed such as goal types, conditions and failure types. The approach followed is strictly intertwined with the agent-oriented BDI framework. A tool has been implemented to derive the code definitions. In this case, designer has to be involved through the process and agent's behavior is explicitly coded at design-time. The positive point in this approach is preservation of goals, but on the other hand new and changed requirements cannot be accommodated at run-time, thus (Type 1, 4 adaptation supported).

Similarly, a conceptual architecture for reconfiguration is proposed by Daliaz et al. in[DGM09]. This architecture guides provides necessary component details while designing SAS. Key principles are defined for this architecture and a variant of Tropos goal models is proposed which represents contextual variabilities. The proposed architecture is targeted for a multi-agent environment, e.g. socio-technical system. Contextual goal models and explicit social variability is considered along with policies to guide reconfiguration of SAS. The limiting feature is that variants of possible adaptations are already thought at design-time. Social interactions among the interacting agents are formalized using commitments and time. In fact, this form is more restrictive, as SAS requires flexibility in their design. However, in case of socio-technical system its a valid choice adopted in this work. Overall, this approach is more focused on designing SAS in context of socio-technical systems, where domain knowledge is required to prior to designing the system. Likewise, this work also limited as accommodating new or changed requirements by involving end-users are not considered, thus (Type 1, 2, 4 adaptation supported).

Baresi et. al. [BP10] has proposed the concept of "Adaptive goals" as an abstraction to represent adaptation strategies, and countermeasures to address goal violations and conflicts by extending KAOS [DvLF93b] method, which are then operationalized as to compose services operationalized as an executable BPEL. Adaptation goals are responsible for the for the evolution/adaptation of the goal model. Adaptation goals represent strategies operationalized as a supervision model. They are responsible for substituting the weaker notion of goal when a violation is detected. Conditions to identify obstacles are transform into the native language of a monitoring systems Dynamo which is used to evaluates formal properties of a BPEL process. This work has been extended in[BPS10], where they represent a framework FLAGS (Fuzzy Live Adaptive Goals for Self-adaptive systems) in which KAOS goals are further categorized as crisp and fuzzy goals. Crisp goals determines the functionality of the system (i.e. representing a functional model). The satisfaction scale of these goals is boolean. Satisfaction of fuzzy goals is represented using a fuzzy scale, formalized using fuzzy logic. In this work, the designer needs to express all the required constraints and identify constraints when a particular may be violated to trigger adaptations, thus (Type 1, 2, 4 adaptation supported).

Altogether, CARE provides a more flexible approach for engineering SAS. It bridges the gap between design-time and run-time RE activities be clearly identifying the role of human users and of SAS. Moreover, CARE support adaptation types, which provides a clear distinction with respect to the state of the art approaches.

4.6 Final Remarks

In this chapter, we presented our novel CARE framework, which bridges the gap between RE activities performed at design-time and at run-time. It identifies the role of human user and specifically the role of SAS at run-time. End-user involvement is critical to the success of SAS performing adaptation at run-time, thus enabling it to support a seamless evolution of its specification. This facet distinguish CARE from the state of the art approaches. Moreover, CARE support the dynamic RE problem at run-time. In line with the recent vision of the RE community for self-adaptive systems, CARE provides run-time process and requirements that can be kept alive to support RE at run-time.

Chapter 5

Engineering Adaptive Requirements

5.1 Overview

In this chapter, we focus on the RE at design-time for SAS and propose a systematic approach to engineer requirements. We introduce a new form of requirements for SAS called "Adaptive Requirements"[1], which is not a primitive concepts but it aggregate requirements. We investigate the need of these that are used to specify flexible requirements of SAS. We exploit the core ontology of RE for SAS defined in Chapter 3 to elaborate the concept of adaptive requirements. We introduce a modeling language, called Adaptive RML, to systematically model adaptive requirements to represents requirements for SAS. The language has graphical primitives in line with classical goal modeling languages and is formalized via a mapping to Techne. Such models enable the analyst to formulate and analyze the requirements problem during early RE, before moving to detailed specification. Early validation is performed by modeling the same case study in an established goal mod-

[1]Note that this term has been introduced in [HED93] as: "Adaptive requirements are requirements for change to be an inherent and on-going capability of the delivered system". We extend this view in case of SAS that adaptive requirements does not only express functional or non-functional requirements but also encompass requirements for feedback functionalities i.e. monitoring specification, evaluation criteria and adaptation actions, thus enabling the change to be managed by SAS using these adaptive requirement that are aggregate set of requirements.

eling language and in Adaptive RML. The results suggest that context and resource concepts, as well as relegation and influence relations should be part of graphical modeling languages used to make early requirements models for SAS and to perform analysis over them. Later we introduce a convenient way to operationalize adaptive requirements using Event-Condition-Action pattern and to derive monitoring specification, evaluation criteria and action specification.

The rest of the chapter is organized as follows. In Section 5.2, we present key concept of adaptive requirements using examples from the Travel companion case study. In Section 5.3, we introduce the visual model language to systematically model adaptive requirements. In Section 5.4, We present the operationalization of adaptive requirements and elaborate on the proof of concept tool to support the specification of adaptive requirements and process to derive monitoring specification, evaluation criteria and action specification. In Section 5.5 we compare our proposed approach with existing works. Finally, in Section 5.6 we summarize the key contributions of this chapter.

5.2 Adaptive Requirements

Requirements for self-adaptive systems must reflect uncertainty about the run-time environment [WSB$^+$10]. It is due to the changes motivated by the variability in the operational context, availability of the resources and in end-user's needs. Traditionally, software requirements are usually characterized along the functional and non-functional classification [NE00]. While eliciting and specifying them, the analyst first attempts to classify the stakeholders' needs that may be elicited through interviews or domain documents, both using natural language, to characterize these requirements. For instance, *I need a user friendly confirmation message after I book the flight*, can be an example of need expressed by the end-user for a travel booking service.

Here, we can classify the requirement according to the functional and non-functional perspective i.e. answering questions like, ***What the system should do?*** and possibly ***How (well) should it do?***. Two functional requirements can be identified from the above requirement e.g. Booking Flight, and Sending Confirmation Message, and one non-functional requirement e.g. Send a User Friendly Message.

Adopting a goal-oriented requirements engineering [vL01a] approach allows us to answer also ***Why the user wants this?; Why in this way?***, questions that result in providing a rationale (i.e. stakeholders *goals*) behind the functional requirements e.g. confirmation to be notified. While analyzing the non-functional, we can see that requirements that does not mandate a precise criteria for their satisfaction are *softgoals* e.g. user friendliness. *Quality constraints* on the other hand are requirements with a precise criteria e.g. confirmation must be sent in less than 5 minutes. Another facet that has been recently identified in the core ontology for RE[JMF08], is *preferences*. They express the attitude of the stakeholder towards a particular requirement e.g. user friendliness can mean short message, differently for other stakeholder it can be format of the message such as html or text. Further analysis of software requirements may lead to more interesting questions i.e. ***Where*** and ***When*** the requirement can be achieved or satisfied, to answer. Thus, leading to a complete specification.

In context of self-adaptive systems, we need such kind of analysis that can provide basis to engineer SAS as a requirements-aware system [SBW+10]. Although, the variability provides alternative ways for the SAS to exploit at run-time (i.e. switching between alternative solutions, Type 1 adaptation), but the rationale to guide its behavior comes from the requirements. Another key requirement for engineering SAS is to equip them with a feedback control loop [BSG+09] mechanism, which enables them to monitor changes in the environment, evaluate theses changes and finally adapt it behavior accordingly (i.e. re-configuration, or adopt an alternative new behavior). Therefore, at requirements level, we need to make explicit the alternatives in goal achievement, i.e. variability in ***What*** and ***How***, which may be further enhanced by the variability in ***Where*** and ***When*** due to the openness of the operational environment. Addressing these question, we also need to make explicit the feedback loop functionalities by answering to the questions i.e. ***what to monitor***, ***what to evaluate*** and ***how to adapt***.

To this aim, we explore and define requirements that specify properties of an self-adaptive systems, that are not only *functional* or *non-functional* but also includes requirements for *monitoring* that is to observe the variability in the operational context, *evaluation criteria* that is to check where system is not respecting the expected values and *alternative* solutions that are *behaviors* to be adopted at run-time by the SAS to ensure the achievement of its intended end-users' goals. We define such requirements as ***adaptive requirements***, defined in short as:

Requirements that encompass the notion of variability associated to either a functionality or a system quality constraint along with monitoring specification, evaluation criteria and alternative solutions.

Requirements specification of SAS must include the *notion of adaptivity* as proposed by Berry et al. [BCZ05]. To support this vision, we propose "adaptive requirements".

Illustrative Example

Lets consider the following example requirement of an end-user:

For instance, *A user friendly confirmation message, after booking is processed, must be communicated to the user on his current device with proper representation.*

The analyst may describe it with the following statement and identify from this the requirements for Travel Companion Application. For instance, by understanding the domain information, an analyst may identify four functional requirements namely *book a flight ticket*, *send confirmation message*, *message communication to an available device* and *format representation of message* by answering the questions i.e ***Why and What*** is required. Further with this analysis, one non-functional requirement is identified i.e. *user-friendly message* by addressing ***How*** well it should send.

The analyst can now make explicit the variability in user's devices (implicit in the above requirement statement) and re-phrase it as follows:

A user friendly confirmation message, after booking is processed, shall be communicated to the user on her current available (device) e.g. smartphone, Laptop by seamlessly observing (monitor) the (events) e.g. flight status is confirmed, or delayed, (quality constraints) e.g. inform in less than 10 minutes and her (context) e.g. profile, location, device, taking into account her preferences (pref) e.g. Send SMS when outdoor, in order to deliver required personalized contents to her current device.

The availability of end-user's device (e.g. smartphone, Laptop) ad variation in her context (e.g. location, profile settings) demands monitoring at run-time. To make SAS aware of the current operational events (e.g. flight status not confirmed), it needs to evaluate conditions – quality constraints (if any) and exploit at best its available alternatives such as send the confirmation message in a proper (formatted) way to the end-user's available device, so to meet her original goals.

Certainly, unanticipated events – may occur with significant probability, should also be considered. For instance, message is not delivered correctly or the Internet connection is lost. As a consequence, a new requirement must be added i.e. SAS must degrade gracefully or involve the end-user to acquire more knowledge to take corrective actions. This refers to the adaptation types described in Chapter 4.

In addition, to cope with such level of uncertainty (what might happen or when it happen), the analyst need to ensure at design-time that while addressing previous questions the requirements shall also try to address ***When*** and ***Where*** uncertainty. Returning to the analysis, the above example can be re-stated as follows:

A confirmation message for booking is generated as soon the booking is processed, and required to possibly communicate the message to the end-user on her current available (device) e.g. smartphone, by seamlessly observing (monitor) the (events) e.g. flight status is confirmed, or delayed, (quality constraints) e.g. inform in less than 10 minutes and her (context) e.g. profile, location, device until the message is delivered in a correct format e.g. scaling it, size, and with personalized representation (e.g. html, text) and as per her preferences i.e. (pref) e.g. Send SMS when outdoor, to her current available device i.e. smartphone, PDA or a different way to notification is applied.

Note that we have added *temporal operators* to relax the above example. This is one of the way that has been recently proposed in Relax Language [WSB$^+$10]. This is a particular way to relax requirements. In Chapter 3, we introduce a more general relation i.e. relegation, which can be considered at this stage to analyze which requirements can be relegated with respect to others. Although, there might be situations in which there is no possibility to find a candidate solution at run-time. For example, the message could not be confirmed as delivered either due to wrong message delivery or connection lost. Then, using for instance end-users' contact info that SAS, is aware of, can either send an email to her secretary or notify her stared contacts e.g. a friend.

The key essence of this analysis is to show that in context of SAS, classical RE departs from its traditional ways of analyzing requirements. Further information is needed to make the implicit knowledge through explicit analysis. Moreover, in stating requirements for SAS, flexibility has to be considered e.g. requirements can be optional i.e. "nice to have" [LMM10a]. In this sense, precise requirements may need to be relaxed depending upon the changes. For this we state the requirements problem as a dynamic RE problem, when SAS has to move from one requirements problem to another by changing its candidate solutions with respect to the context conditions at run-time by involving the end-user. For example, in the above example, it is not made precise when the message will be delivered or it must be delivered, but it has to be delivered.

In this case, flexibility is in ***When*** aspects where requirements can be relegated by other requirements, though they may not be most preferred by providing a way to satisfy the high level goals. With this view, availability of end-users' resources (e.g. smartphone, contacts) addresses (***Where***) questions. Also, end-users' preferences over goals – ***What, Why*** helps in finding the candidate solution – tasks and domain assumptions in a context gives answers to ***How*** and uncertainty in the environment can be reduced by monitoring and evaluating the system and end-user context must be considered (see Chapter 3 for details).

To this end, we have discussed the need to identifying adaptive requirements, which makes explicit the feedback functionalities such as monitoring, evaluation criteria and candidate solutions, while analyzing the end-user needs. Below we define a language using the necessary elements as introduced in our core ontology of RE for SAS – defined in Chapter 3, to support this analysis. We thereby capture and identify "*adaptive requirements*" at the early stages of RE process.

5.3 Definition of Adaptive RML

In RE, early requirements phase results in the information gathered from the stakeholders. An RML supports this activity enabling the analyst to structure and relate this information meaningfully by modeling it therefore formulating a requirements problem and later conduct analysis to resolve the requirements problem by finding candidate solutions. This results into an adaptive requirements specification.

We start by defining an RML, we call it Adaptive RML, along the concepts and relations defined in Chapter 3. A visual syntax, modeling guidelines, refinement patterns and analysis that can be performed over models build using Adaptive RML is presented. In particular to model adaptive requirements and provide a systematic modeling approach using Adaptive RML.

5.3.1 Concepts & Relations

We now present the necessary concepts and relationships that are required to represent adaptive requirements in order to formulate the requirements problem for SAS. Addition of these concepts and relations leads us to an ontology of requirements for SAS and the formulation of the *run-time requirements adaptation problem* as a dynamic problem of changing (e.g. switching, re-configuring, optimizing) – SAS move from one requirements problem to another requirements problem, with respect to the changes in requirements, context conditions, and/or resource availability. We revisit the two new concepts, Context and Resource as well as relations Relegation and Influence that are added to enhance the tool set for the proposed Adaptive RML to model and analyze *adaptive requirements* and requirements for SAS in general.

Context: This concepts allows modeling information that the stakeholders assume to hold when they communicate particular requirements. We say that every requirement depends on one or more contexts to express the fact that the requirement would not be retracted by the stakeholders in every one of these contexts. This information needs to be made explicit in the early stages of requirements modeling for SAS. For instance, in our example we modeled "context" as information about location (e.g. Office or Market), which are defined as concepts in a specific domain ontology (e.g. travel), and we link them to tasks via an inference relation. In Fig.5.3, context is shown as e.g. "C1 [Cx: @Market]" where "@Market" is an instance of a concept term (i.e. Location) defined in a domain ontology. Combining requirements and context reveals interesting cases, where we can see requirements may result in conflict or an becomes inconsistent. In this case, *preferences* play a critical role to connect this information, where an end-user can specify in which context, they require a particular task to be executed e.g. notify about message through SMS while she is in an outdoor location.

Resource: The concept of resource has been well supported in RE methods such as in goal-oriented approaches [DvLF93a, Yu97, PPSM07b]. In our case, we define it as an entity that is referred to by the requirements, e.g., physical/tangible entities such as mobile phone, ticket itinerary; e.g., intangible entities, such as user assets (social relations or contacts). In order to introduce resources in the definition of the requirements problem for SAS, we need to elicit a resource availability function that tells us which resources are available and used in some way, in order to ensure that the relevant domain assumptions and context propositions hold, and that the tasks can be well executed. Here again we may exploit ontology definitions of user-assets and asset modifiers that represents tasks effects on their resources, as proposed in [MNF$^+$10]. In the modeled example shown in Fig.5.3, we introduced "Mobile Phone" and "Laptop" as resources available in different contexts.

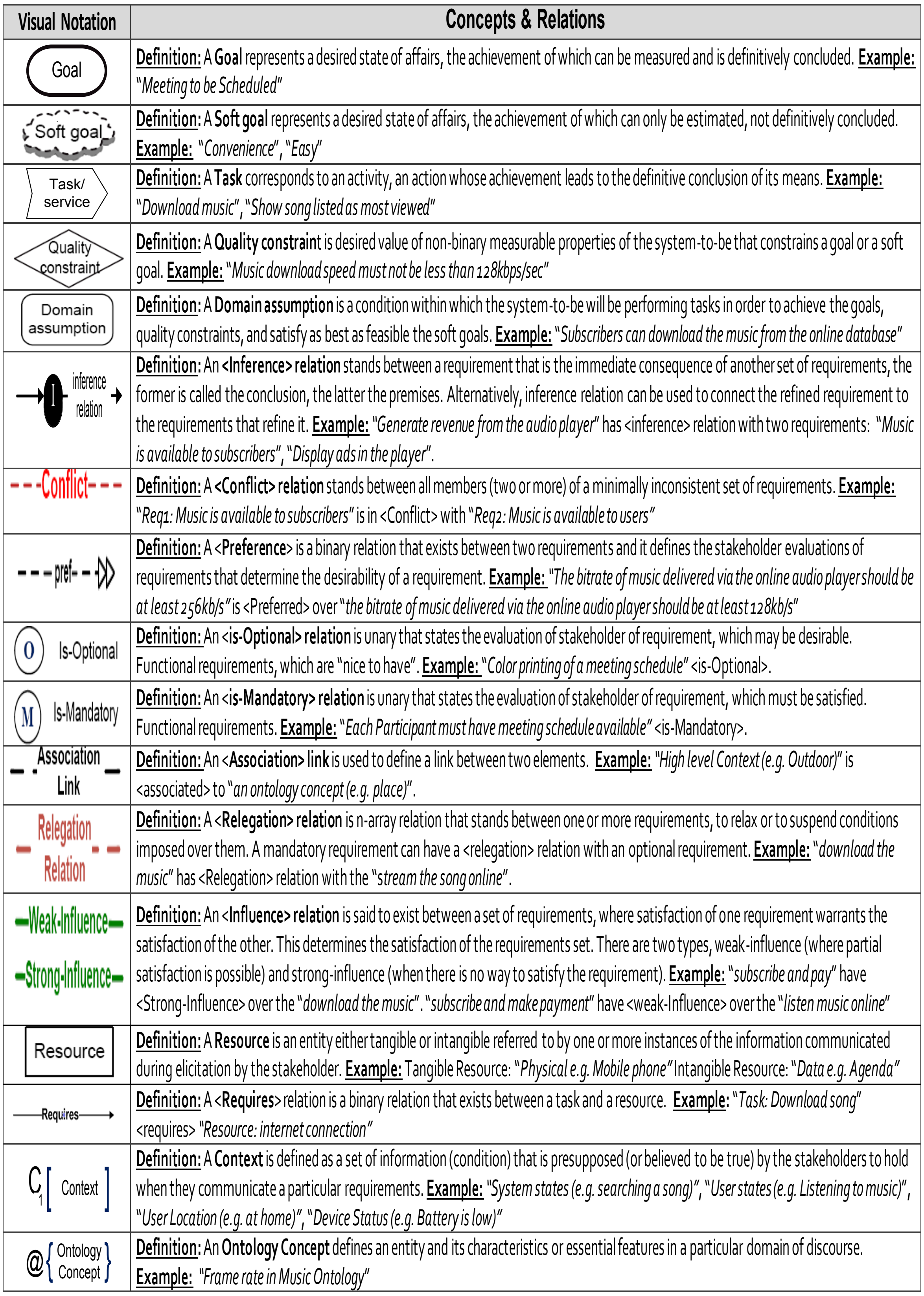
Relegation relation: The purpose of the Relegation relation (`Rel` for short) is twofold. First, it facilitates engineer at design-time to analyze requirements (including goals, quality constraints, preferences) and relegate their associated conditions (e.g. pre/post, achievement, trigger conditions) by anticipating run-time change scenarios. Secondly, it enables SAS at run-time to analyze requirements problem in case of changes that can occur dynamically e.g. change in user's context, violation of domain assumption, resource usage or change in user's need or preference, either through sensing the operational environment or explicitly given by the end-user.

A `Rel` is applied to manage unanticipated events, by flexibly relegating some of the requirements, with the aim to avoid failure in achieving the critical ones. In this case by applying `Rel`, either the solution that operationalizes a goal needs to be replaced, or an instance of the same goal with revised conditions is linked using `Rel` with the original goal e.g. in Fig.5.3, candidate solution “Send via SMS” is relegated by “Place Call”, when context conditions changes. In this example, the instance of the original goal is not compromised rather relegation is considered by replacing the preferred solution with an optional one.

Influence relation: An influence relation (`Inf`) is introduced to analyze the impact of changes in model elements that define different, mutual dependent requirements. This means, if change in the operational environment or in end-user requirements happens at run-time it might cause a change in another requirement. This chain of dependency needs to be identified, since along them we may identify changes consequences such as violation of a goal or a invalid solutions. For example, in Fig.5.3, if no candidate solution is possible to achieve the goal “Message Transfer Method Selected” due to invalid context and domain conditions, then this goal will fail, which causes a violation in satisfying the corresponding goal i.e. “Message Composed”. Similar dependencies can be collected and subsequent consequences are determined by analyzing the impact of changed solution.

5.3.2 Adaptive RML Visual Notations

The Adaptive RML language provides a graphical notation, which is in line with classical goal modeling languages and is formalized via a mapping to Techne. A detailed guide on visual elements is presented in the Table shown in Fig. 5.1: each row contains a graphical symbol and a short description of it’s intended meaning. For the elements that map the Techne core ontology, the



Visual Notation	Concepts & Relations
Goal	Definition: A **Goal** represents a desired state of affairs, the achievement of which can be measured and is definitively concluded. Example: "*Meeting to be Scheduled*"
Soft goal	Definition: A **Soft goal** represents a desired state of affairs, the achievement of which can only be estimated, not definitively concluded. Example: "*Convenience*", "*Easy*"
Task/ service	Definition: A **Task** corresponds to an activity, an action whose achievement leads to the definitive conclusion of its means. Example: "*Download music*", "*Show song listed as most viewed*"
Quality constraint	Definition: A **Quality constraint** is desired value of non-binary measurable properties of the system-to-be that constrains a goal or a soft goal. Example: "*Music download speed must not be less than 128kbps/sec*"
Domain assumption	Definition: A **Domain assumption** is a condition within which the system-to-be will be performing tasks in order to achieve the goals, quality constraints, and satisfy as best as feasible the soft goals. Example: "*Subscribers can download the music from the online database*"
I inference relation	Definition: An **<Inference> relation** stands between a requirement that is the immediate consequence of another set of requirements, the former is called the conclusion, the latter the premises. Alternatively, inference relation can be used to connect the refined requirement to the requirements that refine it. Example: "*Generate revenue from the audio player*" has <inference> relation with two requirements: "*Music is available to subscribers*", "*Display ads in the player*".
Conflict	Definition: A **<Conflict> relation** stands between all members (two or more) of a minimally inconsistent set of requirements. Example: "*Req1: Music is available to subscribers*" is in <Conflict> with "*Req2: Music is available to users*"
pref-	Definition: A **<Preference>** is a binary relation that exists between two requirements and it defines the stakeholder evaluations of requirements that determine the desirability of a requirement. Example: "*The bitrate of music delivered via the online audio player should be at least 256kb/s*" is <Preferred> over "*the bitrate of music delivered via the online audio player should be at least 128kb/s*"
O Is-Optional	Definition: An <**is-Optional> relation** is unary that states the evaluation of stakeholder of requirement, which may be desirable. Functional requirements, which are "nice to have". Example: "*Color printing of a meeting schedule*" <is-Optional>.
M Is-Mandatory	Definition: An <**is-Mandatory> relation** is unary that states the evaluation of stakeholder of requirement, which must be satisfied. Functional requirements. Example: "*Each Participant must have meeting schedule available*" <is-Mandatory>.
Association Link	Definition: An **<Association> link** is used to define a link between two elements. Example: "*High level Context (e.g. Outdoor)*" is <associated> to "*an ontology concept (e.g. place)*".
Relegation Relation	Definition: A **<Relegation> relation** is n-array relation that stands between one or more requirements, to relax or to suspend conditions imposed over them. A mandatory requirement can have a <relegation> relation with an optional requirement. Example: "*download the music*" has <Relegation> relation with the "*stream the song online*".
Weak-Influence Strong-Influence	Definition: An **<Influence> relation** is said to exist between a set of requirements, where satisfaction of one requirement warrants the satisfaction of the other. This determines the satisfaction of the requirements set. There are two types, weak-influence (where partial satisfaction is possible) and strong-influence (when there is no way to satisfy the requirement). Example: "*subscribe and pay*" have <Strong-Influence> over the "*download the music*". "*subscribe and make payment*" have <weak-Influence> over the "*listen music online*"
Resource	Definition: A **Resource** is an entity either tangible or intangible referred to by one or more instances of the information communicated during elicitation by the stakeholder. Example: Tangible Resource: "*Physical e.g. Mobile phone*" Intangible Resource: "*Data e.g. Agenda*"
Requires	Definition: A **<Requires>** relation is a binary relation that exists between a task and a resource. Example: "*Task: Download song*" <requires> "*Resource: internet connection*"
C_1 [Context]	Definition: A **Context** is defined as a set of information (condition) that is presupposed (or believed to be true) by the stakeholders to hold when they communicate a particular requirements. Example: "*System states (e.g. searching a song)*", "*User states (e.g. Listening to music)*", "*User Location (e.g. at home)*", "*Device Status (e.g. Battery is low)*"
@{ Ontology Concept }	Definition: An **Ontology Concept** defines an entity and its characteristics or essential features in a particular domain of discourse. Example: "*Frame rate in Music Ontology*"

Figure 5.1: Visual guide for concepts and relations in Adaptive RML.

corresponding semantics is given in [JBEM10b], while the formal semantic of the additional concepts is defined in Chapter 3.

Worth to be mentioned is that recent research evaluated weaknesses of widely used goal-oriented modeling notations with respect to principles for cognitively effective visual notations [MHM09]. The proposed visual notation consider two among the principles discussed in [MHM09]. The first is visual expressiveness: notation must comprise of color, shape and brightness instead of shape only. Second is Semiotic clarity, which postulates that each graphical symbol must have a 1:1 correspondence with its semantic definition i.e. the concept which they refer to. Our proposed notation takes as much as possible these principles into account.

5.3.3 Systematic Modeling in Adaptive RML

Adaptive RML makes no assumptions and imposes no constraints on how the information is acquired while modeling the requirements for SAS. At modeling time a requirements model in Adaptive RML is constructed by recording and structuring relevant information obtained through elicitation. As a result, the run-time requirements adaptation problem is formulated for the SAS-to-be. New pieces of information are gathered during modeling time to refine the problem iteratively. At analysis time, all candidate solutions to that problem are sought along with their differences to each other and are compared with respect to varying context situations and resource availability.

The modeling process develops by performing iterations of the following activities.

1- Modeling Mandatory and Optional Goals:

We start modeling goals, optative statements that defines the desired properties of the SAS-to-be, via inference relation (i.e. symbol (I)). We use (I) node to depict refinements (e.g. AND/OR decomposition, or means-end relation). Each (I) node connects the model element to be refined to simpler or more concrete elements that refine it. In this way it is concluded that if the requirement is defined by the concrete elements then they can satisfy the more abstract one.

Further, we add softgoals that depicts vague properties of SAS-to-be, which are approximated in terms of quality constraints determining the criteria to measure them.

2- Modeling Domain Assumptions:

While modeling goals we discover domain assumptions that are statements in the domain which are assumed to be always true. We add them via (I) node and add (if any) to each goals or tasks. Subsequently, during refinement, quality constraints can be inferred. We add criteria to measure the goal satisfaction via (I) node. During this, new pieces of information are discovered such as conflicts and preferences among the goals and tasks.

3- Modeling Conflicts and Preference Relations:

Conflicts and preferences are identified during refinement. We discover conflicts between inconsistent/contradictory requirements or tasks node between conflicting set of requirements/tasks. Further, we identify preferences taking into account stakeholder's evaluations about different requirements. We add preference relation between requirements where satisfying one is strictly more desirable than satisfying the other.

4- Modeling Mandatory or Optional Tasks:

Likewise, we model tasks as further refinement of goals. Task modeling can be seen as an analysis activity, where we add tasks via (I) node to operationalize goal. This means, if the tasks will be successfully completed, the goal will be achieved. Goals can be either mandatory or optional (i.e. (M)) or (O) respectively), we model this by adding these unary relation over goals.

5- Modeling Context and Resources:

Once the requirements model is constructed, we further anticipate the various situations in which requirements or tasks can be either achieved or not. We add *context* node to each requirement/task. Context refers to any information, which is presupposed by the stakeholder and we make it explicit, e.g. a location etc.. A domain ontology compliments this context information by precisely defining the terms (instances of context). We link context with an ontology annotation (shown as @) via an association link. As we can distinguish many contexts, we add precise context, where the definition of that is given in a domain ontology to avoid nesting or hierarchies of context.

While discovering tasks and context that can satisfy requirements, we may also identify **resources** that the tasks requires or need to use. We add *resource* node via (Requires) relation with each task. Note that resource concept is also available in other RML, however, we distinguish it as not only tangible e.g. mobile phone, Fax machine, but also intangible e.g. assets such as money, time, agenda. In our model, each resource may have domain assumptions or quality constraint attached to it via (I) node.

6- Modeling Influence and Relegation Relations:

Finally, identify during refinement requirements/tasks may have influence on the achievement of each other. *Influence relation* is added between a set of requirements/task, where the achievement of the former becomes critical due to the achievement of others (strong influence i.e. s-inf). If achievement of the latter is not critical, it will be modeled as weak (w-inf). However, it becomes interesting in case of tasks, where execution of one tasks may have influence of other tasks.

Finally, we look for conflicting context conditions, resource availabilities, quality criteria which may helps to determine requirements/tasks whose achievement can be delayed or relaxed. We add *relegation relation* between requirements/task that are less critical to the requirements/task more critical/preferred to in corporate uncertainty about changes in context or resource availability.

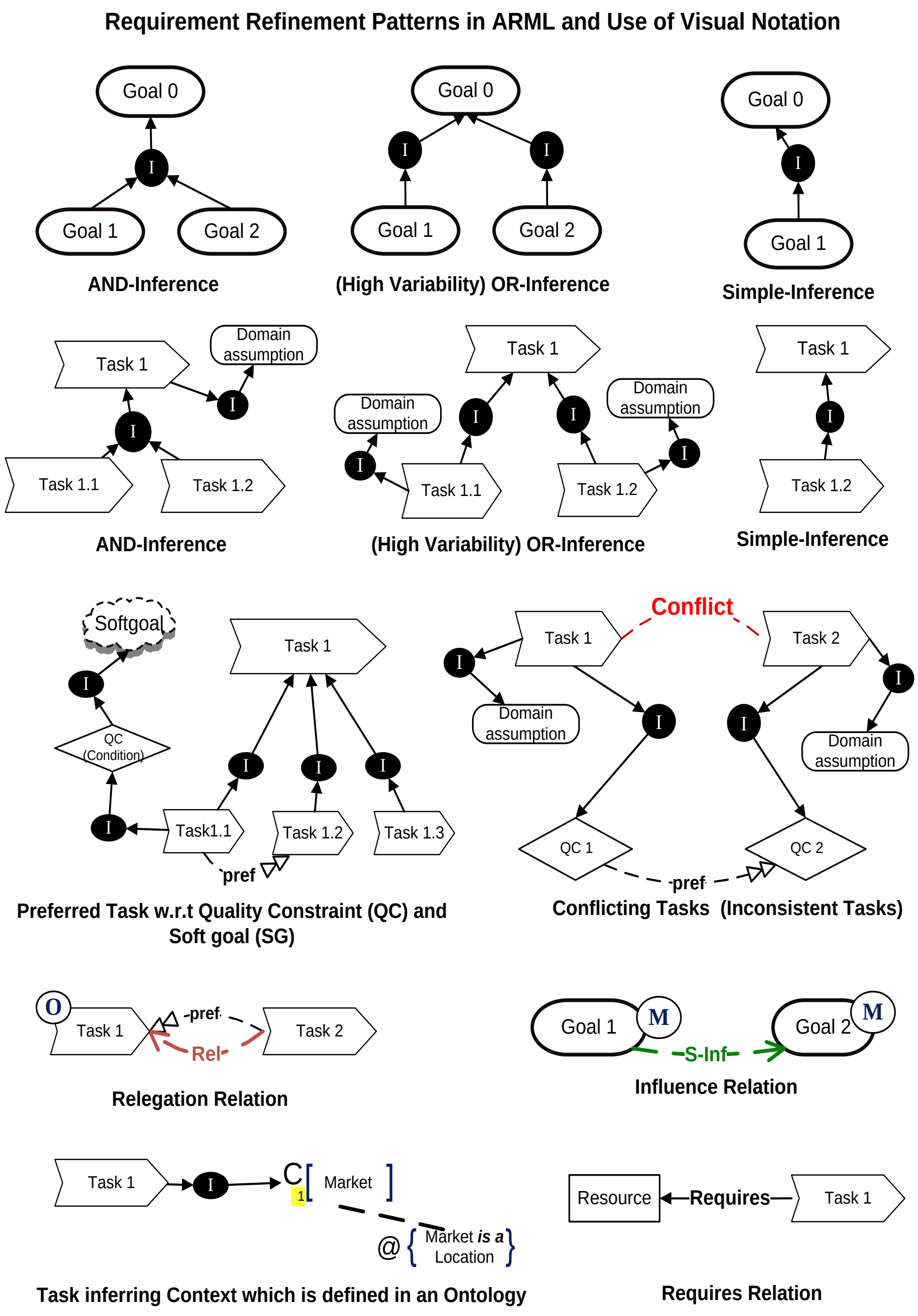


Figure 5.2: Requirements Refinement Pattern in Adaptive RML.

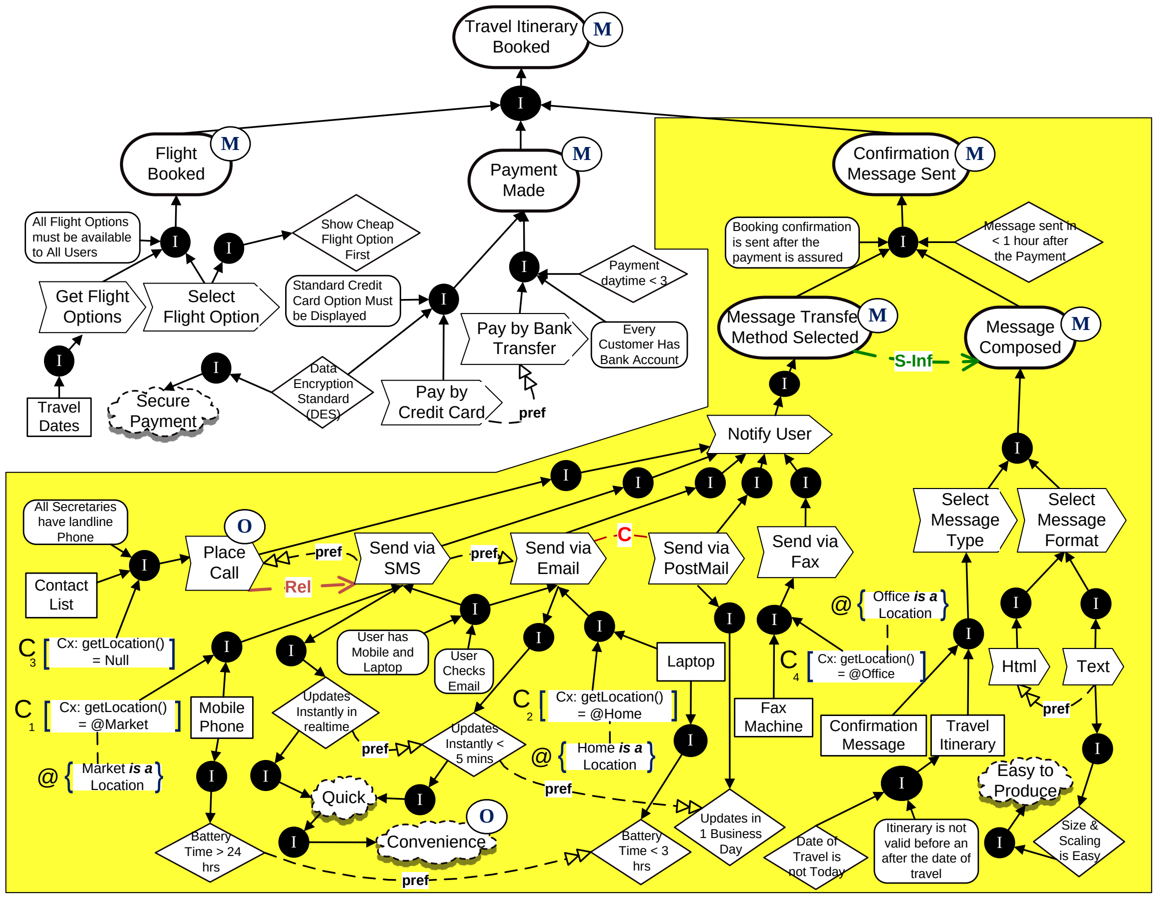

Travel Itinerary Booked
Flight Booked
Payment Made
Confirmation Message Sent
All Flight Options must be available to All Users
Show Cheap Flight Option First
Standard Credit Card Option Must be Displayed
Payment daytime < 3
Booking confirmation is sent after the payment is assured
Message sent in < 1 hour after the Payment
Get Flight Options
Select Flight Option
Pay by Bank Transfer
Every Customer Has Bank Account
Message Transfer Method Selected
Message Composed
S-Inf
Travel Dates
Secure Payment
Data Encryption Standard (DES)
Pay by Credit Card
pref
Notify User
All Secretaries have landline Phone
Place Call
Send via SMS
Send via Email
Send via PostMail
Send via Fax
Select Message Type
Select Message Format
Contact List
Rel
C
Cx: getLocation() = Null
Cx: getLocation() = @Market
Market is a Location
Mobile Phone
User has Mobile and Laptop
User Checks Email
Updates Instantly in realtime
Updates Instantly < 5 mins
Cx: getLocation() = @Home
Home is a Location
Laptop
Office is a Location
Cx: getLocation() = @Office
Fax Machine
Confirmation Message
Travel Itinerary
Html
Text
Easy to Produce
Quick
Convenience
Battery Time > 24 hrs
Battery Time < 3 hrs
Updates in 1 Business Day
Date of Travel is not Today
Itinerary is not valid before an after the date of travel
Size & Scaling is Easy

Figure 5.3: Modeling using Adaptive RML Concepts and Relations.

5.3.4 Requirements Modeling with Adaptive RML

For modeling requirements, we rest on our revised core ontology of RE for SAS. The proposed modeling language for SAS, called Adaptive RML, builds on Techne by adding two new concepts, namely, context and resource, and two relations, i.e. relegation and influence. Adaptive RML has its own visual notation. In the rest of this section we illustrate an Adaptive RML model of iComp with the aim to provide a preliminary qualitative evidence about its support in overcoming the limits mentioned above in modeling requirements for SAS. A detailed account of Adaptive RML will be given in the ensuing sections.

Fig.5.3, shows a requirements model for iComp in Adaptive RML. Its root level goal `Travel Itinerary Booked` is modeled as a mandatory node (*modeled as **M** node, a unary relationship*). It is decomposed via an inference relation into the other mandatory goals: `Flight Booked`, `Payment Made` and `Confirmation Message Sent` (*modeled as black **I** node with a arrow, a binary relation*), to represent the fact that it will be satisfied through the joint satisfaction of these three goals.

Let's focus on the goal: `Confirmation Message Sent` (i.e. the shaded part of the model), which is decomposed into two goals: `Message Transfer Method Selected` and `Message Composed` via inference relation. We can add here information, i.e. the domain assumption `Booking Confirmation is sent after the payment is assured` (*modeled as rounded rectangle*) and the quality constraint `Message sent in < 1 hour after the payment` (*modeled as diamond shape*) connecting them through the same inference node.

An influence relation is added among the two decomposed goals: `Message Transfer Method Selected` and `Message Composed` (*modeled as dotted green line with arrowhead*) to account for the prevailing context conditions and resource availability that influences the achievement of goal: `Message Composed`. For example, if the context conditions support to choose Email as a candidate transfer method, the ways to satisfy goal:`Message Composed` is by selection a correct format that is either text or html.

The analysis of the `Message Transfer Method Selected` proceeds by linking via inference nodes task-rooted subgraphs, which defines candidate solutions. For instance, the goal is decomposed into task e.g. `Notify User` via inference relation. Beside tasks e.g. `Send via SMS` (*modeled as hollow motion arrow*), each candidate solution includes domain assumptions e.g. `User has mobile and laptop`, context e.g. `Market`, `Home` (*labeled as **C** with its number, associated to @ symbol* [2]), and resources e.g. `Mobile Phone` (*modeled as a*

[2] @ labels a concept defined in domain ontology e.g. travel ontology.

rectangle). Preferences (*dotted line with doubled empty arrow heads*) are used to compare requirements in candidate solutions, and thereby compare candidate solutions; e.g., `Send via SMS` is preferred over `Send via Email`. Requirements can be in conflict (e.g. `Send via Email` is in conflict with `Send via PostMail` (*dotted line with C in the middle with red color*). Conflict is shown due to the difference in quality constraint (e.g. Email updates in >5 mins, whereas post mail updates in 1 business day).

Optional solutions, in case of problems (e.g. user is not accessible, as mentioned in the scenario) can be identified via a relegation relation (*modeled as dotted light red line with arrowhead* between two possible candidate solutions). For instance, `Place Call` relegates `Send via SMS`. This allows to take into account the situation in which a user's context changed resulting in being not accessible (e.g. Context: getLocation()= Null), and to describe as preferred the solution to make the user able to access the resources `Confirmation Message` and `Ticket Itinerary`, via contacting her secretary. The `Place Call` task is inferred via a domain assumption (e.g. `All secretaries has landline phone`) and a resource (e.g. `contact list`) and the context (e.g. Context: getLocation()= Null).

5.3.5 Analyzing Adaptive Requirements

Analysis in Adaptive RML suggests which candidate solutions are relevant in the prevailing context conditions and resource availability. A requirements model defines the requirements problem for a SAS-to-be, along with candidate solutions. This model is used by the analyst to discover "Adaptive requirements" by looking at differences between candidates solutions that are modeled.

Adaptive requirements are requirements that not only hold the definition of functional or non-functional requirements but encompass the notion of variability, by having monitoring specification, evaluation criteria and adaptation alternatives. To discover them detailed analysis is performed on the available information represented in the early requirements model. We analyze the candidate solutions that remain valid in a particular situation. We look at the context nodes and domain assumptions, we anticipate changes as we move to a different context and this leads to different resource availability requirements. Alternative solutions can be inferred during this process.

Adaptive requirements help specifying alternative ways to adapt to context and resource changes via a pattern, details of which are shown in Fig.5.2. Consider, while monitoring run-time changes, SAS moves across different contexts by altering the requirements problem that leads to change in candidate solutions. At run-time, several solutions get activated based on context and based on resource availability. Mechanisms for adaptation are triggered, therefore, reasoning over the adaptive requirement leads SAS moves (i.e. enact adaptation) to the candidate solution which is appropriate to the new current context.

For example, an adaptive requirement can be defined as **AR1**: *Message must be composed by selecting an appropriate format*. From this we determine that appropriate format i.e. HTML or Text, needs to be selected as modeled in Fig.5.3. But to select the candidate solution, we need to *monitor* the user's context (e.g. Office, Home) and resources (e.g. Mobile phone or Laptop) and domain assumptions with quality preferences. Along monitoring specification, we need also to specify *evaluation criteria* to check the difference between two tasks. Based on this criteria, among the possible candidate solutions that are adaptation actions *e.g. tasks and domain assumptions in a context*, a possible candidate solution will be selected. For instance, while monitoring the user context, resource, any change can lead to change the selected format, i.e. either html or text format.

So far, we argued on the need of a requirements modeling language (RML) for SAS that enable the analyst to capture and analyze requirements for

SAS by incorporating the above core properties of SAS at early stages of RE. Below we present how the adaptive requirement are operationalized. By operationalizing we mean, how conveniently we specify *adaptive requirements* such that monitoring specification, evaluation criteria and actions that are needed to support adaptation can be derived from them. Given that, SAS at run-time tries to resolve a run-time requirements adaptation problem, by finding and comparing a candidate solution in response to changing context, resource variability using its own requirements model and detailed specification i.e. adaptive requirements.

5.4 Operationalizing Adaptive Requirements

So far, we have introduced the concept of “adaptive requirements” and showed how to model them. In this section, we define the process to operationalize these requirements into a monitoring specification, which can be used to configure monitors for a running SAS. Aside this, evaluation criteria are needed to check weather SAS is violating its requirements or if parameters are changed, for which adaptation is needed by selecting a feasible action. We have developed a prototype tool that formalizes the process to operationalize the adaptive requirements as shown in the Fig.5.4. In our case we have chosen a monitoring system to support this is SALmon [OFMA08]. It is a framework that focuses on monitoring the quality of service (QoS) of web services, evaluate them accordingly to stated conditions, and notify violations to the interested parties. For operationalizing adaptive requirements – deriving monitors from the requirements, SALMon has been extended with new measurement capabilities, such as monitoring the change of status of a service, which goes beyond QoS. SALMon is able to combine both passive monitoring and testing approaches accordingly as per the preference. The framework has been implemented as a SBA itself, providing hence easy integration with other frameworks. It provides the following two services: the Monitor, responsible to retrieve the data of the target services; and the Analyzer, responsible for the evaluation of conditions.

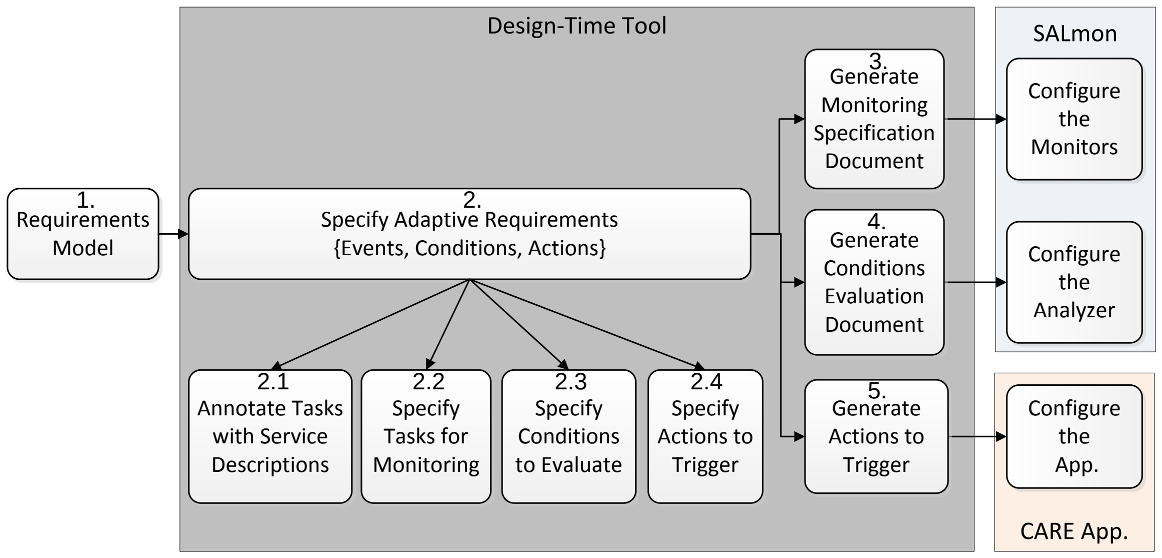

Design-Time Tool
SALmon
CARE App.
1. Requirements Model
2. Specify Adaptive Requirements {Events, Conditions, Actions}
2.1 Annotate Tasks with Service Descriptions
2.2 Specify Tasks for Monitoring
2.3 Specify Conditions to Evaluate
2.4 Specify Actions to Trigger
3. Generate Monitoring Specification Document
4. Generate Conditions Evaluation Document
5. Generate Actions to Trigger
Configure the Monitors
Configure the Analyzer
Configure the App.

Figure 5.4: Design-time process for deriving and configuring Monitor, Analyzer and CARE Application

In addition, we adopt a convenient *Event-Condition-Action* pattern to specify adaptive requirements. For instance,

$$MON : \textit{event} \wedge EVAL : \textit{conditions} \rightarrow TRIGGER : \textit{actions}$$

Below we describe this pattern using travel scenario examples and describe how we use this pattern to specify adaptive requirements from the requirements model.

Specifying Events:

The analyst can include either goals or tasks to monitor (i.e. MON). The analyst navigates through the given defined elements in the requirements model until it reaches the leaf tasks that implement the functionality i.e. tasks, when executed generates events to observe. For instance, from a high-level goal *changes over the flight itinerary being monitored*, the task to *Get Flight*

Status, which is operationalized with a flight booking service, need to be monitored. We specify events pertaining to task e.g. *Flight Status Changed*. Events can be represented in a domain ontology, which is defined or can be reused (e.g. travel domain ontology). Subsequently, to specify the monitoring behavior and to configure the monitor in SALmon, we define a low-level task: *invoke flight service*. The current tool prototype supports the generation of monitors for web services that the SAS is aware of. In order to automate the generation of monitors, the analyst annotates these tasks with the required information (i.e. endpoint, WSDL and SOAP action). The invocation of task:*Get Flight Status* generates events to monitor. The concrete properties of the monitor on each event are obtained from the requirements model through quality constraints defined via inference to the tasks e.g. response time of the service. Regarding the time interval for monitoring, it is up to the analyst to set it, or acquired through end-user preferences e.g. Notify flight status every 20 minutes. Such preferences helps in determining the behavior of the monitor. SALmon facilitate the strategy to monitor either actively i.e. invoking the service every time-interval or passively i.e. observing the interaction between the system and the end-user. Given this information, monitoring specification document is automatically generated i.e. an XML file that describes what is to be monitored, and is used in order to generate and configure the monitors accordingly.

Specifying Conditions:

Conditions provides the precise criteria that must be evaluated to check if there are violations. In the requirements models, conditions are specified via quality constraints that are linked to other elements via inference relation. In fact, quality constraints are the requirements with a precise metric to evaluate or a property with a defined value to be checked otherwise. After specifying the events to monitor, the analyst can specify (EVAL) with the relevant conditions that need to be checked based on the monitored data.

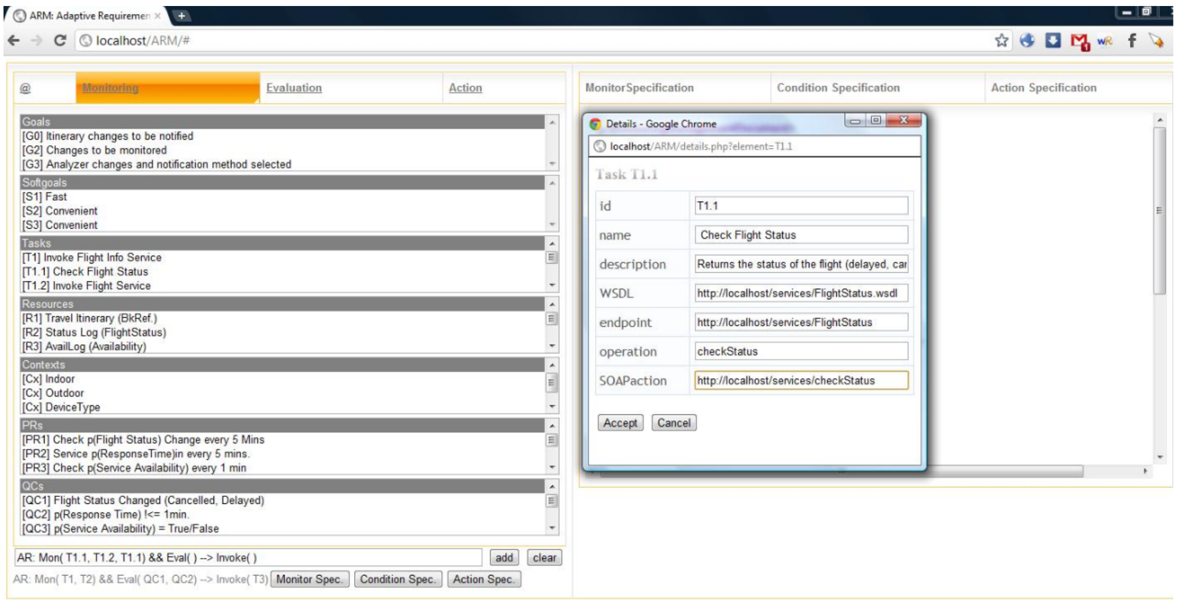
The current tool prototype provides a way to generate condition evaluation document that is used by the analyzer in SALMon to detect if the behavior of the system fulfills the expected functionality with the desired constraints. Conditions are specified using a tuple of <variable, operand, value >. In this tuple, a variable: is either a quality metric or the result of a service method. A quality metric can be either a basic metric or a derived metric based on an aggregation function (i.e. average, maximum, minimum). An operand: is a simple mathematical symbol expressing the comparative rule (i.e. <, <= , = , >= , >). The value: is a numerical, boolean or string value.

This information is used to automatically generate of the condition evaluation document, which is in XML format that describes what the analyzer has to evaluate.

Specifying Actions:

This part consists the execution of an action over the defined elements in the model. There are several kind of actions that can be performed in order to correct or mitigate the malfunction of the system. Currently we have focused on two kind of actions to perform over the requirements model. Namely, SELECT and INVOKE. Operationalizing the SELECT(task): the element included as a parameter in the SELECT function is a composite task that can be met by several alternatives. This action defines the preferred alternative to execute at run-time. For instance, in the given scenario, there is a task 'Notify User' composed of several alternatives (e.g. Send via Email, Send via SMS,etc). When a condition over these tasks is not met, the action SELECT(Notify User) is triggered, which updates the selection of the most convenient and available device to notify the user. INVOKE(task): the element included in the INVOKE function is a task that is executed by the system as a result of the failure of the condition. For instance, if the flight has been delayed, INVOKE(Notify User) notifies the user to his most convenient and available device (e.g. Mobile phone) that the flight has been delayed. The set of defined actions are used to generate the actions specification.

5.4.1 In Practice



Goals
[G0] Itinerary changes to be notified
[G2] Changes to be monitored
[G3] Analyzer changes and notification method selected

Softgoals
[S1] Fast
[S2] Convenient
[S3] Convenient

Tasks
[T1] Invoke Flight Info Service
[T1.1] Check Flight Status
[T1.2] Invoke Flight Service

Resources
[R1] Travel Itinerary (BkRef.)
[R2] Status Log (FlightStatus)
[R3] AvailLog (Availability)

Contexts
[Cx] Indoor
[Cx] Outdoor
[Cx] DeviceType

PRs
[PR1] Check p(Flight Status) Change every 5 Mins
[PR2] Service p(ResponseTime)in every 5 mins.
[PR3] Check p(Service Availability) every 1 min

QCs
[QC1] Flight Status Changed (Cancelled, Delayed)
[QC2] p(Response Time) !<= 1min.
[QC3] p(Service Availability) = True/False

AR: Mon(T1.1, T1.2, T1.1) && Eval( ) --> Invoke( )

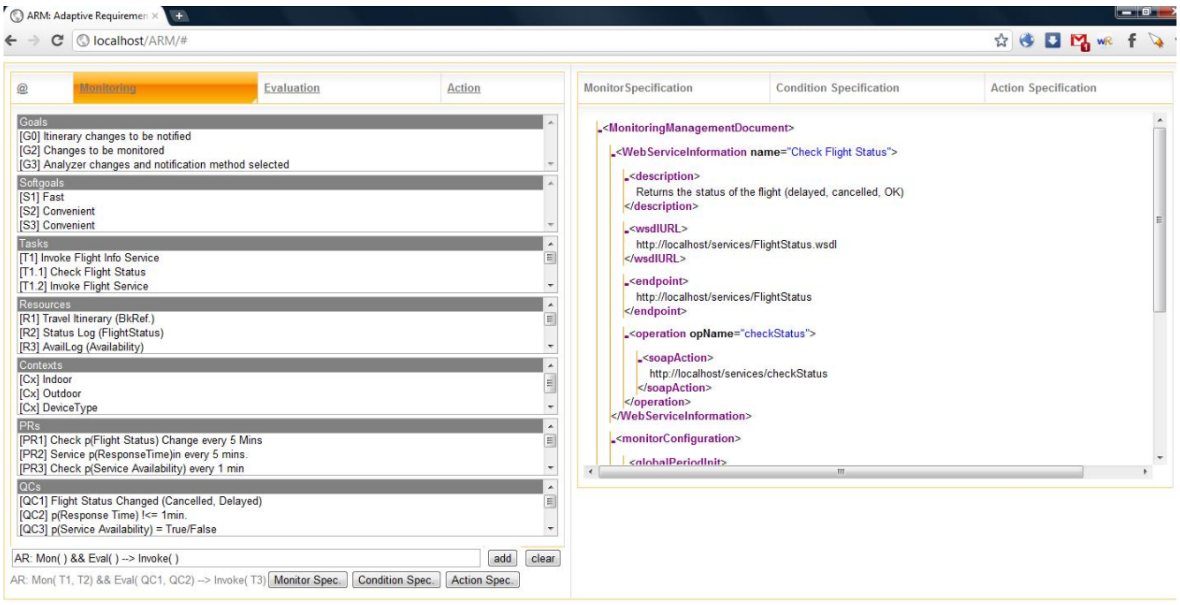
AR: Mon(T1, T2) && Eval(QC1, QC2) --> Invoke(T3)

Task T1.1

id	T1.1
name	Check Flight Status
description	Returns the status of the flight (delayed, car
WSDL	http://localhost/services/FlightStatus.wsdl
endpoint	http://localhost/services/FlightStatus
operation	checkStatus
SOAPaction	http://localhost/services/checkStatus

Figure 5.5: Design-time process for Annotating the Tasks

At design-time, the analyst can conveniently import the requirements model (step 1) into the design-time tool. The tool loads the elements of the model in the GUI and presents a categorized view of the elements. Using the above ECA pattern the analyst then specify adaptive requirements (step 2). The first step is to annotate the leaf tasks with the requires service descriptions (step 2.1). A snapshot of the tool for annotating the tasks with service descriptions is shown in Fig.5.5. The tasks are added in the ECA pattern for monitoring i.e. specifying events to be monitored (step 2.2). Correspondingly, a monitoring specification document can be generated automatically, which is then used to configure monitors in SALMon (step 3). A snapshot of the tool for deriving the monitoring specification document is shown in Fig.5.6. Next to this, is to specify conditions, where an analyst can add quality constraints relating to the tasks. This determines which criteria has to be evaluated and the associated property (step 2.3). After adding this information in the ECA pattern, the condition evaluation document can be generated to configure the analyzer in SALMon (step 4). Finally, to specify actions, the analyst can browse the elements in the requirements model i.e tasks to be triggered in case the conditions are not met (step 2.4). This generates and action specification, which is then used to enable adaptation decisions in the running SAS (step 5). The outcome of this process is a set of adaptive requirements that are



Goals
[G0] Itinerary changes to be notified
[G2] Changes to be monitored
[G3] Analyzer changes and notification method selected

Softgoals
[S1] Fast
[S2] Convenient
[S3] Convenient

Tasks
[T1] Invoke Flight Info Service
[T1.1] Check Flight Status
[T1.2] Invoke Flight Service

Resources
[R1] Travel Itinerary (BkRef.)
[R2] Status Log (FlightStatus)
[R3] AvailLog (Availability)

Contexts
[Cx] Indoor
[Cx] Outdoor
[Cx] DeviceType

PRs
[PR1] Check p(Flight Status) Change every 5 Mins
[PR2] Service p(ResponseTime)in every 5 mins.
[PR3] Check p(Service Availability) every 1 min

QCs
[QC1] Flight Status Changed (Cancelled, Delayed)
[QC2] p(Response Time) !<= 1min.
[QC3] p(Service Availability) = True/False

AR: Mon( ) && Eval( ) --> Invoke( )

AR: Mon(T1, T2) && Eval(QC1, QC2) --> Invoke(T3)

```
<MonitoringManagementDocument>
  <WebServiceInformation name="Check Flight Status">
    <description>
      Returns the status of the flight (delayed, cancelled, OK)
    </description>
    <wsdlURL>
      http://localhost/services/FlightStatus.wsdl
    </wsdlURL>
    <endpoint>
      http://localhost/services/FlightStatus
    </endpoint>
    <operation opName="checkStatus">
      <soapAction>
        http://localhost/services/checkStatus
      </soapAction>
    </operation>
  </WebServiceInformation>
  <monitorConfiguration>
    <globalPeriodInit>
```

Figure 5.6: Deriving Monitoring Specification Document

operationalized to configure monitors, analyzer of SALMon framework and for the SAS-to-be, instantiating CARE framework. We discuss the initial tool developed for this and present some initial preliminary results gathered on the tool performance and scalability in Chapter 7.

5.5 Related Work

Recent works on software engineering for SAS have identified several open research challenges in [CdLG+09]. Along the engineering perspective Burn et al. [BSG+09] recognize the need for making explicit the feedback control loop at a higher level of abstraction. A recent proposal on awareness requirements has been argued recently in [SLRM10], which focuses on the the requirements for feedback loops. In this case, our proposal of adaptive requirements and the process to operationalize them makes explicit the feedback loop functionalities – monitoring specification, evaluation criteria and adaptation actions, not only at the requirements level but also at the operational level supporting the analyst.

Along the well recognized issues in RE for SAS, which includes: four levels of RE for SAS [BCZ05], adaptive requirements are engineered at design-time to identify and specify requirements for run-time adaptation – conforming to level 1 and 3. In a recent proposal, a requirements language (RELAX) is proposed to deal with uncertainty in requirements with respect to the environment conditions [WSB$^+$10]. The language provides temporal operators to handle uncertainty. The semantics of these operators are formalized in Fuzzy Branching Temporal Logic. A set of analysis methods are then provided to support goal modeling refinement towards detailed design, which exploits mitigation strategies based on obstacle analysis, and lead eventually to relax constraining conditions (i.e. our quality conditions) [CSBW09a]. In this case our proposal of adaptive requirements and its modeling in Adaptive RML provide the Relegate relation, which is more general than the RELAX operators. Since we do not commit to fuzzy logic, we only ask for a way to represent alternatives and to compare them. In this sense, RELAX can be seen as a particular way to relax requirements (a particular way to implement the Relegate relation).

Taking requirements modeling perspective, Sawyer et al. [SBH$^+$07] propose four levels of LoREM. In this work, alternative configurations for dynamically adaptive systems (DAS) are modeled using softgoals and dependencies among different components of the DAS using *i** by different categories of designers/developers. Our proposal is near to this proposal, however, we focus of modeling adaptive requirements using a rich set of concept, where variability in the goal model is considered along domain assumptions, quality constraints, context and resources. Thus, specifying adaptive requirements makes more explicit the feedback functionalities. Analogously, in the approach by Morandini et al. in [MPP08], requirements for adaptation are modeled using an existing goal-modeling language with proposed extension (e.g. goal types, conditions and failure modeling) and later transforming goal models as an agent capabilities using BDI architecture. In this case, our proposal is similar in respect of modeling explicit information relevant to requirements for SAS, but we provide more extended ontology with more explicit concepts and relations than them. Moreover, specification of adaptive requirements is different in a sense that they target the feedback functionalities.

In addition, the dimension of ***context*** in RE for SAS is important. It has been argued that alternative behaviors must be supplied to the system enabling them to switch from one to another by overcoming the changes in the environment, while monitoring the context [SYN07b]. To capture the contextual variability, explicit knowledge about the domain is required. In [ADG09] proposed variation points as a way to annotate the goal models to represent pre-defined contexts and alternative behavior to be exploited while reasoning over them. To use this approach, a requirements driven reconfiguration architecture is proposed leveraging the concept of context and monitor-diagnosis-compensate loop in [DGM09]. However, the notion of context is trickier and brings newer requirements to be analyzed while specifying requirements for SAS.

In ARML, we provided an explicit graphical notation, where context properties can be modeled taking into account the domain ontology, which defines the domain concepts and their instances. Moreover, our Adaptive requirements, follow similar ideas, but go beyond the above mentioned approaches by making explicit domain assumptions and requirements for feedback loops.

On the basis of recent works, we recognized issues in requirements modeling for SAS that provides premise to the proposal of Adaptive RML. For instance ***requirements monitoring*** [FF95, FFVLP98, Rob09], where the running systems must be monitored during its execution as per its own requirements model, any run-time deviation or violation leads to the modification to reconcile its behavior to its requirements. In case of SAS this is critical, as it operates in an open environment where changes can occur dynamically in the operating context, availability of resources and end-user needs can change over time. For this reason, we adaptive requirements are modeled explicitly in Adaptive RML, in which we model requirements for SAS that guides the detailed specification, which will eventually include monitoring specification, evaluation criteria and adaptation action.

Requirements reflection is another issue, where ideas from computational reflection has been borrowed to provide SAS the capability to be aware of its own requirements [SBW$^+$10]. Similarly, online goal refinement [KM07] is of prime importance considering the underline architecture of the intended SAS. To support run-time reasoning of requirement by SAS itself, adaptive requirements are identified at an early stage and modeling them with explicit concepts supports the formulation of the run-time requirements adaptation problem for SAS (discussed in Chapter 3). This leads to more detailed specification which is operationalized in terms of concrete feedback loop functionalities. Thus, providing ample support for online refinement and requirements awareness.

More recently, emerging communities such as requirements@run-time and models@run-time have started to focus on the issues such as run-time management of the requirements, their representation and determining the boundary between design-time and run-time. Our proposal of adaptive requirements is in line with these ideas.

5.6 Final Remarks

In this chapter we introduce a new aggregate type of requirement has been proposed, *adaptive requirements*, which are relevant for RE for SAS. We proposed a visual requirements modeling language – Adaptive RML that provides necessary concepts and relations in line with the core ontology of RE for SAS (see Chapter 3). In contrast to recent proposals [MPP08, CSBW09b, BPS10] that rest on well established goal-oriented modeling languages (i.e. i*, Tropos, Kaos), Adaptive RML builds on the abstract requirements modeling language Techne [JBEM10b], which provides a richer set of concepts, along the core ontology for RE defined in [JMF08], and supports requirements analysis leading to sets of candidate solutions for the stated requirements problem. Additional concepts and relationships are used in Adaptive RML (i.e. context, resource, relegation and influence) to model and represent the run-time requirements adaptation problem and perform analysis using these models.

In addition, we provide a tool supported process to operationalize adaptive requirements. A prototype tool in support of the analyst is built that provides a convenient way to process to derive monitoring specification, evaluation criteria and adaptation actions using the requirements models. Adaptive requirements are specified adopting the Event-Condition-Action pattern, which provide an easy way to operationalize them. Focusing on RE for SAS, work on a RE case tool will be the further steps of this research.

Chapter 6

Application of the CARE Framework

6.1 Overview

The objective of this chapter is to study the application of CARE framework to a travel case study, realizing a small proof of concept application to illustrate the process. The resulting prototype application provides initial results on realizing the CARE architecture and to acquire further improvements identified during the implementation. We refer to our CARE's conceptual architecture (in Chapter 4.4) and the concepts and artifacts that CARE framework prescribes. We first illustrate the description of the system using the case study scenario. Following our design-time approach we model the requirements (using ARML) and then specify adaptive requirements. For run-time, we discuss the prototype developed to execute the given scenario under discussion. With this we show in practice that how CARE can be applied and to justify to some extent that continuous RE can be realized involving the end-user. We then discuss the lesson learned from this experience and propose further steps.

The rest of the chapter is organized as follows. In Section 6.2, we discuss the description of the scenario adopted from the travel companion case study. In Section 6.3 we model the requirements acquired using the system description and specify adaptive requirements. In Section 6.4, we present the prototype that instantiates CARE's conceptual architecture at run-time. In

Section 6.6, we discuss our experience and propose further steps. Finally, in Section 6.7 we summarize this chapter.

6.2 Description of the Travel Companion

Travel Companion (hereafter, iComp) is a *self-adaptive software*. It manages end-users travel and meeting schedules while on the go. It supports the end-user to specify their preferences and receive timely information about their booking status (e.g., confirmed, canceled, in progress). The booking preference is collected using the iComp GUI. Once end-user has specified her travel itinerary (e.g. itinerary number, time of the flight, origin, destination and service source etc.), iComp starts to monitor the relevant service for checking the flight status (e.g. flight service). The notification message about the flight status must be sent to the end-user as per her preferences i.e. via Email, or Call, or SMS (i.e. most preferred method) instantly (i.e. in less than an hour before the flight) on her device (i.e. laptop or mobile) depending on her context (i.e. home or market or office etc.) and system context (e.g. the Internet is available, service is available etc.). The notification message must be sent to end-users device by selecting a suitable message format (i.e. size, scaling, format) depending as per her device context (e.g. smart phone). In case there are some problems (i.e. user is not accessible, network is not available, or device is not reachable), the notification message must be ensured as sent by adopting an alternative method e.g. sending to alternative contact (if given) e.g. office secretary, or apply a retry strategy (e.g. attempt to resend the message after every 5 minutes than 10 minutes etc.) until the message is confirmed to be delivered.

The “iComp” will ideally be deployed on the mobile phone of the end-user as a thin client. The main goal of “iComp” is to support the end-user’s by adapting to new candidate solutions (e.g. using the available services) in response to changes in their requirements or preferences. The case described above demands iComp to adapt while monitoring the “flight status” of a itinerary with a specified “booking reference”, which the end-user provide while customizing the travel details in “iComp”. In case of any changes, e.g., any event caused during service invocation, change in user flight status etc. the “iComp” chose at best the candidate solutions (e.g. Notify user via SMS or Email using the available services). In case the mobile battery is going down, it notifies the thick client host to adopt an alternative mean to notify end-user in case of changes.

For example, while the end-user is traveling, i.e., her current context is identified by gathering data about her location using the location sensing in the her smartphone. Conversely, if she is indoor e.g. office, the “iComp” has to adapt itself (making the mobile silent or if there is any event to be notified it notifies via email, etc.) to be able to accurately inform the end-user about changes in her itinerary.

6.3 Design-Time Modeling of Requirements

We have adopted and revised the scenario from the travel case study. Here we add more details with respect to the specific scenario mentioned above. First, we need to represent explicitly the knowledge about the domain of discourse. For this reason we make explicit in our requirements model this information, which may appear redundant or trivial, but this is necessary when a running SAS needs to reason on this artifact. For instance, the user requirement is summarize as: *The notification message about the flight status must be sent to the end-user as per her preferences i.e. via Email, or Call, or SMS (i.e. most preferred method) instantly (i.e. in less than an hour before the flight) on her device (i.e. laptop or mobile) depending on her context (i.e. home or market or office etc.) and system context (e.g. the Internet is*

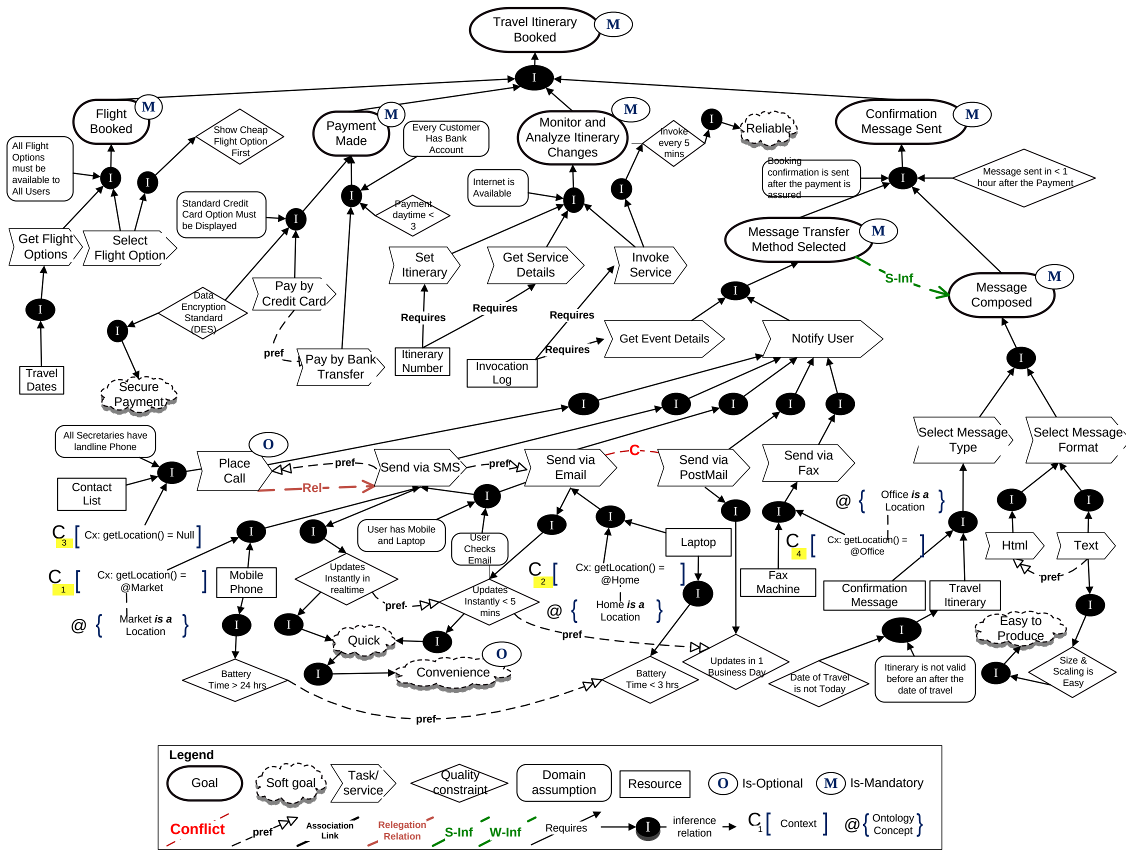


Figure 6.1: A Goal Model in ARML

available, service is available etc.). We model this case and discuss how ARML help in modeling this scenario and then we discuss how adaptive requirements can be specified through this model. In Fig. 6.1, we present a the high-level goals of “iComp” that are refined using the ARML concepts and notations. Following the guidelines described in Chapter 5. We start by eliciting information from the domain description, or the system details to identify domain assumptions, specific entities and conditions pertaining to the specific scenario under discussion. We represent this knowledge in a domain ontology, where all the domain concepts are defined e.g. Travel ontology.

Fig. 6.1, shows a requirements model for “iComp” in Adaptive RML. Its root level goal `Travel Itinerary Booked` is modeled as a mandatory node (*modeled as **M** node, a unary relationship*). It is decomposed via an inference relation into the other mandatory goals: `Flight Booked`, `Payment Made`, `Monitor and Analyze Itinerary Changes` and `Confirmation Message Sent` (*modeled as black **I** node with a arrow, a binary relation*), to represent the fact that it will be satisfied through the joint satisfaction of these three goals.

Let's focus on the goal: `Monitor and Analyze Itinerary Changes`, which is refined into three tasks: `Set Itinerary` and `Get Service Details` and `Invoke Service` via inference relation. We can add here information, i.e. tasks: `Set Itinerary` and `Get Service Details`, requires a resource i.e. Itinerary Reference (Itinerary Number – shown as a resource in the model). Further we add information related to the softgoals. Notice, task: `Invoke Service` has a precise quality constraint `Invoke service every 5 minutes` linked via inference node. This quality constraints satisfice the softgoal i.e. `Reliable` through an inference node. This means that task: `Invoke Service`, is reliable if it satisfies the quality constraint. During the invocation of the service, the task: `Invoke Service` logs the service invocation events and for this it requires a resource i.e. Invocation Log.

The analysis of the `Message Transfer Method Selected` is critical here, which is refined into two tasks: `Get Event Details` and `Notify User`. Subsequently, the task:`Notify User` is decomposed into tasks: `Send via SMS` (*modeled as hollow motion arrow*), each candidate solution includes domain assumptions e.g. `User has mobile and laptop`, context e.g. `Market`, `Home` (*labeled as **C** with its number, associated to @ symbol* [1]), and resources e.g. `Mobile Phone` (*modeled as a rectangle*). Preferences (*dotted line with doubled empty arrow heads*) are used to compare requirements in candidate solutions, and thereby compare candidate solutions; e.g., `Send via SMS` is preferred over `Send via Email`. Requirements can be in conflict (e.g. `Send via Email` is in

[1] @ labels a concept defined in domain ontology e.g. travel ontology.

conflict with `Send via PostMail` (*dotted line with C in the middle with red color*). Conflict is shown due to the difference in quality constraint (e.g. Email updates in >5 mins, whereas post mail updates in 1 business day).

Optional solutions, in case of problems (e.g. user is not accessible, as mentioned in the scenario) can be identified via a relegation relation (*modeled as dotted light red line with arrowhead* between two possible candidate solutions). For instance, `Place Call` relegates `Send via SMS`. This allows to take into account the situation in which a user's context changed resulting in being not accessible (e.g. Context: getLocation()= Null), and to describe as preferred the solution to make the user able to access the resources `Confirmation Message` and `Ticket Itinerary`, via contacting her secretary. The `Place Call` task is inferred via a domain assumption (e.g. `All secretaries has landline phone`) and a resource (e.g. `contact list`) and the context (e.g. Context: getLocation()= Null).

6.3.1 Specifying Adaptive Requirements

To manage the uncertainty, once we model the requirements and perform an early analysis of them, we need to specify adaptive requirements that aggregates primitive concepts and make explicit the feedback functionalities e.g. monitoring specification, evaluation criteria and adaptation tasks (see the model in Fig. 6.1). We adopt event-condition-action patterns to specify adaptive requirements. For instance, we take the example of the task: `Invoke Service`. Note that, when this task is invoked there may be several events that needs to be monitored triggering other tasks. Returning to the specification of adaptive requirements, we refer to the following ECA pattern:

$$MON : \textit{event} \land EVAL : \textit{conditions} \rightarrow TRIGGER : \textit{actions}$$

In the given scenario, we identified the "Adaptive Requirements" that an analyst can specify with the above pattern: For instance, task:`Invoke Service` is executed every 5 minutes to satisfice the softgoal `Reliable`. The execution of this task requires a service description, which is annotated using the design-time tool (see Section 5.4, for details). Here we are interested to show, how adaptive requirements will aggregate other requirements and are makes explicit the monitoring specification and evaluation criteria. Note that in the requirements model, evaluation conditions are derived through quality constraints. Adaptation actions are specified as triggers (e.g. INVOKE, SELECT functions) over tasks specified in the model. If the boolean result of EVAL is false, the consequence is to trigger the tasks as per user's preference. Actually, adaptive requirements can also aggregate other adaptive requirements, e.g. AR1 must hold in order AR3 not to be violated. Below we state the adaptive requirements related to the goal:`Monitor and Analyze Itinerary Changes`:

AR 0:

$$MON : onEvent(checkInternet()) \wedge EVAL : (DA : Internet(isAvailable)) \rightarrow TRIGGER : (INVOKE : NotifyUser(inform.immediate == False) \wedge SELECT : LogInfo())$$

In the above adaptive requirement, we use a domain assumption i.e. "Internet is Available" to specify a monitoring requirement. This requirement helps in the implementation to code the behavior of the monitoring component. In this case, we introduced a condition to be evaluated and an event to log the information. The monitor at run-time will not trigger task: `Notify User`, but just logs the information in this case.

AR 1:

$$MON : onEvent(SetItinerary()) \wedge EVAL : (currentTime \Leftarrow PreferredTime \wedge PreferredCommunicationMethod == SMS \wedge Reasource.ItineraryReference(isAvailable)) \rightarrow TRIGGER : (INVOKE : NotifyUser(inform.immediate == False) \wedge SELECT : LogInfo())$$

Here, we introduce a condition while executing task:`Set Itinerary` preferences and conditions to evaluate that are notification time, method to notify and availability of resource. These preferences provide requirements, which are helpful to derive monitoring specification and also to specify behavior of the evaluator component.

AR 2:

$$MON : onEvent(GetServiceDetails() \wedge AR1) \wedge EVAL : (Service(isAvailable)) \rightarrow TRIGGER : (INVOKE : NotifyUser(inform.immediate == False) \wedge SELECT : LogInfo())$$

In this adaptive requirement, we introduce a condition about the availability of the service. Monitor component can log information while invoking the service time to time. Further to specify this behavior we may also specify the time to invoke the service and get details about its availability and response time. For this reason, we rely on SALMon framework, which defines these metric to evaluate while monitoring a service.

AR 3:

$$MON : onEvent(InvokeService() \wedge GetEventDetails() \wedge AR2) \wedge EVAL : (invocationTime \leq 5min \wedge FlightStatus \neq LiveStatus) \rightarrow TRIGGER : (INVOKE : NotifyUser(inform.immediate == true) \wedge SELECT : SendViaSMS(isPreferred) \| SendViaEmail(isPreferred) \| PlaceCall())$$

This adaptive requirement aggregates another adaptive requirement (AR2), which is also monitored while invoking the service for getting the flight status. In face, the current flight status does not match with the “Live status”, the task:Notify user is invoked and one of the candidate solution i.e. SendviaSMS or SendviaEmail is executed depending upon the context conditions.

Specifying adaptive requirements in ECA pattern provides benefits in many ways. First, it raises the level of abstraction from low level if-then-else providing more reasoning capabilities. Second, it enables us to write adaptive requirements in a convenient way to express adaptive requirement by making explicit the monitoring specification, evaluation criteria and adaptation tasks.

Third, we can represent multiple instances of adaptive requirements as a solution to the requirements problem in the requirements database (Δ). Moreover, situations where a change in context demands change in the solution to the requirements problem can be well specified with adaptive requirements represented in ECA pattern. In addition, a *self-adaptive software* can overcome the uncertainty in the environment by reasoning over these adaptive requirements and performs adaptations with respect to adaptation Type 1,2 and 3.

6.4 Instantiating CARE at Run-Time

The requirements obtained through modeling process in the previous step are used to realize a prototype application ("iComp") that instantiates the CARE framework. For initial prototype we implement these requirements directly. We are interested in the adaptive behavior of the prototype application enabling us to gain first assessment on realizing CARE framework. In Fig. 8.1, we present the design-time view and the run-time view of the prototype. Design-time view depicts the approach we followed while modeling and specifying adaptive requirements. At run-time software agents exploits web services that a available to fulfill the service request (i.e. represented as RRA, see Section 4.3) given by the end-user.

The run-time view shows the actual working of the "iComp" as a set of interacting agents. Following the CARE architecture, we concentrated on the $UIAgent$, which provides a graphical user interface to the end-user to customize "iComp" and provide service requests as preferences. $UIAgent$ then interacts with the $Monitor-Agent$, which is responsible for monitoring the web services to fulfill the end-user service request. It looks up in the $Delta$ for checking Goal axioms and concepts axioms from the ontology to better match the end-user request with the available web services. Service invocation events are logged and $Evaluator - Agent$ is informed if there

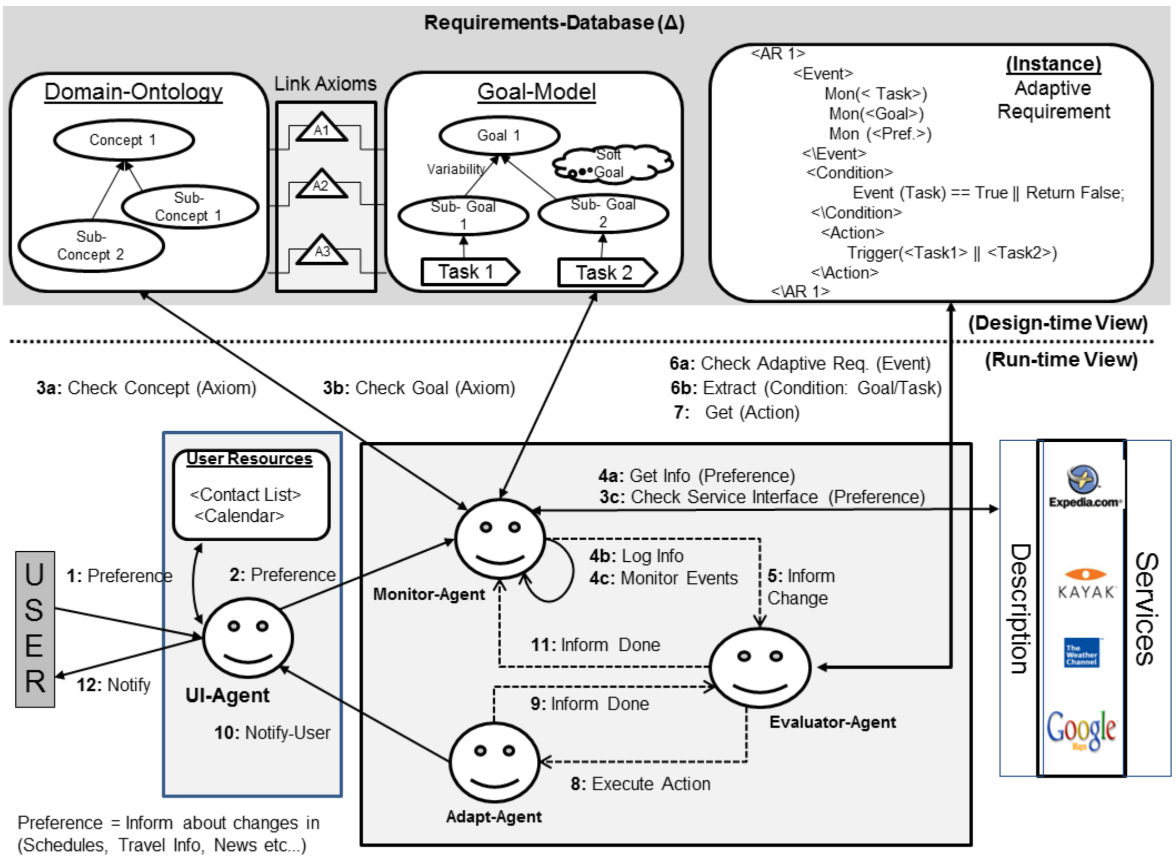


Figure 6.2: Run-time Process.

are any violations. $Evaluator - Agent$ is responsible for deciding about the best candidate solution to select. It looks up the $Delta$ for the instances of adaptive requirements that provides the rules to guide the selection. In case of any change or violation events detected by the $Monitor - Agent$ it sends execute commands to the $Adapt - Agent$. Note that this prototype is a limited version of the CARE architecture. Once acquiring execute command from $Evaluator - Agent$, the $Adapt - Agent$ notifies the $UI - Agent$ agent to provide feedback to the end-user.

In this prototype implementation, we acquired the service descriptions WSDLs and $Evaluator - Agent$ is aware of their methods. The core agents in this setting form the Monitor-Eval-Adapt control loop, which is guided by the adaptive requirements and end-user service requests. Each agent send

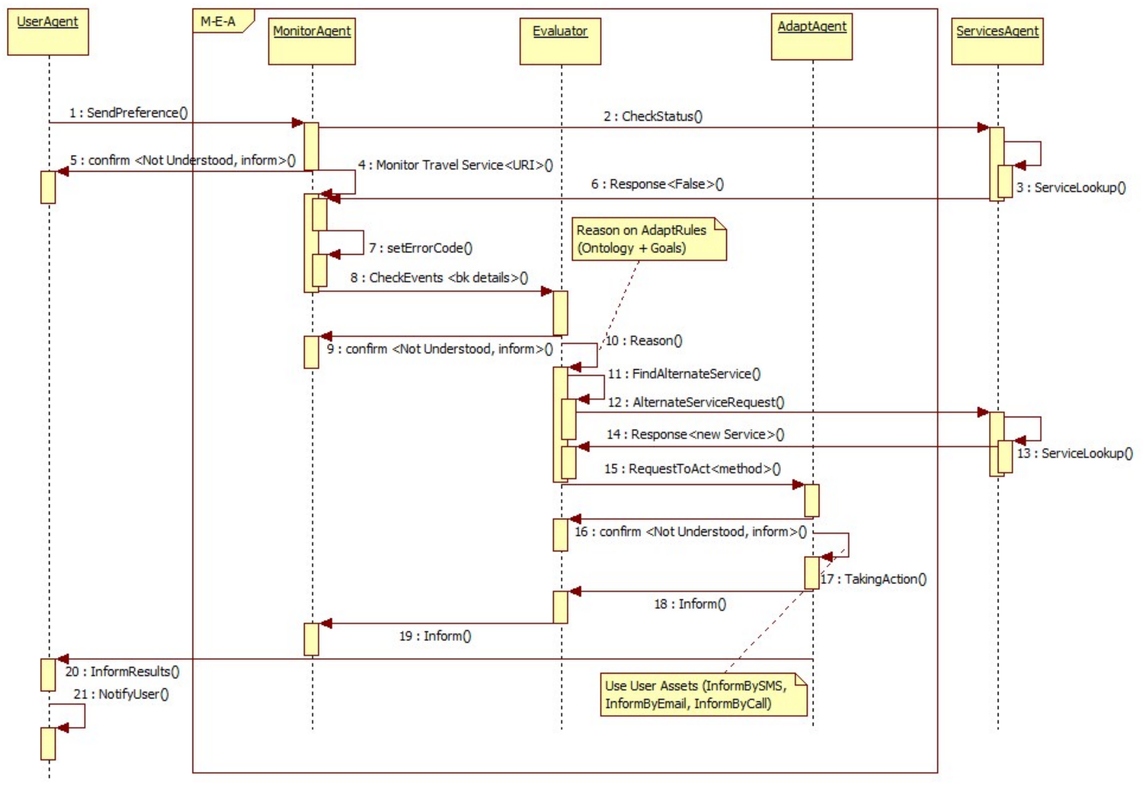

UserAgent
M-E-A
MonitorAgent
Evaluator
AdaptAgent
ServicesAgent
1 : SendPreference()
2 : CheckStatus()
3 : ServiceLookup()
5 : confirm <Not Understood, inform>()
4 : Monitor Travel Service<URI>()
6 : Response<False>()
Reason on AdaptRules (Ontology + Goals)
7 : setErrorCode()
8 : CheckEvents <bk details>()
9 : confirm <Not Understood, inform>()
10 : Reason()
11 : FindAlternateService()
12 : AlternateServiceRequest()
13 : ServiceLookup()
14 : Response<new Service>()
15 : RequestToAct<method>()
16 : confirm <Not Understood, inform>()
17 : TakingAction()
18 : Inform()
19 : Inform()
20 : InformResults()
21 : NotifyUser()
Use User Assets (InformBySMS, InformByEmail, InformByCall)

Figure 6.3: Sequence flow of the Scenario.

messages to provide feedback among themselves based on a common shared ontology of the domain i.e. Travel ontology.

To further understand the sequence of the execution at run-time, we formalize the scenario under discussion using the sequence diagram as shown in the Fig. 6.3. In this sequence diagram, we mainly elaborate the actual execution flow and interactions among the agents that are developed.

6.5 Analyzing Run-Time Adaptation

To perform continuous reasoning and refinement by iComp itself, it requires end-user customizations e.g. preferences new knowledge or input from the stakeholders (e.g. end-user) to refine its specification. To give an intuition about how the requirements specification can support run-time adaptation, in

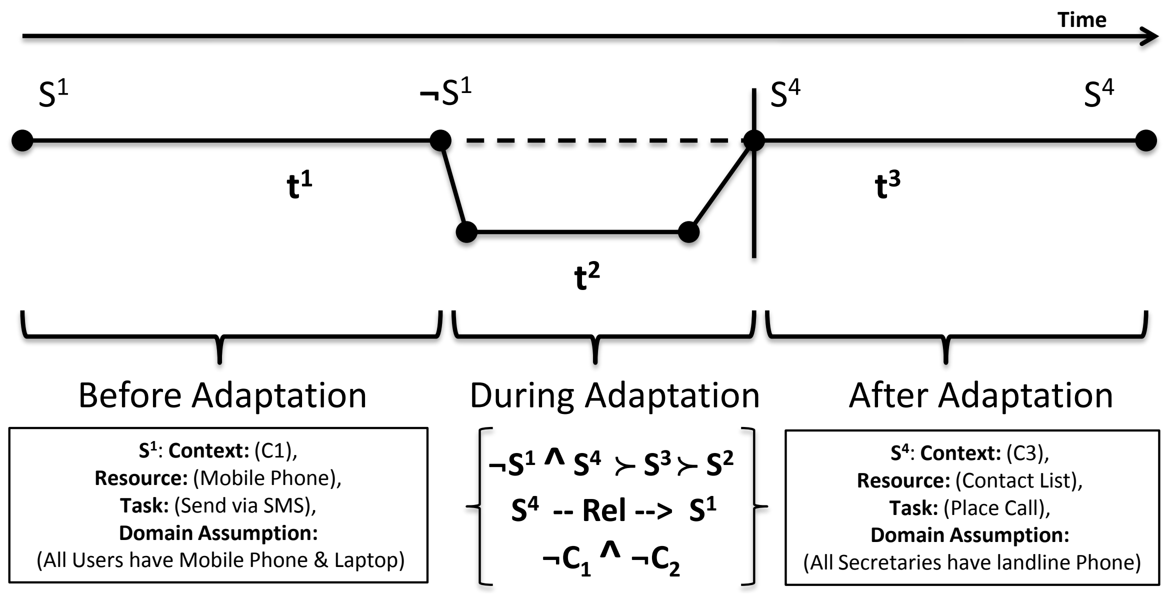

Time
S^1
$\neg S^1$
S^4
S^4
t^1
t^2
t^3
Before Adaptation
During Adaptation
After Adaptation
S^1: Context: (C1), Resource: (Mobile Phone), Task: (Send via SMS), Domain Assumption: (All Users have Mobile Phone & Laptop)
$\neg S^1 \wedge S^4 \succ S^3 \succ S^2$
S^4 -- Rel --> S^1
$\neg C_1 \wedge \neg C_2$
S^4: Context: (C3), Resource: (Contact List), Task: (Place Call), Domain Assumption: (All Secretaries have landline Phone)

Figure 6.4: Runtime Adaptation Sequence of SAS.

Fig.6.4, an adaptation sequence is shown along the time dimension, where the SAS operates as per the candidate solution (S1) selected to satisfy the particular context and resource variation. At this time (t1) the SAS, while monitoring, evaluates the user's current situation and attempts to satisfy a given set of goals (e.g. `Confirmation Message Sent`, `Message Transfer Format Selected`) and quality constraints via its candidate solution. A candidate solution is composed of tasks, domain assumptions that hold valid for a context and available resources to achieve such tasks.

E.g., candidate solution **S1**: *Context: (@Market), Resource: (Mobile Phone), Task: (Send via SMS), Domain Assumption: (All Users have Mobile Phone & Laptop)* was selected, but due to traveling, the context is not recognized anymore. Therefore the SAS has to reason about this change at time (t2) by looking at the difference in candidate solutions with respect to context conditions, resource availabilities and user preferences. SAS performs the reasoning based on the differences among the alternative candidate solutions, which states a comparison and ranking of the solutions based on criteria e.g.

(S1) Send via SMS is not valid, (S2) Send via Email is not feasible as user's context is not recognized. Thus the change in requirements problem, changes the candidate solution in different contexts and with different resources. The adaptive requirements play critical role here, as they operationalize the mechanisms for adaptation i.e. monitor and evaluating the difference between candidate solutions and provides criteria to compare and rank them. To reason on adaptive requirements, automated reasoning techniques (e.g. AI Planning) or efficient rule engines can be provide more deductive reasoning capabilities.

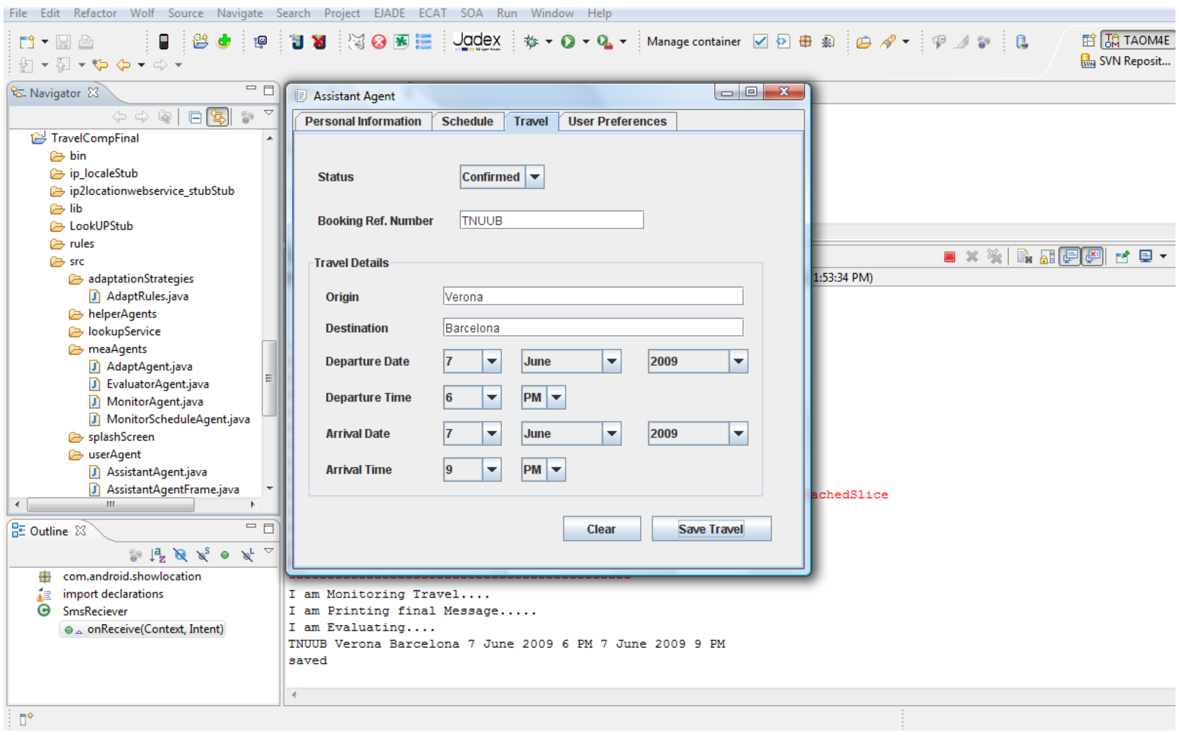
Finally SAS selects a candidate solution e.g. "Place Call" by evaluating the relegation relation, specified earlier in the adaptive RML model and detailed in adaptive requirements e.g. S4 $\succ$ S3 $\succ$ S2. The new candidate solution **S4**: *Context: (Null), Resource: (Contact List), Task: (Place Call), Domain Assumption: (All Secretaries have landline Phone).*

Although there might be situations where there is no possible candidate solution is available by searching the requirements model. In this case, they SAS can involve end-user to gain feedback or acquire new knowledge pertaining to the specific requirements problem. Once acquired new information, the SAS need to refine its existing specification per se. In parallel, it tries to look for a possible solution (e.g. searching for a relevant service). This leads SAS to move from one requirements problem to another, while monitoring the end-user context and resource availability.

6.6 In Practice

In this section, we describe the prototype application that is developed. To guide the process of implementation, we mainly refer to the CARE's conceptual architecture that guide the realization of this proof of concept application. To fulfill this aim, we have used the existing framework JADE[2] as a can-

[2]http://jade.tilab.com/



TravelCompFinal
bin
ip_localeStub
ip2locationwebservice_stubStub
lib
LookUPStub
rules
src
adaptationStrategies
AdaptRules.java
helperAgents
lookupService
meaAgents
AdaptAgent.java
EvaluatorAgent.java
MonitorAgent.java
MonitorScheduleAgent.java
splashScreen
userAgent
AssistantAgent.java
AssistantAgentFrame.java

com.android.showlocation
import declarations
SmsReciever
onReceive(Context, Intent)

Status	Confirmed
Booking Ref. Number	TNUUB

Travel Details

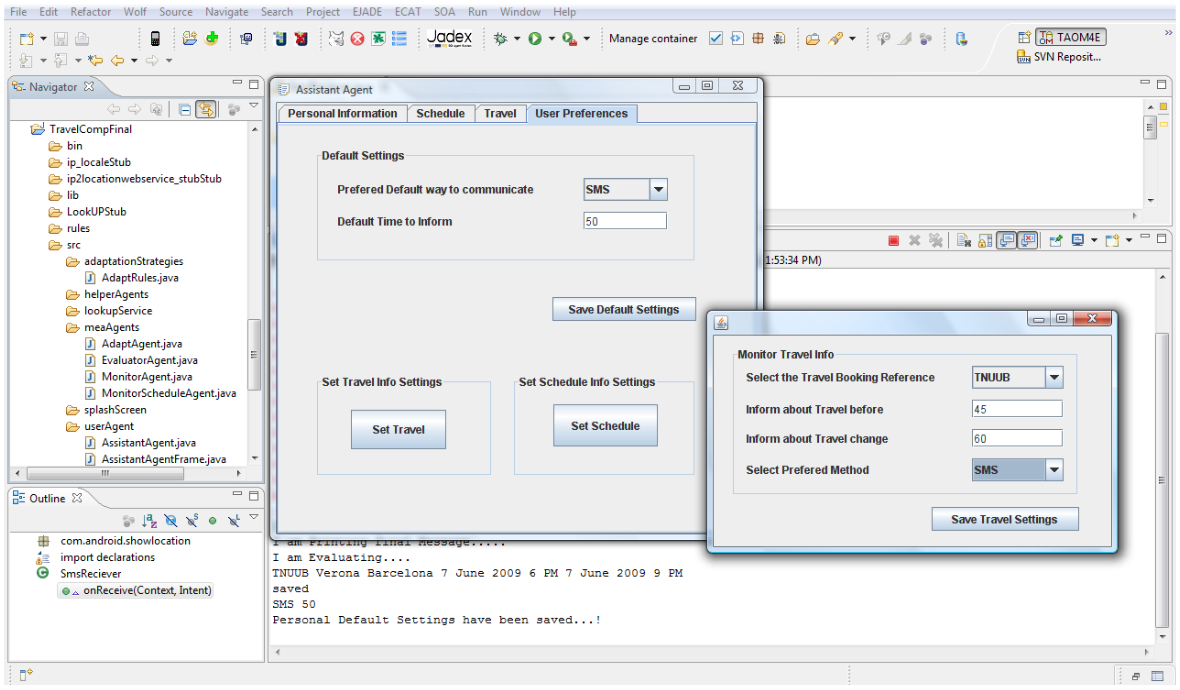
Origin	Verona		
Destination	Barcelona		
Departure Date	7	June	2009
Departure Time	6	PM	
Arrival Date	7	June	2009
Arrival Time	9	PM	

```
I am Monitoring Travel....
I am Printing final Message.....
I am Evaluating....
TNUUB Verona Barcelona 7 June 2009 6 PM 7 June 2009 9 PM
saved
```

Figure 6.5: “iComp” GUI Screen

didate development platform and existing web services (e.g. Google Map, Weather.com) and also our own web services mainly developed in Java to simulate the flight booking service. The agents that are developed preforms CARE activities.

At the start of the application, the $UIAgent$ displays the graphical interface of “iComp” as shown in the Fig.6.5. End-User can customize the application by setting meeting agenda and travel schedules by entering information pertaining to travels (e.g. travel itinerary). Preferences related to the meeting agenda and travel schedules can be set, which provides input to the $Monitor - Agent$ to observe them (e.g. services that are required to be invoked or acquire live feed etc.). For instance, in Fig. 6.6, preferences related to travel can be set i.e. the method for communication and time to notify. $Monitor - Agent$ take these setting and start monitoring the service, in this case it is travel flight information service.



Default Settings

Prefered Default way to communicate: SMS

Default Time to Inform: 50

Monitor Travel Info

Select the Travel Booking Reference: TNUUB

Inform about Travel before: 45

Inform about Travel change: 60

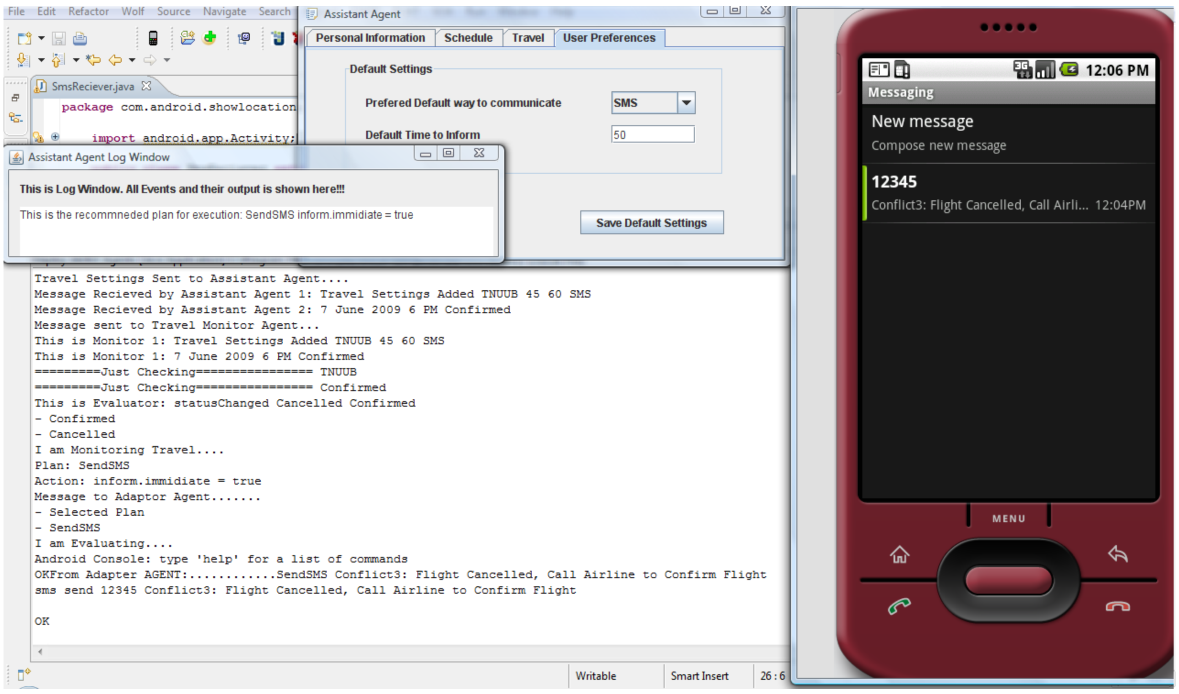
Select Prefered Method: SMS

```
I am Printing Final Message.....
I am Evaluating....
TNUUB Verona Barcelona 7 June 2009 6 PM 7 June 2009 9 PM
saved
SMS 50
Personal Default Settings have been saved...!
```

Figure 6.6: “iComp” Preference Screen

A thin client is also developed using Android OS[3]. This concept application is a service-based application which relies on the external services to meet the end-user expectations. For instance, in the given scenario, end-user set the preference method for communication as SMS, and to be notified quickly. The details about the travel itinerary provided by the end-user (e.g. Itinerary Number) are matched with the relevant web service, which we developed and was running on a local server. The application match the provided “Flight Status” with the actual “Live Status”. If the status is not matched (see AR 3), the $Monitor - Agent$ detect the event and identify violation and notify the $Evaluator - Agent$, it lookup AR 3, and evaluation condition returns false, which means the “Flight Status == Canceled”. It looks for the preference of the end-user and get the context details (e.g. location). It then issues a execute message to the $Adapt - Agent$ to perform the required action i.e. Send SMS. The final notification is received by the end-user on her device i.e. smartphone.

[3]http://code.google.com/android/



```
package com.android.showlocation
    import android.app.Activity;
```

Prefered Default way to communicate: SMS
Default Time to Inform: 50

This is Log Window. All Events and their output is shown here!!!

This is the recommneded plan for execution: SendSMS inform.immidiate = true

```
Travel Settings Sent to Assistant Agent....
Message Recieved by Assistant Agent 1: Travel Settings Added TNUUB 45 60 SMS
Message Recieved by Assistant Agent 2: 7 June 2009 6 PM Confirmed
Message sent to Travel Monitor Agent...
This is Monitor 1: Travel Settings Added TNUUB 45 60 SMS
This is Monitor 1: 7 June 2009 6 PM Confirmed
=========Just Checking=============== TNUUB
=========Just Checking=============== Confirmed
This is Evaluator: statusChanged Cancelled Confirmed
- Confirmed
- Cancelled
I am Monitoring Travel....
Plan: SendSMS
Action: inform.immidiate = true
Message to Adaptor Agent.......
- Selected Plan
- SendSMS
I am Evaluating....
Android Console: type 'help' for a list of commands
OKFrom Adapter AGENT:............SendSMS Conflict3: Flight Cancelled, Call Airline to Confirm Flight
sms send 12345 Conflict3: Flight Cancelled, Call Airline to Confirm Flight

OK
```

12345
Conflict3: Flight Cancelled, Call Airli... 12:04PM

Figure 6.7: “iComp” Adaptation in the Scenario

This implementation provided us initial improvements that we have identified during the implementation of this application. However, in the current implementation, the run-time refinement of requirements (i.e. update requirements specification activity in CARE) is not presented. The current prototype does not support these actions. Further steps in this research are aimed at providing a more refined version.

6.7 Final Remarks

So far, we have introduced a working prototype application, which instantiates the CARE framework. In fact, the prototype is limited in functionality and reasoning capabilities. It supports (adaptation Types 1-3), however, more interesting scenarios for adaptivity can provide better assessment. As a qualitative evaluation of this prototype, current version of the prototype can be easily evolved by adding more capabilities and agents e.g. new web services. In each iteration new functionality or mechanism can be introduced, although implementation effort is required and cannot be ignored. With respect to performance, the current version performs well. Actually it depends upon many factors e.g. complexity of the algorithm and number of constraints and requirements to satisfy in a given domain of discourse. Therefore, further evaluation along scalability and performance is needed in the future.

While developing this application, we have identified several directions for improvement, such as (i) continuous refinement of requirements specification; (ii) specification of end-user preferences e.g. priorities and ratings about requirements and environment that can change over-time; (iii) on-line reasoning for contextual changes e.g. user's situational information which includes location-awareness and relevant information about user's surrounding to provide meaningful support for decision making at run-time; and (iv) efficient methods for service discovery. We aim to consolidate the CARE framework with them.

Expanding on this experience, as an ongoing future work we are focusing on (i) defining a method for continuous refinement of requirements. We are interested in minimal changes to the specification at run-time, mainly, to know what to change and what are the consequences of this change. To this aim, we are defining an ontology of actions that helps SAS to evolve and manage requirements by itself by involving the end-user. In fact, SAS at run-time exploits it's requirements database – a machine readable form of requirements model and reasoning rules that guide the evolution process.

Chapter 7

Evaluation

7.1 Overview

In this chapter we present the evaluation of our design-time framework. First we check the scalability and performance of the prototype tool presented in Chapter 5. The results shows the applicability of the tool with respect to models of increasing size. Second, we present our modeling experience on a travel case study. We report our qualitative assessment to show the differences between the proposed ARML and existing state of the art goal modeling languages. Third, we present in detail the evaluation from an empirical survey on ARML to collect initial feedback on the effectiveness of ARML (modeling, reasoning and visual notation). These initial evaluation results discussed below provides promising results. However, improvements are critical, which are the focus of further steps of this research.

The rest of the chapter is organized as follows. In Section 7.2, we report the results obtained from performance evaluation of the design-time tool. In Section 7.3, we present a qualitative assessment on modeling requirements for SAS using state-of-the-art goal modeling languages. In Section 7.4, we report results from an empirical survey performed on the effectiveness of the proposed ARML. In Section 7.5, we summarize this chapter.

7.2 Evaluation of Design-Time Tool

To evaluate the feasibility of the proof of concept tool, we have performed a set of experiments to provide early assessment on the scalability and performance.

7.2.1 Environment of the experiments

The following experiments were conducted in a Intel Core 2 DUO 2.40 Ghz with 4 GB of RAM, running windows Vista 32 bits.

7.2.2 The parameters of the study

We have constructed 500 variants of a Goal Model. Starting from an initial goal model of just 10 elements (goals, softgoals, task, resource, context, preferences and quality constraint), we have built a set of 500 variants adding on each variant 10 more elements randomly (e.g. model250 has 2500 elements). All generated goal models are stored as XMI files (see Appendix B//to be added).

7.2.3 Experiment 1: Time to load the variants in the tool

These variants of the model are implemented in eclipse. As a first step they are imported into the tool, so the analyst can annotate the elements and generate the specifications. The time it takes to import a goal model depends with respect to the number of elements this model has. The time to load a variant includes parsing time of the xmi file i.e. retrieve the data and store it into the database of the tool in a convenient structure. The graph shown in the Fig.7.1 represents the load time of the variants.

In order to better analyze the results over these data points we have applied curve fitting techniques to infer which the function that better fits the given results is. The most accurate and convenient function is: $0.0000008x^2 + 0.0016x + 0.473$.

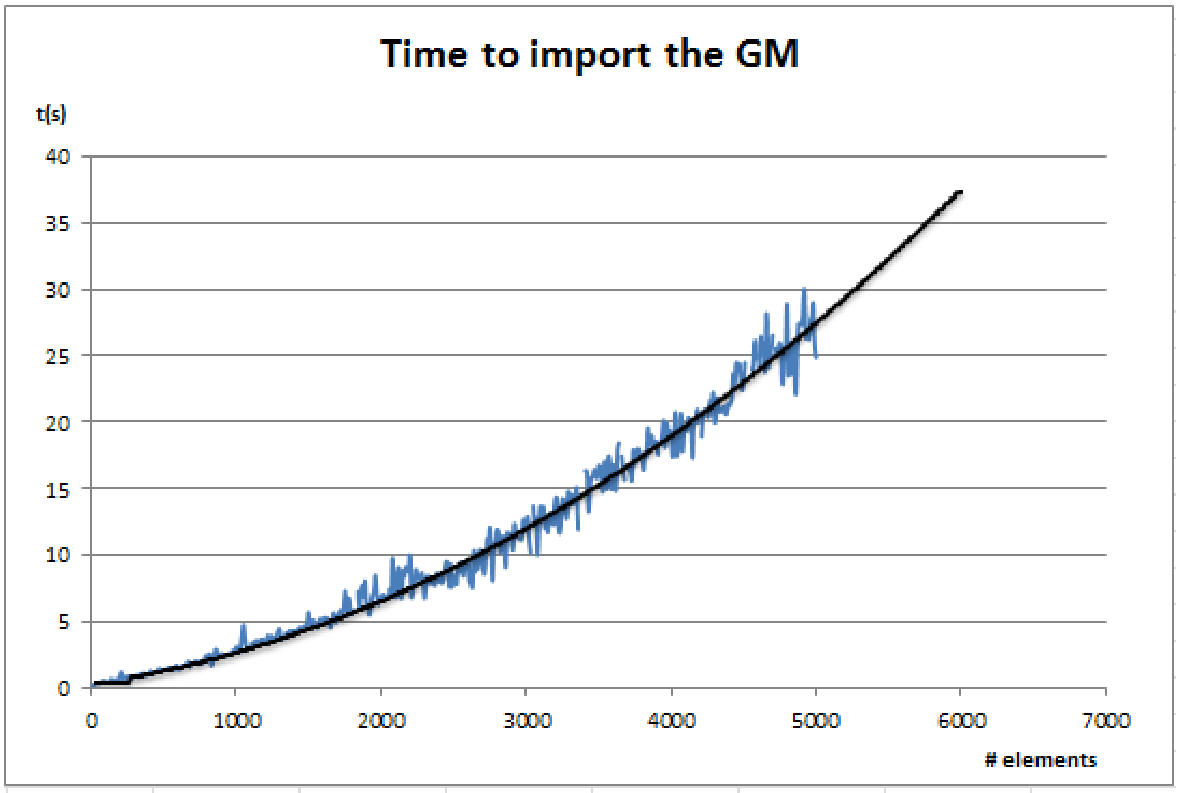


Figure 7.1: Time to load variants in the tool

As we can see in the graph shown in Fig.7.1, although being a quadratic function, the term in x^2 is significantly small compared to the term in x. We can observe that even for a goal model of 5000 elements, this process takes less than 30 seconds to import the model to the tool. A medium-size goal model of 200 elements takes around 1 second.

7.2.4 Experiment 2: Time to generate the monitor specification

Once the goal models are imported to the tool and the analyst has annotated all the elements with the required information, the analyst can automatically generate the monitoring specification documents. As stated previously, each task in the model is instrumented by a service description. A related monitoring specification is generated automatically for each annotated task (e.g. a goal model with 15 tasks instrumented by services, will generate 15 monitoring specifications). The time to generate the monitoring specification depends on the number of model elements. Mainly, parsing and transformation has been taken into account.The graph shown in Fig.7.2 shows the time to generate the monitoring specification.

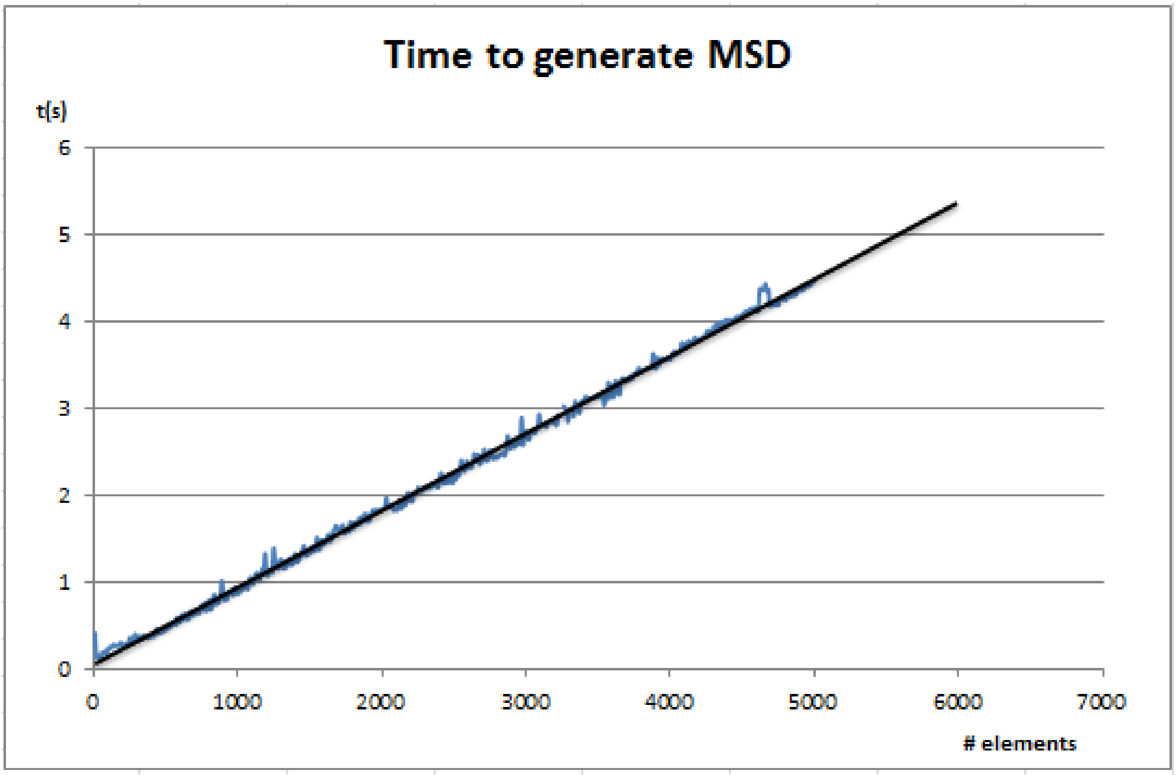


Figure 7.2: Time to generate monitoring specifications

In order to better analyze the results over the data points gathered in the graph, we have applied curve fitting techniques to infer which the function that better fits the given results is. The most accurate and convenient function is: $0.0009x - 0.0432$

As we can see, the results in this stage of the process are even more promising. For a goal model of 5000 elements, this process takes less than 5 seconds to generate the different set of monitoring specifications. A medium-size goal model of 200 elements takes around 0,3 seconds.

7.2.5 Experiment 3: Time to configure the monitors

The generated monitoring specifications are the input to configure SALMon to start monitoring. The time it takes to configure SALMon from a set of monitoring specifications derive from a goal model variant is represented in the graph as shown in Fig.7.3. The graph shows the performance in seconds with respect to the size of the initial goal model.

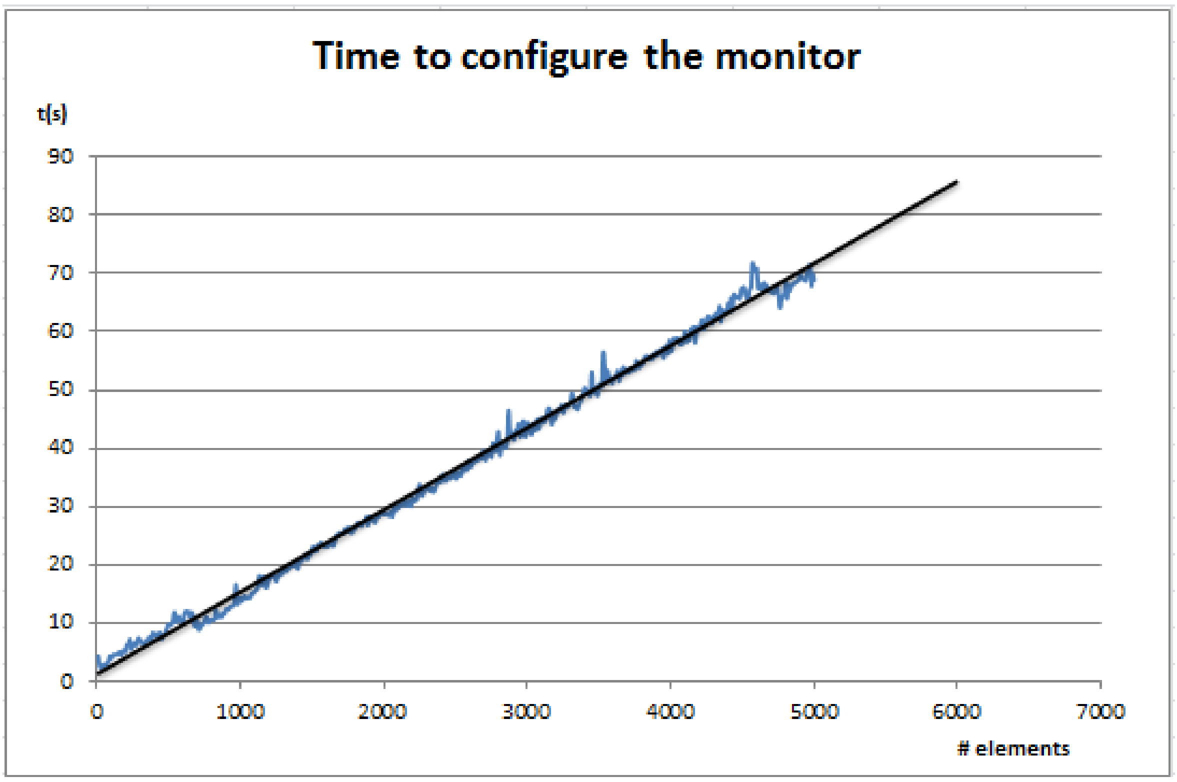


Figure 7.3: Time to configure monitors

In order to better analyze the results over these data points we have applied curve fitting techniques to infer which the function that better fits the given results is. The most accurate and convenient function is: $0.0141x + 1.0994$.

As we can see, configure the monitors is the most time-consuming step of the process, although being a linear function, the term in x is significantly bigger than the main terms in the other functions. Nevertheless, we can see that for a goal model of 5000 elements, this process takes around 1 minute and 10 seconds. A medium-size goal model of 200 elements takes around 5 seconds, which demonstrates the feasibility of our approach, with respect to the performance and scalability.

7.3 Qualitative Evaluation of Adaptive RML

Requirements modeling in the early steps of RE aims to establish in a rough way the overall purpose of the system, how well it should achieve that purpose, how it would achieve it, and what resources it needs to mobilize in order to do so. This requires the elicitation of requirements from stakeholders, their interpretation, restructuring, refinement, classification as instances of concepts, the identification of relations between them, and their representation in models.

Most of the ongoing works on modeling requirements for SAS falls within goal-oriented modeling, since the goal abstraction provides an intuitive way to capture high level criteria the system should fulfill in order to meet end-users expectations. In this section, we focus on a simple scenario for a travel booking application. We model it with the *i** and *Tropos*[1] requirements modeling language and identify key elements that are missing in these languages to modeling requirements for SAS. Here we describe an excerpt of a scenario from a travel companion case study introduced earlier, in which self-adaptive properties of the system are illustrated.

Scenario:*The iComp application is a self-adaptive system that aids business travelers while on the move. It supports them in booking their travels, making payment and receiving timely updated information about their booking confirmation (e.g., confirmed, canceled, in progress). The booking confirmation messages must be sent to the user (customer) via Email or SMS (i.e. most preferred method) instantly (i.e. in less than an hour or maximum less than*

[1]Modeling tool for Tropos methodology is used i.e. TAOM4e.

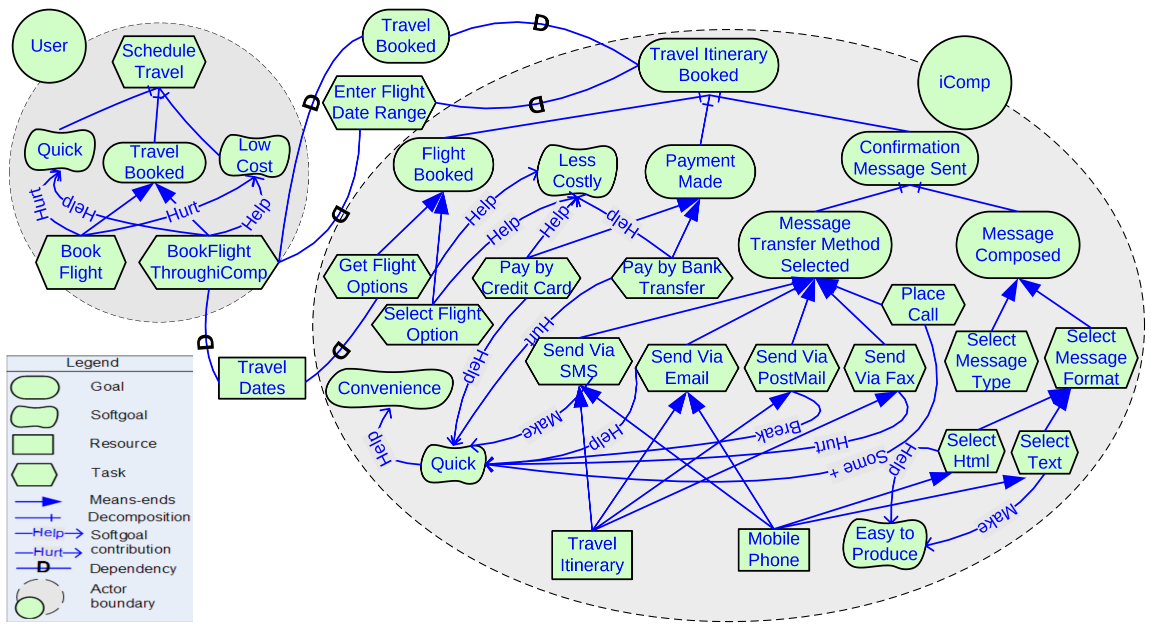


Figure 7.4: Modeling using iStar Concepts and Relations only.

1 day) on their device (i.e. laptop or mobile) and depending on their context (i.e. home or market). Payment must be ensured before iComp sends the message (i.e. composing the message) by selecting a suitable message format (i.e, size, scaling, format) to adapt to the device from which they will be read. Finally, in case there are some problems (i.e. user is not accessible) and the message cannot be delivered to the user then iComp must send the message to an alternative recipient (e.g., the customer's secretary).

7.3.1 Requirements Modeling with i*

Figure 7.4 shows *i** strategic dependency and rationales models for the travel booking scenario. In the main scenario, the user and the system are represented by circles, whereby the content of the dashed ovals (strategic rationales) represents their goals, tasks, and resources. We can see in this model that what leads the user to chose the iComp for travel booking results from the analysis of the root task of `Schedule Travel`, which is decomposed into the goal of `Travel Booked` and the softgoals of `Low Cost` and `Quick`. These softgoals are negatively influenced (shown with contribution links) by the subtask of `BookFlight`. The task `Book Flight through iComp`, however, partially satisfices the `Low Cost` and `Quick` softgoals. This task in turn depends on the `iComp` Actor, since the associated goal `Travel Itinerary Booked` has been assigned to the system. The strategic rationales model of iComp reveals a decomposition of goal `Travel Itinerary Booked` into three main goals `Flight Booked`, `Payment Made` and `Confirmation Message Sent`. For example, we can now reason about `Confirmation Message Sent`, which is decomposed into two goals i.e. goals: `Message Transfer Method Selected` and `Message Composed`. Along their subsequent means of accomplishing tasks, and assess their contributions towards softgoals `Quick` and `Easy to produce`, which helps in ranking a particular solution. For example, tasks: `Send via SMS` and `Send via Email` with means to use resources: `Travel Itinerary` and `Mobile Phone` contributes fully and partially to the softgoal `Quick`, which in turn satisfices softgoal `Convenience`. The aim of this analysis is to identify one particular solution that satisfies the high level goals and optimally satisfies the softgoals.

Modeling iComp in *i** lead us to identify some limitations of the language when used for SAS. *i** does not provide concepts for the modeling of alternative solutions to the requirements problem, which are feasible in different contexts. For instance, in context (e.g. Home) the candidate solution should be *Send Message via Email* and in another context, e.g. Market, *Send message via SMS* should be more appropriate in case no 3G or no smartphone is available for the user, and so on. That is we were not able to model the fact that the context of the user may change as well as resources availability, and ultimately to capture monitoring conditions and evaluation criteria that should characterize the dynamically adaptive behavior of the system. Also, in *i** it is not possible to model quality constraints, such as *send the message within one hour after the payment*, and domain assumptions that need to be made explicit during the analysis as they contribute to the definition of the requirement problem, such as *standard Credit Card Options must be Displayed*.

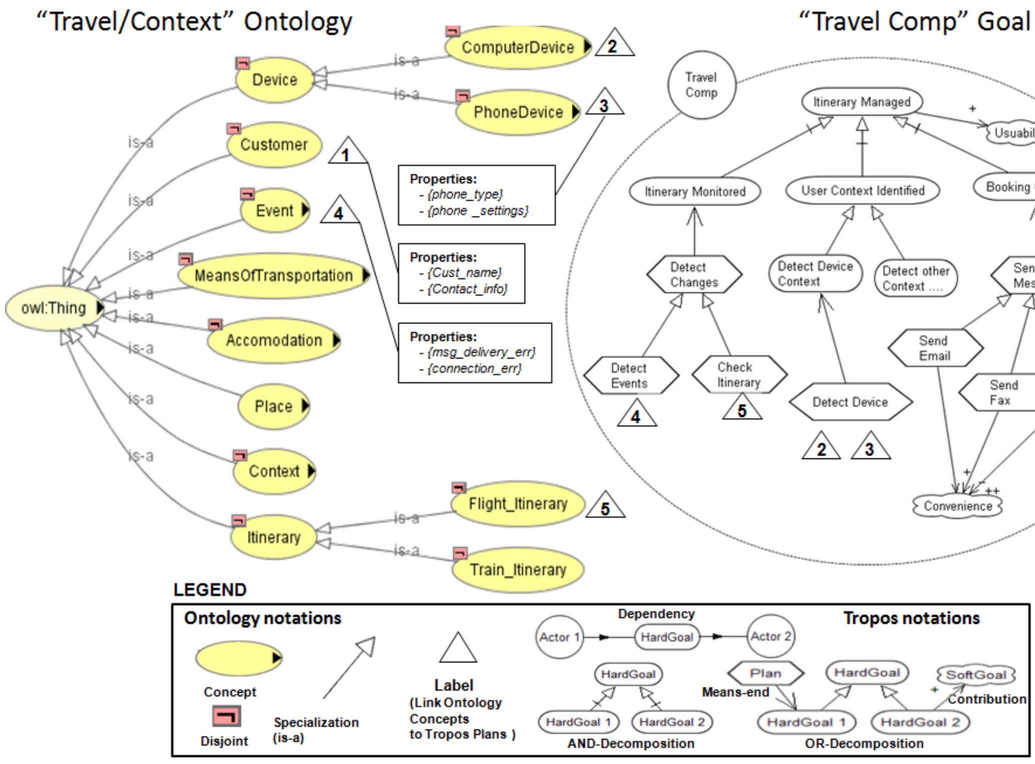


Figure 7.5: Modeling using Tropos Concepts and Relations (R) and Annotated Goal model with domain properties modeled using Domain Ontology (L)

7.3.2 Requirements Modeling with Tropos

Following the *Tropos* – software development methodology [PPSM07a], we move from a goal-oriented model of the domain's stakeholders (including end-users of the system-to-be) to the specification of the system requirements by delegating end-user goals to the system. Hard-goals represent the rationale behind functional requirements (depicted as ovals in fig. 7.5, right side) and soft-goal (cloudy shape) represent non-functional aspects of the system-to-be.

Elaborating the example, we analyze the a variant of the requirements in the above scenario in terms of hard-goals and soft-goals thus addressing the (***Why***) question along with the (***What***).

The goal Booking Confirmed has been delegated by the end-user to the system, represented by the actor "iComp", and it is analyzed as a sub-goal of a more general goal of the system, namely Itinerary Managed, through AND-decomposition. Following the details of the Booking Confirmed analysis, we may found that this goal is achieved through a plan $\langle SendMessage\rangle$, depicted in terms of means-end relation in fig. 7.5. This plan specifies the action of sending a confirmation message to the user device as soon the booking is processed. This plan can be realized by alternative behaviors, namely $\langle SendEmail\rangle$, $\langle SendSMS\rangle$ and $\langle SendFax\rangle$, represented by the plan OR-decomposition. These three alternative plans address the variability in the operational context, which affect (***How***) the goal of confirming booking can be achieved by the system.

Criteria for selecting the right alternative are represented by contribution relationships (arrows labeled with "+" or "-") to soft-goals. For instance, the selection of the plan $\langle SendSMS\rangle$ should be performed when the user's device is a smartphone/PDA, as it contributes (++) positively to the users' soft-goal {Convenience} as compared to $\langle SendFax\rangle$, which contributes (–) negatively to it.

Information needed to drive the plan selection are collected at run-time by the system following monitoring requirements expressed in the analysis of the goals e.g. Itinerary Monitored and User Context Identified. For instance the plan $\langle DetectEvent\rangle$ that is part of the means-end analysis of the Itinerary Monitored goal contains specification of how to detect unforeseen events, as message delivery error or connection lost, and provide criteria for the system to switch to a behavior in which the end-user's contact info are used to inform colleagues or friends about the confirmed booking. In Fig.7.6, we present a plan specification that depicts the adaptive requirements for the plan $\langle SendMessage\rangle$.

So far, exploiting goal modeling reveals the high-variability that is needed in the adaptive requirements for SAS. However, to better specify these requirements information pertaining to the domain assumptions and facts must be made explicit. This information provides a better way to the analyze variability. Recent research on the use of ontology techniques for requirements engineering has proven their effectiveness. For instance, an integration of goal models and ontologies has been recently proposed by Shibaoka et al. in [SKS07]. This approach employs goal-oriented modeling complemented by knowledge representation technique (ontologies) to solve difficulties in refining goal models, thus facilitating the analyst to elicit requirements. This approach makes an important step, thus provides a starting point for our investigation, where we use goal-models and ontologies to elicit requirements for self-adaptive software with the further objective to allow the software system, at run-time, to reason on them. Returning to our analysis, we introduce another step for modeling adaptive requirements in *Tropos*.

Domain Modeling:

Representing the knowledge about the domain using ontology provides a way to elicit domain assumptions. We model application domain and operational context properties in the ontology as shown on the left side of the fig. 7.5. For example, in the ontology we identify the common concepts used in travel domain such as Itinerary, Accommodation, Means of transportation (by plane or by train etc.). After identifying the concepts, we define relationships between these concepts. It helps in understanding the said domain assumptions that remains true in a domain of interest. To express these assumptions we use concept properties and inference rules, which represents them. For example, one possible domain assumption can be: *Customer's itinerary is valid not before and not after the departure date*. Moreover, concepts that describe the operational context are represented in our ontology, such as Context, Device, Event having a property *(Message deliver error)* and Customer having a concept property such as *(Contact Info)* etc.

```
//$Plan\_Model(\langle SendMessage\rangle)$ to accomplish Goal (BookingConfirmed)
begin procedure $Plan\_Model(\langle SendMessage\rangle)$
do triggerGoals [UserContextIdentified,ItineraryManaged];
   begin
      for $Goal$ [UserContextIdentified]
      do executePlan $\langle DetectDevice\rangle$; //@param: phone_type, phone_setting
      return; //@result: device
   end;
   begin
      for $Goal$ [ItineraryManaged]
      do executePlan $\langle DetectChanges\rangle$; //@param: msg_delivery_err, conn_err
      return;//@result: eventMessage
   end;
$decision = decision\_on\_AltPlans(device, eventMessage)$;
case $decision$:
      - Select case:$\langle SendSMS\rangle$; //if device = PDA, eventMessage = null
      - Select case:$\langle SendEmail\rangle$; //if device = Laptop, eventMessage = null
      - default case: $\langle SendFax\rangle$; //if device = null, eventMessage = null
$ifnot$ [$decision$]
   $then$ lookupContact; //@param: cust_name, contact_info
$alt\_decision = decision\_on\_AltPlans(cust\_name, contact\_info)$;
case $alt\_decision$:
      - Select case: $\langle SendEmail\rangle$; //contact_info
      - default case: $\langle SendFax\rangle$; //contact_info
end procedure;
```

Figure 7.6: Example of Plan description

The analyst after adopting a domain ontology or building it explicitly needs to link it with the goal models. This leads to the further step of annotating the goal model with the ontology concepts and their respective properties.

Linking Ontology to Goal Model:

The links between domain concepts and their properties (represented in the ontology) to the plans and goals (represented in the goal model) are shown in the Fig. 7.5 (see label 1 to 5). For instance, the concept Phone Device in the ontology is linked with the plan Detect Device in the goal model (with the label 2 and 3). These labels associate the properties *phone_type* (smartphone, PDA etc.) and *phone_setting* (e.g. Operating system, memory etc.) with the plan Detect Device to provide an additional information to the plan's specification. Similarly, the concept Event is associated with plan Detect Events, which provides the plan to detect events such as *msg_delivery_err* as shown in the fig. 7.5 as label 4.

This helps analyst in detailing software behaviors that encompass not only the actions to satisfy (functional) goals, but also monitoring and evaluation actions. Fig. 7.6 describes the plan $\langle SendMessage \rangle$, which provides the means to achieve the (functional) goal Booking Confirmed. This plan requires to trigger monitoring goals that are pursued in parallel, providing data for the evaluation actions that drive the selection of the right alternative e.g.$\langle SendSMS \rangle$ to send confirmation message to the users' device (assuming the device is identified as smartphone or PDA).

Based on the experience we earned in modeling adaptive requirements using existing goal modeling approaches, we see that similar efforts has been made to capture requirements for SAS by extending *i*/Tropos* [MPP08, ADG09]. The main idea behind these extensions is to annotate goal models. For instance, in [MPP08] goal achievements conditions and environment modeling (using UML class diagrams) is used to annotate the *i*/Tropos* goal model, and transform them for use with an implementation architecture (e.g. BDI). Similarly in [ADG09] location abstraction is used to formalize context and annotating the variation point (e.g. AND/OR decomposition) with in a goal model. This approach provides a systematic design-time approach to build context models based on locations concepts (e.g. using UML class diagrams). Common to both approaches is the use of UML notation to formalize the concept of environment and context hierarchies. Both approaches are focused on finding a single best solution in case of adaptation.

In summary, both of these approaches and our modeling efforts using *i** and *Tropos* are limited to show how the system can move across contexts (with changing domain assumptions, resource availability) by altering the requirements problem with respect to the variety of candidate solutions. We now move to model the same scenario in ARML to reveal its potential edge over the existing approaches.

7.3.3 Requirements Modeling with ARML

Differently from the previously mentioned extensions of goal-oriented modeling languages for SAS, we rest on Techne [JBEM10b], an abstract modeling language for early requirements, which adopts the core ontology for RE [JMF08]. This core ontology extends the goal-oriented perspective allowing to model optional requirements, preferences, and to treat fully non-functional requirements in terms of approximations and quality constraints. The basic elements of Techne models are requirements, modeled as propositions that are labeled as domain assumptions, goals, quality constraints, or tasks. A requirement can be mandatory or optional. Connections between model elements are used to represent how the satisfaction of an element may impact the satisfaction of the other, through inference and conflict. Preferences are used to compare requirements in terms of desirability. Performing the analysis of a requirements problem specified in Techne, results in finding candidate solutions in terms of tasks and quality constraints that together satisfy all mandatory goals and cover, as much as possible, optional ones.

The proposed modeling language for SAS, called ARML, builds on Techne by adding two new concepts, namely, context and resource, and two relations, i.e. relegation and influence. ARML has its own visual notation. In the rest of this section we illustrate an ARML model of iComp with the aim to provide a preliminary qualitative evidence about its support in overcoming the limits mentioned above in modeling requirements for SAS. A detailed account of ARML will be given in the ensuing sections.

Fig.7.7, shows a requirements model for iComp in ARML. Its root level goal `Travel Itinerary Booked` is modeled as a mandatory node (*modeled as **M** node, a unary relationship*). It is decomposed via an inference relation into the other mandatory goals: `Flight Booked`, `Payment Made` and `Confirmation Message Sent` (*modeled as black **I** node with a arrow, a binary relation*), to

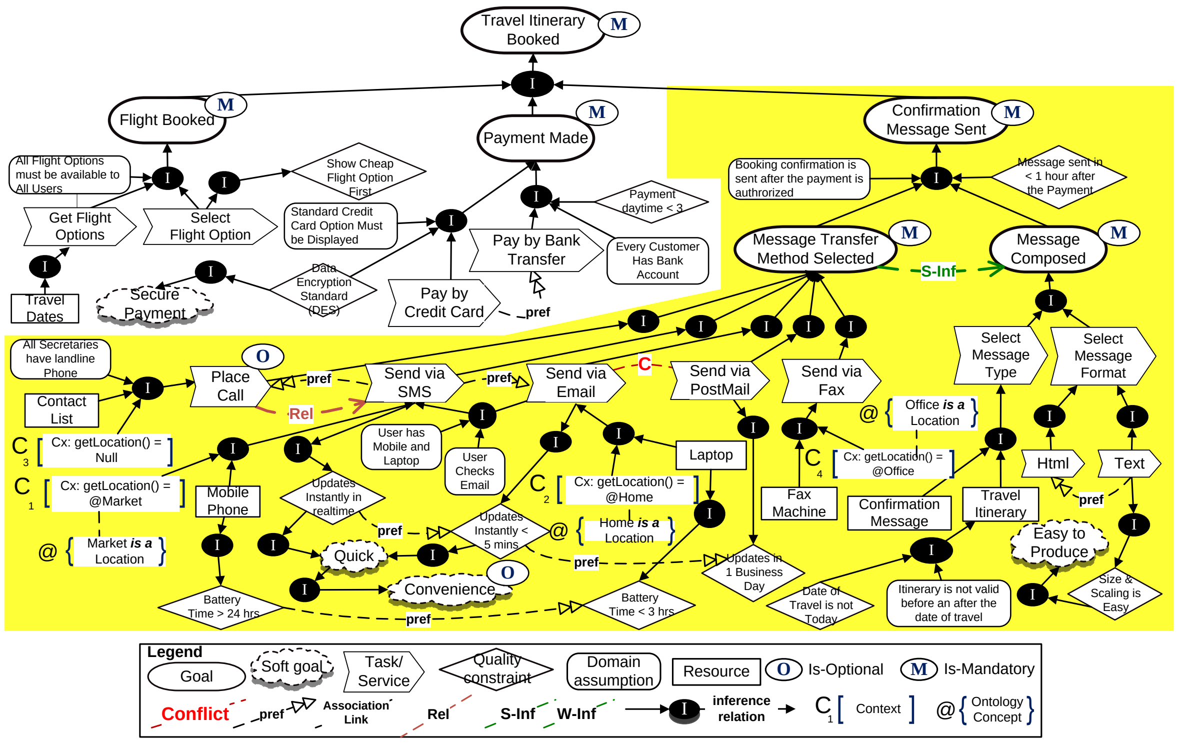


Figure 7.7: Modeling using ARML Concepts and Relations.

represent the fact that it will be satisfied through the joint satisfaction of these three goals.

Let's focus on the goal: `Confirmation Message Sent` (i.e. the shaded part of the model), which is decomposed into two goals: `Message Transfer Method Selected` and `Message Composed` via inference relation. We can add here information that were missing in *i** model, i.e. the domain assumption `Booking Confirmation is sent after the payment is assured` (*modeled as rounded rectangle*) and the quality constraint `Message sent in < 1 hour after the payment` (*modeled as diamond shape*) connecting them through the same inference node.

An influence relation is added among the two decomposed goals: `Message Transfer Method Selected` and `Message Composed` (*modeled as dotted green line with arrowhead*) to account for the prevailing context conditions and resource availability that influences the achievement of goal: `Message Composed`.

For example, if the context conditions support to choose Email as a candidate transfer method, the ways to satisfy goal:`Message Composed` is by selection a correct format that is either text or html.

The analysis of the `Message Transfer Method Selected` proceeds by linking via inference nodes task-rooted subgraphs, which defines candidate solutions. Besides tasks e.g. `Send via SMS` (*modeled as hollow motion arrow*), each candidate solution includes domain assumptions e.g. `User has mobile and laptop`, context e.g. `Market`, `Home` (*labeled as **C** with its number, associated to @ symbol*[2]), and resources e.g. `Mobile Phone` (*modeled as a rectangle*). Preferences (*dotted line with doubled empty arrow heads*) are used to compare requirements in candidate solutions, and thereby compare candidate solutions; e.g., `Send via SMS` is preferred over `Send via Email`. Requirements can be in conflict (e.g. `Send via Email` is in conflict with `Send via PostMail` (*dotted line with C in the middle with red color*). Conflict is shown due to the difference in quality constraint (e.g. Email updates in <5 mins, whereas post mail updates in 1 business day). Notice that it was not possible to model these information with *i**.

Optional solutions, in case of problems (e.g. user is not accessible, as mentioned in the scenario) can be identified via a relegation relation (*modeled as dotted light red line with arrowhead* between two possible candidate solutions). For instance, `Place Call` relegates `Send via SMS`. This allows to take into account the situation in which a user's context changed resulting in being not accessible (e.g. Context: getLocation()= Null), and to describe as preferred the solution to make the user able to access the resources `Confirmation Message` and `Ticket Itinerary`, via contacting her secretary. The `Place Call` task is inferred via a domain assumption (e.g. `All secretaries has landline phone`) and a resource (e.g. `contact list`) and the context (e.g. Context: getLocation()= Null).

[2] @ labels a concept defined in domain ontology e.g. travel.

7.3.4 Summary

Gain in expressiveness of ARML with respect to *i** and *Tropos* models are summarized below:

- we can model information about context, resources and domain assumptions that need to be monitored by the SAS in order to enable adaptation;
- softgoals evaluation in *i** and in *Tropos* is subjective and provides no clear evidence to rank a solution. In ARML, candidate solutions can be ranked and evaluated via quality constraints over measurements that may be collected by the SAS;
- candidate solutions can be associated with contexts.
- candidate solutions can be associated with resources that are needed.

On the visual modeling notation, a further effort is needed to fit with the proposed recommendation set out in [MHM09], for improving usability and communicative effectiveness of visual notations in RE modeling. To this aim, we have conducted an initial survey involving the subjects. Results of this empirical survey are summarized in the next section.

7.4 Empirical Survey on ARML

7.4.1 Goal of the Study and Research Questions

In order to provide first assessment on the effectiveness of ARML (modeling, reasoning, and visual notations), we performed a (pilot) survey to gain confidence on the proposed language. In the study, we ask some expert requirements engineers to exercise with ARML and then fill some questionnaires, (using a Likert scale, see below), to express their opinion about several aspects of ARML. The aim is to gain an early feedback with a clear goal of acquiring the expert perceived judgment.

In particular, we are interested to answer the following research questions:

- **RQ1:** Are the concepts and relations proposed in ARML effective to model requirements for SAS at design-time?
- **RQ2:** Are the concepts and relations proposed in ARML useful for reasoning by the SAS itself at run-time?
- **RQ3:** Are the visual modeling notations in ARML adequate for modeling requirements for SAS?

These questions let us investigate the *effectiveness* and *usefulness* of the concepts and relations along with the *adequacy* of the visual notations proposed in ARML to model requirements for SAS.

7.4.2 Hypothesis

Based on the above stated research questions, we now describe our null hypothesis that we use to evaluate our candidate method.

RQ1 is related to the *effectiveness* of the modeling notations proposed in ARML. Below we state the following null hypothesis and alternative hypothesis for **RQ1**:

- H_1^0**:** Concepts and relations proposed in ARML are not effective to model requirements of SAS by the analyst at design-time.
- H_1^{alt}**:** Concepts and relations proposed in ARML are effective to model requirements of SAS by the analyst at design-time.

RQ2 is related to the *usefulness* of the modeling notations proposed in ARML to perform reasoning over them at run-time by the SAS itself. Below we state the following null hypothesis and alternative hypothesis for **RQ2**:

- H_2^0**:** Concepts and relations proposed in ARML are not useful to reason on them by the SAS itself at run-time.
- H_2^{alt}**:** Concepts and relations proposed in ARML are useful to reason on them by the SAS itself at run-time.

RQ3: is related to the *adequacy* of the visual modeling notations that are proposed in ARML for modeling requirements for SAS. Below we state the following null hypothesis and alternative hypothesis for **RQ3**:

- H_3^0**:** Visual Modeling notations representing the concepts and relations in ARML are not adequate to model requirements of SAS.
- H_3^{alt}**:** Visual Modeling notations representing the concepts and relations in ARML are adequate to model requirements of SAS.

We formulate each hypothesis with a clear direction (i.e, they are one-tailed) since we assume that the new concepts and relations proposed by ARML provide better support for dynamic changes to be taken into account while modeling requirements for SAS, with respect to the more traditional methods (e.g., standard *i** or *Tropos*).

Hypothesis testing

Each research question is investigated by means of questionnaires having the aim of collecting the expert judgment and feedback. In such questionnaires, the subjects are asked to express their opinion and judgment by means of the Likert scale [1-5], where 1 means "Strongly agree", 5 means "Strongly disagree" while 3 represents the "Undecided" (the "medium" value between positive and negative opinion/judgment). Hence, for each question X, by using the Wilcoxon test we check if the score obtained by the questionnaire is <3, this means that X is adequate (positive evaluation is given by the subject). For instance,

- Question: Is the evaluation of 'X' perceived as adequate by subjects?
- H^0: 'X' is negatively perceived (score >3)
- H^{alt}: 'X' is positively perceived (score ≤ 3)

In other terms, for the interpretation of the data acquired from the survey, we adopt the following tests and evaluation measures:

- (A) Regarding the analysis of survey questionnaires, we evaluate each question by verifying that the answers is either "**Strongly agree**" (i.e., 1 in our Likert scale) or "**Agree**" (i.e., 2 in our Likert scale). We test medians of the distributions, using a the Wilcoxon test for the null hypothesis Qm >3 where 3 corresponds to "**Undecided**", and Qm is the median for question Q.
- (B) Having a p-value <0.05, which means confidence of 95% and 5% of probability of giving an error in rejecting a true null hypothesis (i.e., Type-I-error).

7.4.3 Subjects

In the first run of the survey, we involved 9 subjects, mainly experienced researchers with a good knowledge about the goal modeling in general, especially modeling with Tropos [BPG$^+$04]. All subjects are working at FBK[3] conducting research in information systems or related discipline to have balance in the subject's expertise.

7.4.4 Survey Design

For the design of the survey, inspirations have been taken from Goal-Question-Metric (GQM) approach [BCR02]. Following this approach, the survey design

[3]http://www.fbk.eu

is structured using a goal graph. Below in Fig 7.8, we present this design. Main goal of the survey is to: ***Acquire Perceived Judgment***, from a set of experienced subjects. The research questions are formulated to help satisfying

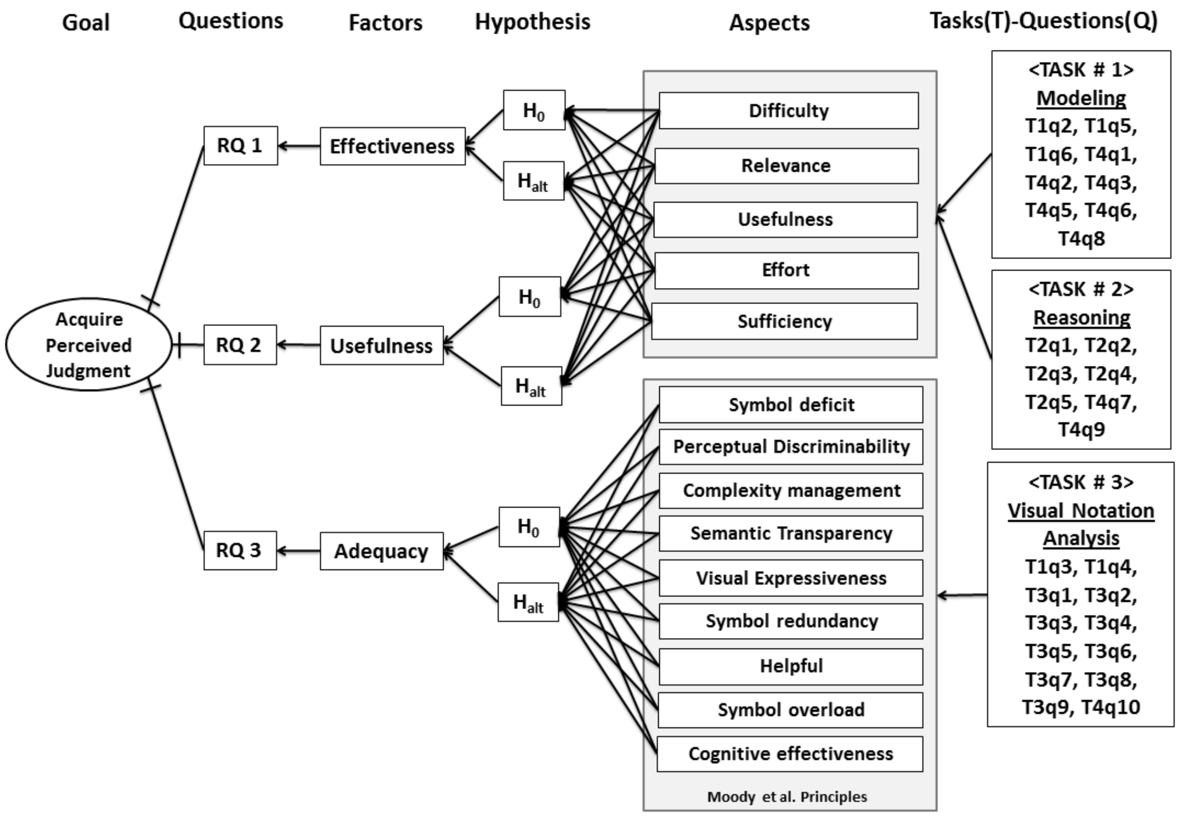
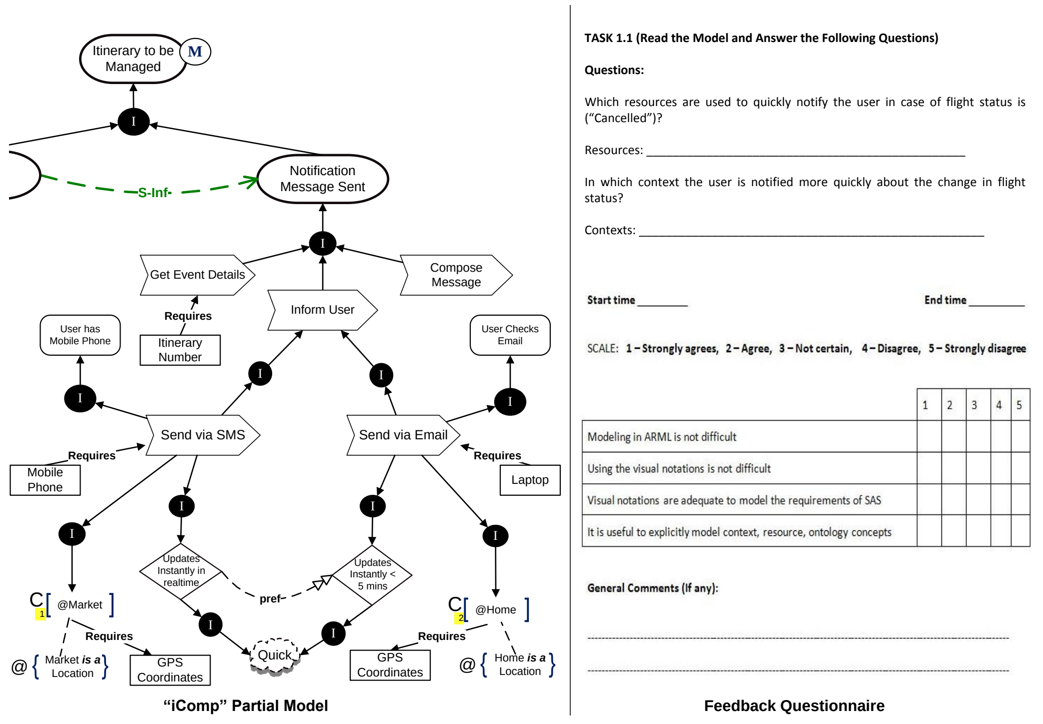


Figure 7.8: Survey Design

this goal i.e. **RQ1**, **RQ2** and **RQ3**. These questions are further refined into respective factors: *Effectiveness*, *Usefulness* and *Adequacy*. Each of this factor is then refined into two hypothesis H^0 and H^{alt} (in the same way of those described above). To gain confidence which hypothesis to accept or reject, we refined the factors into several aspects e.g. *usefulness*, *difficulty*, *effort*, *relevance*, *adequacy*. Related to these aspects, specific tasks (e.g. Task # 1) and questions (e.g. T1q2, T1q5 for Task # 1) are designed.

We designed three main tasks: **Modeling** of requirements, **Reasoning** over the requirements model, and **Visual Notations Analysis** of ARML. In each task, subjects are asked to perform small exercises and answer a questionnaire related to their perceived experience about using the concepts and notations of ARML.



TASK 1.1 (Read the Model and Answer the Following Questions)

Questions:

Which resources are used to quickly notify the user in case of flight status is ("Cancelled")?

Resources: ____

In which context the user is notified more quickly about the change in flight status?

Contexts: ____

Start time ________ **End time** ________

SCALE: **1 – Strongly agrees, 2 – Agree, 3 – Not certain, 4 – Disagree, 5 – Strongly disagree**

	1	2	3	4	5
Modeling in ARML is not difficult					
Using the visual notations is not difficult					
Visual notations are adequate to model the requirements of SAS					
It is useful to explicitly model context, resource, ontology concepts					

General Comments (If any):

Figure 7.9: Survey Material Snapshot

RQ1 is related to modeling with ARML. The subjects are asked to conduct some comprehension exercise, answer some comprehension questions about an ARML model describing a SAS system, and use ARML to evolve the model for modeling some new scenarios about the SAS system. Fig.7.9 (Left) shows, as example, a partial screen shot of the model given to the subjects and few comprehension questions we asked (Right Top). Then subjects are asked to fill the questionnaire related to their perceived experience in modeling with ARML. Fig.7.9 (Right bottom) shows, as example, some questions we asked in the questionnaire for collecting the expert feedbacks.

RQ2 is related to a reasoning task. The subjects are asked conduct some reasoning activities using an ARML model and then fill the questionnaire related to their perceived experience.

For **RQ3** and corresponding hypothesis, aspects have been mapped to Moody et al. [MHM09] principles. Here the task of subjects is visual notations analysis by answering a set of questions the ARML notation, where each question corresponds to a specific principle.

In this survey, we ask the subjects to play the role of an analyst, system (SAS) and analyst/novice user to perform modeling, reasoning and visual notations analysis tasks respectively. This is important because the type of feedback we expect requires the subject to play these roles while performing the tasks.

As a matter of design refinement, a pilot with one subject (who is an experienced researcher), is performed. Based on the acquired results, we refine our design and execute it with 8 subjects in one lab session. The candidate treatment of the survey that we use to evaluate our hypothesis is: **ARML** concepts and relations, modeling guidelines and visual notation.

We use a self-adaptive software system with requirements for adaptivity belonging to travel companion application ("iComp", hereafter) that support its user's in planning, organizing, monitoring, deciding and managing information pertaining to their travels. The material given to the subjects includes:

- Visual Notations Guide
- Requirements of "iComp"
- A partial requirements model of "iComp"
- Scenario

7.4.5 Survey Procedure

One lab session is performed. A 15 minutes tutorial on ARML, its purpose, concepts, visual notations, modeling guidelines is given to the subjects in order to be sure that subjects have understood the key elements of ARML. In addition, description is provided in the material given to the subjects and the tasks they are required to perform. The procedure for the survey is as follows:

Pre-Questionnaire: Questions about the experience of the subjects

Task 1: Modeling the requirements of SAS.

TASK 1.1:

- Read the given requirements of iComp system
- Read the given partial requirements model of iComp system modeled in ARML
- We ask the subjects to fill the questionnaire regarding the comprehension of the requirements model

TASK 1.2:

- Change the given model as per the given requirement
- Read the given partial requirements model of iComp system in ARML
- We ask the subjects to fill the questionnaire about their modeling experience

TASK 1.3:

- We ask the subjects to fill a Post-Modeling Feedback questionnaire to collect their modeling experience and overall judgment about the effectiveness of modeling in ARML.

Task 2: Reasoning over the requirements model using a given scenario

TASK 2.1:

- Read the partial requirements model of iComp system
- Read the given scenario
- We ask the subjects to answer the question on the usefulness of the concepts in the requirements model for reasoning over a given scenario, answers are expected to be descriptive

TASK 2.2:

- Read the partial requirements model of iComp system

- Read the given scenario

- we ask the subjects to answer the question on usefulness of the concepts in the requirements model for reasoning over a given scenario, answers are expected to be multiple-choice

TASK 2.3:

- We ask subjects to fill a Post-Reasoning Feedback questionnaire to collect their reasoning experience and overall judgment about the usefulness of the concepts in requirements model on which a system (SAS) can reason.

Task 3: Visual Notations Analysis

TASK 3.1:

- Read the given visual notations guide

TASK 3.2:

- We ask the subjects to fill the Post-Visual Notation Analysis Feedback questionnaire to collect their judgment about the adequacy of the notations in ARML based on their experience in performing previous tasks (modeling, reasoning) and in general.

Task 4: We ask the subjects to fill a Post-Survey Feedback questionnaire to collect their perceived judgment and experience related to Tasks (1–3). Latter we classify these questions to the relevant RQ they are referring.

7.4.6 Data Analysis and Interpretation

In this section, we statistically analyze the results of the survey. First we present the aggregated results gathered through descriptive statistical methods. For the statistical analysis we use the R tool[4].

Aggregated Results of RQ1, RQ2, RQ3

In Fig.7.10, the boxplots reports the results of the three research questions.

[4]R is a language and environment for statistical computing and graphics. It is available at http://www.r-project.org.

We decide to use the Wilcoxon test, to accept or reject the hypothesis. The feedback is collected using a Likert scale [1-5]. Subject answers to the questions asked for each task ranges between "Strong Agree" and "Strong Disagree" . The distributions of the values are spread around low values of the Likert scale. In Table 7.1, we show the median of the subject answers and

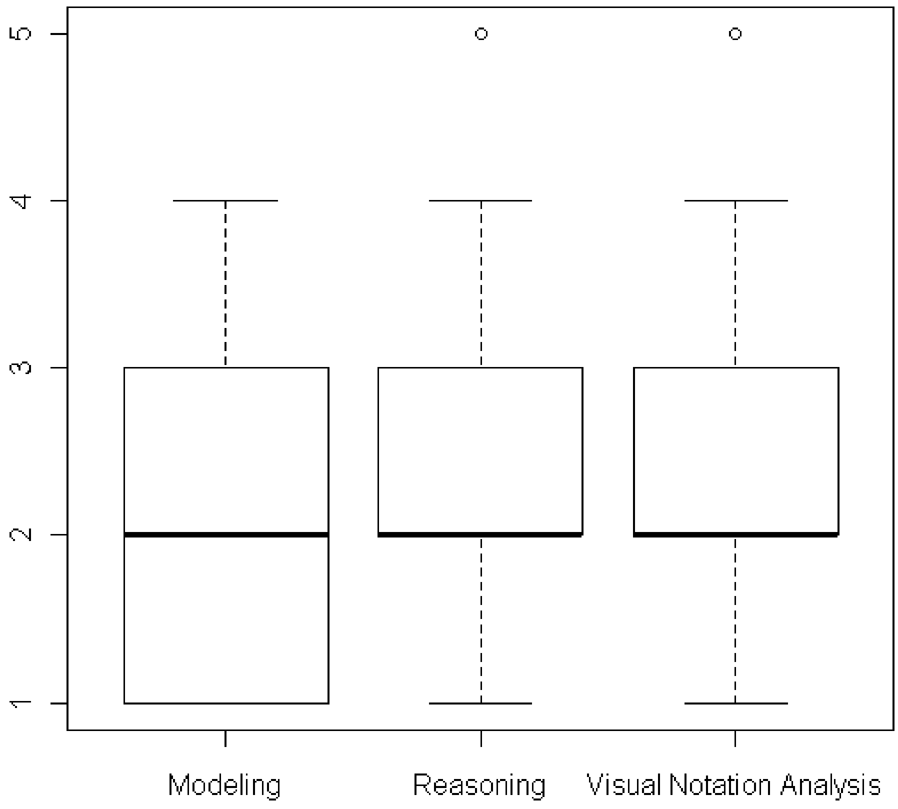


Figure 7.10: Averaged Results of Three Tasks

the respective p-Value for each **RQ**. The p-Values (i.e. <0.05) statistically confirm the observed positive trend. For instance, the distribution of the results for **RQ1** i.e. effectiveness of the modeling, is quite balanced. The obtained p-Value is significantly less <0.05 i.e (4.803e-09). In this case the perceived judgment by the subject ranges between "strong agree" and "agree". Whereas, the distribution of the results for **RQ2** i.e. on the usefulness of concept for reasoning by the system on the requirements model, shows a positive trend with a p-Value <0.05 (i.e. 8.578e-05), which statistically confirms the positive trend. In fact, the perceived judgment by the subject ranges between "strong agree" and "medium" on a Likert scale. Likewise, for **RQ3**, the distribution shows a positive trend with a p-Value <0.05 (i.e. 3.407e-09), which statistically confirms a positive trend. Actually, the perceived judgment of the subject in this case ranges between "strong agree" and "medium" on a Likert scale.

Finally, we reject the three null hypothesis related to the three **RQs** i.e. H_1^0**:**, H_2^0**:**, H_3^0**:**.

Research Questions	**Median**	**P-value**	**Hypothesis**	**Reason**
RQ1	2	4.803e-09	reject H_1^0	Agree
RQ2	2	8.578e-05	reject H_2^0	Agree
RQ3	2	3.407e-09	reject H_3^0	Agree

Table 7.1: Aggregated Results of Three Tasks (Modeling, Reasoning, Visual Notations Analysis)

Below, we present the detailed analysis of the results obtained for each research question. We use boxplots to represent the distribution of the results and tables to report the obtained statistical values. Finally, in the detailed summary table, we showed the mapping of task-questions we asked from subjects.

Detailed Analysis of RQ1

Fig.7.11 depicts the boxplots for the **RQ1**. We investigate the effectiveness of the modeling concepts proposed in the ARML. The distribution of the results for each task-question we asked from the subjects is balanced and the values range from "Strong Agree" to "Agree" on a Likert Scale.

For instance, in Table 7.2, for (T1q5) the obtained p-Value (i.e. 0.008831) statistically confirms the perceived judgment as "Strong Agree". This means that subject's comprehension about the concepts such as "Context" and "Resource" are valid for modeling requirements for SAS. For this reason, we

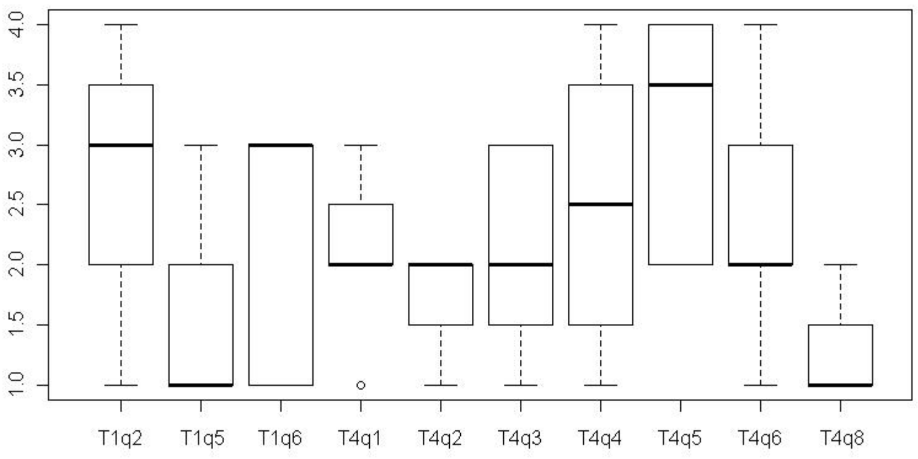


Figure 7.11: Aggregate Results of Modeling Questions

reject the H_1^0, which means the concepts are effective. However, in case of (T1q6), the obtained perceived judgment is “Uncertain” and this result is not statistically confirmed by the p-Value (i.e. 0.07446). This means that the subject’s comprehension about the relations such as “influence” and “Relegation” were either not clearly understood or they are not sure, if they are valid enough for modeling the requirements for SAS. For this reason, we could not reject the H_1^0, which means the concepts are not effective. This means, we need to improve or make it clear to the subjects about the correct use of these relations.

Modeling Questions	**Median**	**P-value**	**Hypothesis**	**Reason**
T1q2	3	0.2858	not reject H_1^0	Uncertain
T1q5	1	0.008831	reject H_1^0	Strong Agree
T1q6	3	0.07446	not reject H_1^0	Uncertain
T4q1	2	0.01313	reject H_1^0	Agree
T4q2	2	0.005139	reject H_1^0	Agree
T4q3	2	0.02667	reject H_1^0	Agree
T4q4	2.5	0.1397	not reject H_1^0	Agree/Uncertain
T4q5	3.5	0.6817	not reject H_1^0	Uncertain/Not-Agree)
T4q6	2	0.05991	not reject H_1^0	Agree
T4q8	1	0.005139	reject H_1^0	Strong Agree

Table 7.2: Aggregated View of the Results (Design-Time Modeling)

In the Table 7.3, we summarize the interpretations of the statistical results we obtained. The questions we ask from the subjects (as a feedback) are linked with a particular aspect, as shown in the Fig.7.8. These aspects are linked with the hypothesis related to the "effectiveness". We observe in Table.7.2 the question (T4q6) shows that we could not reject the H_1^0, this could be because the modeling effort is high as shown in the summary Table.7.3. It means that we need to either improve the modeling guidelines or the subject were not able to follow completely the modeling guidelines. Whereas, in Table.7.2, for the question (T4q8) we reject H_1^0. In this case, the subject's perceived judgment about the effectiveness of the modeling concepts show a promising feedback with a value of "Strong Agree" on the Likert scale. The reason for this is that the modeling concepts were clear to the subjects while performing the modeling task.

Modeling Questions	**Aspects**	**Comment**
T1q2	Difficulty in Modeling	Uncertain
T1q5	Useful Concept for Modeling	Strong Agree
T1q6	Useful Relations for Modeling	Uncertain)
T4q1	Clarity of Modeling Guidelines	Agree
T4q2	Concepts and Relations are Useful for Modeling	Agree
T4q3	Adequacy of Modeling Concepts and Relation	Agree
T4q4	Difficulty in using Modeling Concepts	Agree/Uncertain
T4q5	Difficulty in using Modeling Relations	Uncertain/Not-Agree
T4q6	Effort in Modeling	Agree
T4q8	Relevant and Useful Concepts for Modeling	Strong Agree

Table 7.3: Modeling Tasks-Questions Mapping to Aspects and Comments

Detailed Analysis of RQ2

Fig.7.12 depicts the boxplot for the **RQ2**. We investigate the usefulness of the modeling concepts in the requirements model for reasoning by the system (SAS) itself. In this case, we asked the subjects to play the role of a running system. The distribution of the results for each task-question we asked from the subjects is between and the values range from "Medium" to

"Agree" on a Likert scale. For instance, in Table 7.4, for (T2q4) the obtained

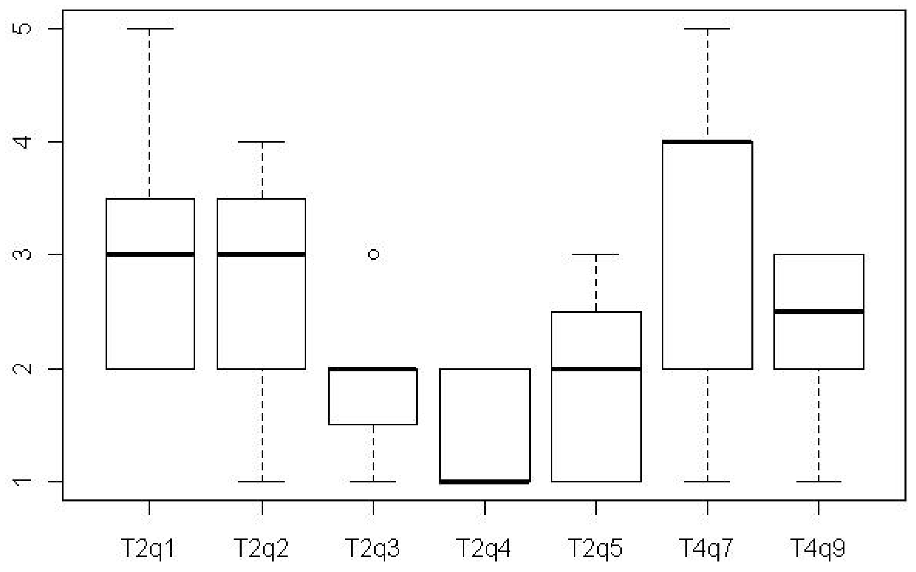


Figure 7.12: Aggregate Results of Modeling Questions.

p-Value (i.e. 0.00577) confirms the perceived judgment with a value "Strong Agree" on Likert scale. The reason is that subject's were able to reason on the requirements model using concepts such as "Context" and "Resource". As per their perceived judgment we reject the H_2^0, which means the concepts are useful. However, in case of (T4q7), the perceived judgment is "Not Agree" on a Likert scale. This means that the subject's effort to reason on the given scenario using the requirements model is high. Moreover in case of (T4q7) the obtained p-Value $>$0.05 (i.e. 0.7213), hence we could not reject the H_1^0. To interpret this trend, the justification is that the task we asked from the subject is to reason on the requirements model by playing the role of a running SAS. In fact this task is more effectively performed by the SAS in reality. Since there are many constraints and properties to consider by the system to perform this kind of reasoning. In short, the overall perceived judgment of the subjects on the usefulness of the concepts for reasoning shows a positive trend.

In the Table 7.5, we summarize the interpretation of the obtained statistical

Reasoning Questions	Median	P-value	Hypothesis	Reason
T2q1	3	0.5562	not reject H_2^0	Uncertain
T2q2	3	0.2858	not reject H_2^0	Uncertain
T2q3	2	0.008831	reject H_2^0	Agree
T2q4	1	0.00577	reject H_2^0	Strong Agree
T2q5	2	0.01601	reject H_2^0	Agree
T4q7	4	0.7213	reject H_2^0	Not Agree)
T4q9	2.5	0.04449	reject H_2^0	Agree/Uncertain

Table 7.4: Aggregated View of the Results (Run-Time Reasoning)

results. The questions we ask from the subjects (as a feedback) are linked with a particular aspect, as shown in the Fig.7.8. These aspects are linked with the hypothesis related to the "usefulness". We observe in Table.7.4 the question (T4q9) shows that we reject the H_2^0. The obtained p-Value (0.04449) is <0.05. Whereas, the distribution of the results range between the values "Agree" and "Uncertain" on the Likert scale. We observe this trend in the summary Table.7.5, where we see that the subjects agree that the concepts are useful for reasoning, however, requires high effort to reason (e.g. the case of T4q7). This shows that concepts in the requirements model are clear to the subjects. Actually, the reasoning task is not trivial for human subjects, as it requires several parameters to consider (e.g. concepts and relations checking) to perform reasoning. Therefore, the results obtained in this particular case shows a values i.e. "Not Agree" on the Likert scale.

Detailed Analysis of RQ3

Fig.7.13 depicts the boxplots for the **RQ3**. We investigate the adequacy of the visual modeling notation proposed in ARML. We asked the subjects to analyze the visual notations following the visual notation guide given to them (the guide shows the visual notations, the definition of the concept they refer to and some examples).

The distribution of the results for each task-question we asked from the subjects ranges between the values "Medium" to "Agree" on a Likert scale. For instance, in Table 7.6, for (T3q2) the obtained p-Value (i.e. 0.01264)

Reasoning Questions	Aspects	Reason
T2q1	Difficult to Reason on Concept and Relations in Requirements Model	Uncertain
T2q2	Sufficient Concepts and Relations in Requirements Model to Reason	Uncertain
T2q3	Relevant Concepts and Relations in Requirements Model for Reasoning	Agree
T2q4	Useful Concepts in Requirements Model for Reasoning	Strong Agree
T2q5	Useful Relations in Requirements Model for Reasoning	Agree
T4q7	Effort in Reasoning over Requirements Model	Not Agree
T4q9	Useful to Reason on Requirements Model	Agree/Uncertain

Table 7.5: Reasoning Tasks-Questions Mapping to Aspects and Comments

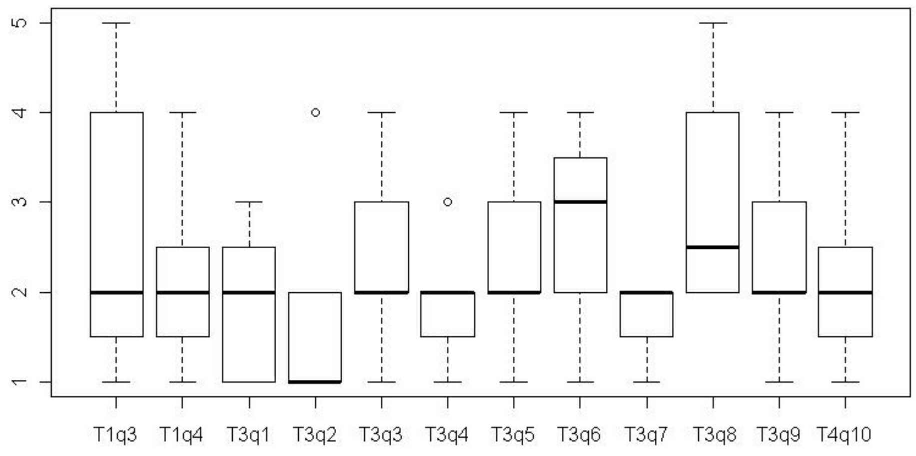


Figure 7.13: Aggregate Results of Visual Notations Questions.

<0.05, which statistically confirms the perceived judgment as “Strong Agree” on the Likert scale. Reason for this is that the subjects read the visual notations guide and are able to distinguish between each notation clearly. The aspect we link to this question is one of the Moody et al’s [MHM09] principle i.e. Perceptual Discriminability, which states that graphical symbols should be clearly distinct from each other. However, in case of (T3q8), the obtained p-Value (i.e. 0.536), which does not statistically confirms the results. It results into values “Agree/Uncertain” on the Likert scale. This question is linked with the Moody et al. principle i.e Complexity management, which states

that graphical symbol must posses explicit way to deal with complexity. The question relates to the increased complexity of the model using these visual notations. The results reveal a positive trend, however some of the subject were uncertain about this. For this reason, we could't reject H_3^0. Overall, the perceived judgment about the adequacy of the visual notations shows a positive trend.

VN Questions	**Median**	**P-value**	**Hypothesis**	**Reason**
T1q3	2	0.258	not reject H_3^0	Agree
T1q4	2	0.03249	reject H_3^0	Agree
T3q1	2	0.01601	reject H_3^0	Agree
T3q2	1	0.01264	reject H_3^0	Strong Agree
T3q3	2	0.05991	not reject H_3^0	Agree
T3q4	2	0.008831	reject H_3^0	Agree
T3q5	2	0.05991	not reject H_3^0	Agree
T3q6	3	0.2858	not reject H_3^0	Uncertain
T3q7	2	0.005139	reject H_3^0	Agree
T3q8	2.5	0.536	not reject H_3^0	Agree/Uncertain)
T3q9	2	0.05991	not reject H_3^0	Agree
T4q10	2	0.03249	reject H_3^0	Agree

Table 7.6: Aggregated View of the Results (Visual Notation Analysis)

In the Table 7.7, we summarize the statistical results to better interpret them. The questions we ask from the subjects are linked with the Moody et al. principles as shown in Fig. 7.8. These principles are linked with the hypothesis, which enables us to accept or reject based on the feedback of the subject about the "adequacy" of the visual notations in ARML.

We observe in Table.7.4 the question (T3q6) shows that we could not reject the H_3^0, this is because the obtained p-Value (i.e. 0.2858) is >0.05, however the distribution of the feedback is around the values "Uncertain" on a Likert scale. In Table.7.7, we show the interpretation based on the results obtained. The justification is that we asked the question: *is it easy to remember the visual notations*, which is linked the principle: "Symbol Redundancy". Following newer notation is not so easy to remember, however the trend is towards the positive values, but we could't reject the hypothesis. Finally, we conclude

VN Questions	Moody's Principle	Comment
T1q3	Semantic Complexity	Uncertain
T1q4	Relevant to Model Requirements for SAS (not a Moody's Principle)	Agree
T3q1	Symbol Deficit	agree
T3q2	Perceptual Discriminability	Agree
T3q3	Semantic Transparency	Uncertain
T3q4	Visual Expressiveness	Agree
T3q5	Symbol Overload	Uncertain
T3q6	Symbol Redundancy	Uncertain
T3q7	Helpful for purpose (not a Moody's Principle)	Agree
T3q8	Complexity Management	Agree/Uncertain
T3q9	Cognitive Effectiveness	Agree
T4q10	Cognitive Effectiveness	Agree

Table 7.7: Visual Notation Analysis Tasks-Questions Mapping to Moody et al Principles [MHM09])

from this result that subject were able to understand the visual notations, their meaning and the concepts attached to these notations. Our overall hypothesis about the adequacy of the visual notations is positive.

7.5 Final Remarks

Summarizing the contributions of this chapter, we first reflect on the set of experiments conducted on evaluating the performance and scalability of the design-time tool for specifying adaptive requirements and later following the process as described in (Section5.4), we derive monitoring specification pertaining to each task that conforms to a candidate solution in a service-based application setting (we operationalize each task through an available service description). The initial results obtained show better performance with respect to increasing number of model size.

The second evaluation is on the qualitative assessment of the requirements models. We modeled the example scenario taken from the travel companion case study. First we modeled it with *i** and then with *Tropos*. We then modeled the same with ARML. Comparing the models we provide an initial assessment that existing goal modeling languages such *i** and *Tropos* lacks modeling constructs to model the properties of a SAS-to-be e.g. domain assumptions, context etc. While modeling in ARML using its visual notations, which have graphical mapping with the revise core ontology of RE for SAS and Techne, we were able to represent the required properties of SAS-to-be. This assessment is based on the modeling elements and requirements modeled.

To collect test our hypothesis that ARML provides better modeling constructs to model requirements for SAS in comparison with existing goal modeling languages, we performed an empirical survey experiment. We involved with a small number of expert people with experience ranging between 2-11 years (on the average 6 years) working in requirements engineering especially requirements modeling using e.g. standard *Tropos*. The main motivation of conducting this empirical survey was to collect feedback. Fig. 7.8 shows the survey design and the factors we investigated. The survey was subdivided in three tasks with small exercises with accompanied questionnaires to collect feedback. With this survey the involved subjects tried modeling requirements and reasoning over the requirements model of a SAS using ARML. The feedback collected based on the with the perceived judgment of the subject after performing the given tasks e.g. modeling requirements (playing the role of an analyst), reasoning over requirements model (playing the role of a SAS) and analyzing visual notations of ARML. We acquired this feedback such that the subjects implicitly compare ARML with their experience, e.g., in *Tropos*. To be sure that the subjects have adequate/sufficient knowledge to give us feedback, we: (i) gave a tutorial on ARML; and (ii) asked the subjects to do some exercises related to the tasks given to them. We collected their feedback using the Likert scale (1-5) questionnaires.

The results achieved are promising. We can accept our alternative hypothesis related to each tasks with statistical evidence i.e. modeling in ARML is effective (RQ1), reasoning over ARML models by the systems is useful

(RQ2) and visual notation proposed in ARML are adequate for modeling requirements for SAS (RQ3). With detailed analysis we can conclude that some improvements in ARML modeling concepts and relations such as further clarification on their use or explanation can help subjects to model more effectively.

Chapter 8

Conclusions and Future Work

8.1 Conclusions and Summary of Contributions

The motivation of this thesis work roots in the need of a deeper investigation of key challenges raised in ongoing research on requirements engineering for *self-adaptive software systems*, which can be stated as:

- traditional design-time decision (e.g. selection of the appropriate system behavior) has to be delayed as much as possible; and
- requirements artifact of *self-adaptive software systems* need to be kept alive at run-time [SBW+10].

While approaching these research problems, we investigated how requirements engineering for *self-adaptive software* departs from conventional requirements engineering and ultimately how to bridge the gap between design-time and run-time requirements engineering.

We introduced core concepts that are needed to perform RE for *self-adaptive software*, thus providing a conceptual basis for formulating the requirements problem as a dynamic problem. In our opinion, this approach goes beyond state-of-the art works, which address requirements engineering for *self-adaptive software* proposing methods to anticipate at design-time situations (conditions) for run-time adaptation and to analyze them in terms of alternative solutions to be instantiated at run-time. In fact, our framework aims at realizing "continuous RE", also enabling SAS to perform continuous RE by itself i.e. continuous reappraisal of its requirements at run-time involving the end-user.

We now summarize the core contributions of this thesis work:

1. ***Conceptual and Theoretical Framework:***

   We introduced a theoretical framework that provides conceptual grounding for the concepts needed to perform RE for *self-adaptive software*. Using this framework the requirements problem for SAS can be formulated as a dynamic problem. Dynamicity is due to changes in the end-user needs (e.g. goal, preferences), context and availability of the resources thereby finding a candidate solution that resolves it. This conceptual framework rests on and extends the recently revisited core ontology for RE that is founded on the concepts from belief, desire, intentions, goal-oriented modeling and foundational ontology. We revised the revisited core ontology and added concepts: "Context and Resource" and relations: "Influence and Relegation". This contributes to the first objective of this thesis (outlined in Section 1.2). Although, the conceptual framework is general and takes no assumptions or bound to specific implementation constraints, concretely, operationalization of the concepts proposed requires implementation issues to be taken into account. More sophisticated mechanisms could be used e.g. machine learning, automated or probabilistic reasoning that may provide more formal support to these concepts and relations proposed in this framework.

2. ***Engineering Adaptive Requirements:***

   The revised core ontology for RE for *self-adaptive software systems* provide necessary concepts to model their requirements and perform early analysis over these models. An analyst could acquire information from the end-user or domain analysis or through conventional inquiry method (e.g. interview, market studies etc.), where the core concepts supports in structuring this communicated information. To fulfill our first objective (outlined in Section 1.2), the analyst at design-time, which starts at early requirements (elicitation and analysis) and ends with the specification of the system satisfying the requirements, could model and perform early analysis over the requirements in terms of *adaptive requirements*. Adaptive requirements are not a primitive concept but they aggregate dynamic information about requirements making explicit the feedback functionalities i.e. monitoring specification, evaluation criteria and adaptation actions.

To represent these requirements we proposed a requirements modeling language, called Adaptive RML (see Section 5.3). We introduced a systematic modeling guidelines and visual modeling notations, which has graphical mapping to Techne - an abstract and formal RML. We adopted a convenient formalism based on event-condition-action pattern to operationalize adaptive requirements. This provides a way to derive monitoring specification, which are used to configure an adaptive service-based application (Section 5.4). We developed a proof of concept tool to support the process of specifying adaptive requirements (from the requirements model) and deriving the monitoring specification automatically from the adaptive requirements specification.

3. ***CARE Framework:***

   Continuous changes occur in the real context, which maybe caused due to change in end-user needs or preference, variation in context and availability of resources demanding change in the requirements problem formulated for *self-adaptive software* at design-time. Thus, to enable *self-adaptive software* to be aware about its own requirements and to adapt in response to continuously to these changes, we introduced a continuous adaptive requirements engineering (CARE) framework. It provides support for continuous refinement of requirements by the *self-adaptive software* itself involving the end-user at run-time. New or changed requirements requires new solutions (i.e. available services) to be searched, thus leading to refinement of the existing specification. At this stage *self-adaptive software* acts as an analyst. We described it's details in Chapter 4. In fact, CARE operates based on the defined adaptation types to determine the level of RE it has to perform. We defined four adaptation types (see Section 4.2). Concepts and artifacts that could be used at design- time or at run-time were defined. SAS while in operation, instantiating CARE, exploits these artifacts which are based on the revised core ontology of RE for *self-adaptive software systems*. For *self-adaptive software* operating in an open environment i.e. the Internet and services, we defined a conceptual architecture that instantiates CARE at run-time (see Section 4.4).

Contributing to the second objective of this thesis outlined in Section 1.2, CARE provides a way for *self-adaptive software systems* to perform continuous RE involving the end-user at run-time by being aware of the dynamic changes. Taking into account the wide land- scape of research on method and techniques on software engineering of *self-adaptive software systems*, and more recently on requirements-aware systems, CARE framework provides a novel approach by keeping the representation of the requirements at run-time, enabling the *self-adaptive software systems* to perform RE by itself involving the end-user. As per our knowledge, no existing work argues this form of RE to support run-time adaptation. More sophisticated mechanisms for web service composition or orchestration and a easy natural language approach to search and add web services could improve CARE framework for requirements acquisition and solution provisioning at run-time.

We evaluated our CARE framework through application to the case study (see details in Chapter 6). For design-time approach we performed performance and scalability tests for the proof of concept design-time tool and a empirical survey study on the effectiveness of the proposed ARML involving the experienced subjects (see details in Chapter 7).

8.1.1 Generality and Limitations

The types of applications that were considered in this thesis work involves the end-user mainly representatives of service-based applications and mobile softwares e.g. health-care, e-commerce, productivity softwares. In that sense, goals, contexts and preferences of human actors as well as information on what activities they perform cannot be assumed to be fully observable. Adaptation may take an unpredictable amount of time to produce a result i.e. a revised solution to the changed requirements problem. In this context, formulation of the requirements problem is not intended to support the automated solution as a satisfiability problem [GMNS03], rather as an optimization problem, where satisfiability may be exponential. For instance, when the model size exceeds a minimum size its may be problematic for many practical applications. Therefore, adopting classical AI planning could provide a candidate approach for performing the reasoning task at run-time by the SAS itself. A further investigation of systematic and accessible approaches for structuring domain theories for efficient yet meaningful computation is strongly motivated by recent results [LMM10b, QLP11].

Furthermore, expanding on the run-time adaptation, CARE framework provides a novel approach to overcome the dynamic RE problem i.e. finding solutions to the changed requirements problem taking into account the contextual changes, resource availability and end-user preference by the system itself involving the end-user. Thus, a seamless synchronization with human-accessible requirements models of goals and domains is necessary. Requirements model expressed exploiting the concepts in revised CORE ontology of RE for SAS and specified as adaptive requirements provides ample tool set for the SAS at run-time to reason on its own requirements.

8.2 Future Works

Several interesting directions have been identified during this thesis work. On some of the ideas initial exploration has been conducted, providing preliminary results that we summarize in the rest of this chapter, together with future research directions.

8.2.1 Integrating Adaptive Requirements and Preferences-based Reasoning for Run-time

System are becoming more and more pervasive, where end-user needs and preference plays a critical role in customizing and adapting at run-time. In this thesis, we introduced a new class of requirements, adaptive requirements, to specify properties for SAS. In our view, we see the requirements problem

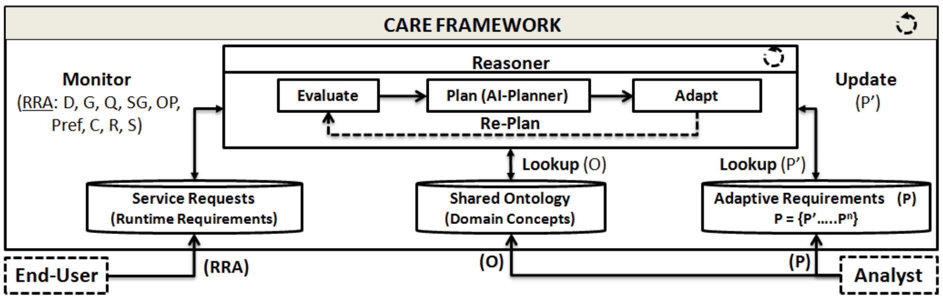


Figure 8.1: Continuous Reasoning Process at Run-Time

at run-time as an optimization/planning problem. Therefore, methods that support runtime reasoning of adaptive requirements are needed. Recently in [LMM10b] Liaskos et al. have proposed preference requirements expressed as preference goals, which plays a critical role in customizing software solutions exploiting preference based AI planning. The role of end-user is critical (as proposed in this thesis) while reasoning about the adaptive requirements and run-time adaptivity. Thus, integrating preference goals in adaptive requirements can play an important role. We plan to investigate the usefulness of recently proposed techniques for automated reasoning with requirements goal and preference models [LMM10b] to support runtime adaptivity. As a first step, we extend adaptive requirements by integrating preference-based reasoning and automated planning to enable a continuous adaptive reasoning of requirements at runtime (see Fig.8.1). We have presented this vision using a navigation system example and highlighted challenges for further research in [QLP11]. We investigated how reasoning about user goal and preference models, supported by state-of-the-art AI planning techniques, can effectively play this decision-making role, through recursively revising goal, preference and contextual models to reflect real world change and subsequently reasoning about new solutions in the revised situation. The major benefits of this approach are that the decision making mechanisms of SAS align directly with human-accessible requirements models, facilitating thereby systematic engineering and accessibility both at design-time and at run-time. They are also based on well-studied automated reasoning technologies that have demonstrated significant progress over the past years. To support explore on this idea, we aim to exploit a real case study where we can exercise this approach and provide empirical evidence.

8.2.2 Online Requirements Engineering

Self-adaptive software systems can manage dynamic events occurring at run-time, such as unavailability of services, hardware and platform changes as well as the change of a users preferences and needs. Engineering such applications significantly challenges the role of requirements engineering (RE).

Usually, RE activities are carried out at the outset of the whole development process, but in the context of self-adaptive SBA, they are also needed at run-time thus enabling a seamless SBA evolution. This also includes that SBA should be able to capture, understand and satisfy end-user needs while they emerge. The end-user involvement has been recognized important during the requirements elicitation in situ. Seyff et al. proposed a requirements blogging tool (iRequire) to support end-user to express needs in situ [SGM10]. In this thesis, we extended this view where SAS while performing continuous RE involves the end-users supporting goal- and user-oriented adaptation. Integrating iRequire and CARE framework, we aim to investigate an On-line RE approach (see Fig.8.2) i.e. requirements acquisition by involving the end-user and the system itself. To fulfill this aim, Online RE can be performed on the fly using iRequire and CARE to provide requirements capturing and analysis capabilities enabling the end-users to communicate (new) needs, which are then turned into new (or changed) requirements. These requirements have to be satisfied through a combination of available services or left as new requirements to be addressed off-line within a software evolution process. We

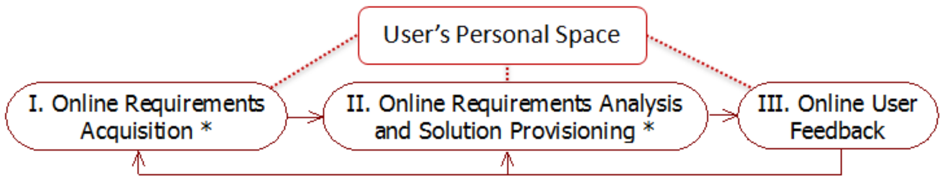

User's Personal Space
I. Online Requirements Acquisition *
II. Online Requirements Analysis and Solution Provisioning *
III. Online User Feedback

Figure 8.2: Online RE process

presented our planned research aiming at investigating the above described idea in [QSP11]. We envision to combine and extend our recent work to come up with a tool-based approach that involves end-users in the realization of their needs using self-adaptive SBA. As a further step in this direction, we are investigating novel techniques for the provisioning of end-users feedback so that mobile vendors can exploit this feedback to improve their applications and services. Apart from this, we are also investigating knowledge representation techniques to better represent the concept of “personal space”.

8.2.3 Towards the framework for Evolving Requirements

In this thesis work, we presented continuous RE framework and defined adaptation types (1–4), where online evolution is supported by the SAS itself involving the end-user. Offline evolution is recognized as a type of system’s adaptation that usually needs to be realized offline, involving human intervention (analyst/designers and other stakeholders) [QP10, QPEM10]. Extending

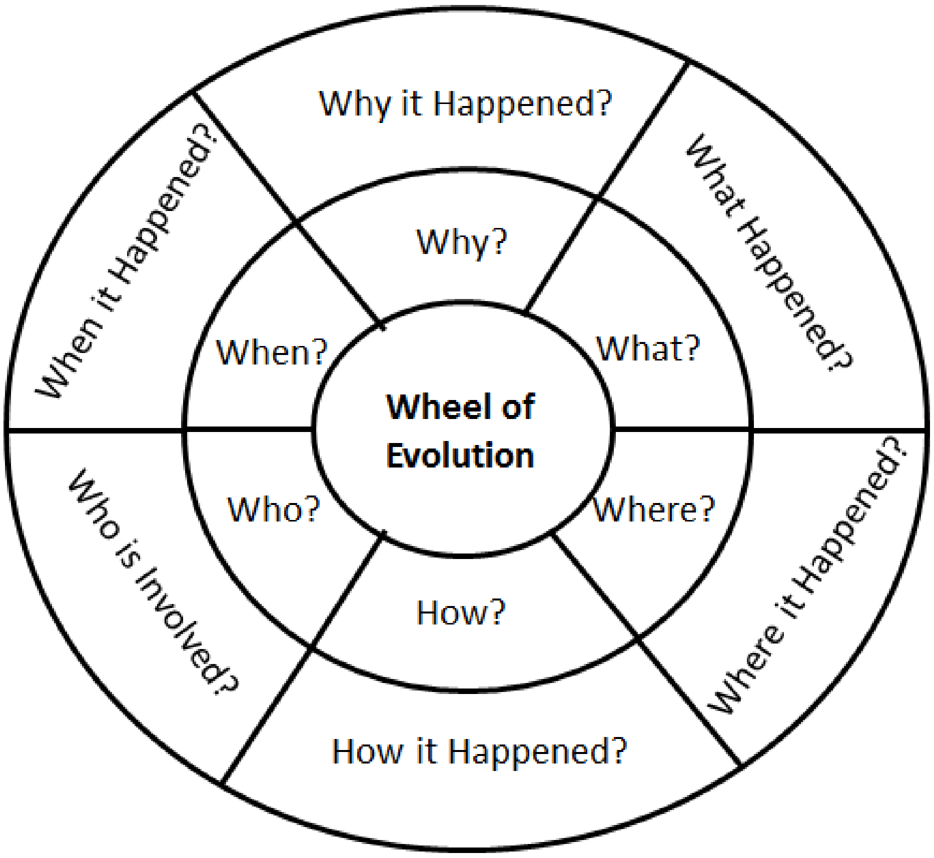


Figure 8.3: Wheel of Evolution

this view, we envisioned a framework which provides concepts and analysis techniques to support an evolutionary and “participatory” process for requirements evolution in STS (a socio-technical system, which involves human end-users). Incorporating ideas from participatory design[Nie04, LSSZ10] and law compliance [SMPS09], we investigated methods that can be used to support the analysis of these types of requirements changes and ultimately on system evolution, with the involvement of key stakeholders. adopt a participatory design approach as process backbone, to enable the participation of stakeholders in design activities [CRC07, Nie04], including the software systems (within the STS). Participatory design provides peculiar techniques for engaging participants, such as scenarios and of "persona" (sort of fictional user) or ethnographic study techniques. We analyzed the integration of specialized techniques, which are suitable for the analysis of particular concerns, as for instance law-compliance. To support the whole process of analysis, we proposed a wheel of evolution that guides our whole framework (see Fig 8.3). We reported our experience and preliminary investigation using this framework in [PQS^{+}11]. Along this direction, we aim to provide a taxonomy of changes that can guide analysis while evolving the software. Moreover, decision making theories could play a critical role in this work, we are interested to provide a library of decision mechanisms at support of the system and also for the analyst to evolve requirements.

8.2.4 3T process integrating concepts from Agent systems and Testing in RE for SAS

Along the direction of a tool-supported development process supporting multi-level feedback during the development life cycle of SAS is needed. We proposed *3T* process, a systematic requirements-driven approach, to develop adaptive systems by combining goal-oriented and agent-oriented design concepts in [QP08]. Extension to this work, we involve an automated testing paradigm that compliments this process to provide a tool supported process preserving requirements along the development process to ensure traceability, and providing early feedback on the design revisions gathered through automated continuous testing performed at runtime once an executable version of the system is available (see Fig.8.4). Preliminary results showing

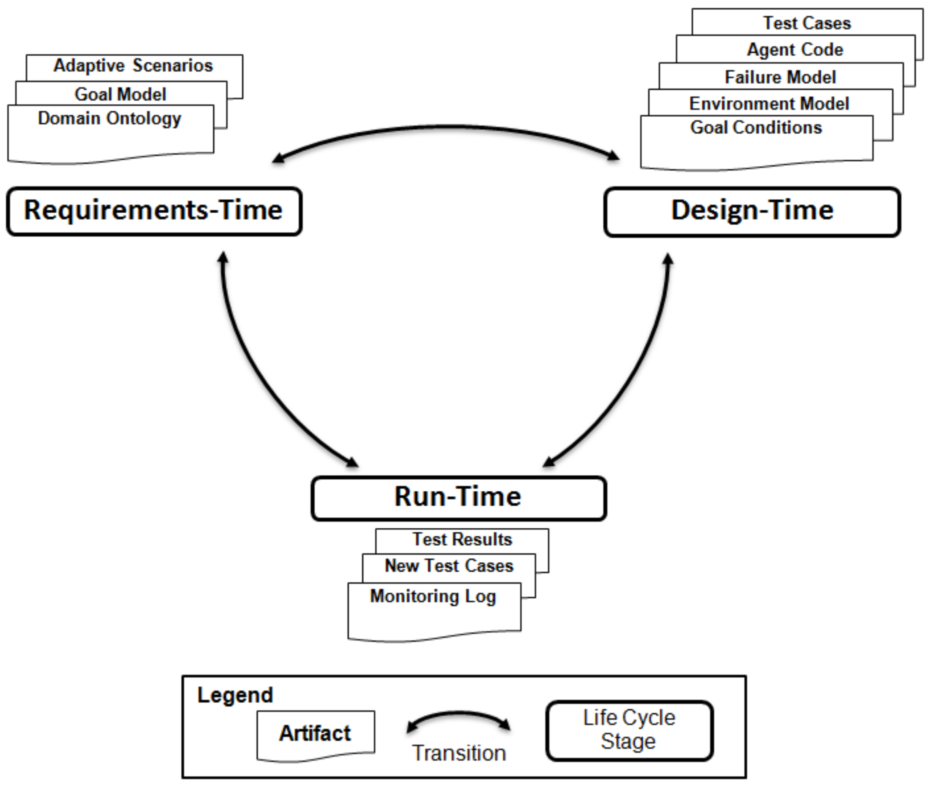


Figure 8.4: 3T AS development cycle

improvement in adaptivity properties of the system applied to an exemplar case study are discussed in [QMNP10]. We envision to further extend this work providing a unified process that can support the development of adaptive systems in a process-oriented way. Recently, Ghezzi has presented an invited talk at SEAMS 2011[1], where in this talk, he has identified the key differences between development-time and run-time with respect to the classical problem formulation of Zave et al. [ZJ97a]. Summarizing this, Ghezzi argues that classical software engineering practices must be shifted to run-time, this results in feature-rich functionalities and monitoring and reasoning capabilities to the systems. Thus, the boundary between development-time and run-time is diminishing. This thesis work supports this idea. Though, we take the

[1]http://www.hpi.uni-potsdam.de/giese/events/2011/seams2011/keynoteGhezzi.html

perspective of requirements engineering, where changes in the external environment (e.g. contextual or resource availability) motivates the SAS to adapt and continuously refine its requirements. Contributions presented in this thesis work address the problem to bridge the existing gaps between design-time to run-time. A further consolidated effort is needed, which is an ongoing work to provide a process-oriented approach realizing the goal of seamless adaptation.

8.2.5 Analyzing Intentional Interoperability Requirements

Service-based Applications (SBA) for the future Internet of Services exploit existing web services, to enable end-users accessing real services. This makes the role of the specific end-user even more crucial and demands that SBA are aware of their users' goals and preferences and that they ensure seamless inter operation among the heterogeneous services, accordingly. In the light of the adaptivity property of SBA, we investigate in this work a systematic requirements engineering approach to analyze ***intentional interoperability*** requirements, resulting from the analysis of the individual prosumers' goals and preferences while acting in the Internet of Services, and the social dependencies to satisfy them i.e. what we call "intentional interoperability". In this context, goal-oriented and process modeling techniques provides ample support. We envision a framework that rests on a set of concepts, models and analysis steps that guides the software engineer to investigate interoperability requirements by following a top-down approach. Fig. 8.5 attempts at giving a unified picture of concepts, models and analysis steps of the framework. The analysis output will be a set of intentional interoperability requirements characterized in terms of intentions (social dependencies), services and resources interoperability needs, and service qualities. Recently, Dalpiaz [DGM12] has proposed social variability in context of socio-technical systems, where agent-based interactions are formalized using commitments. We are near to this purpose, however we target more open and feature rich environments (e.g.

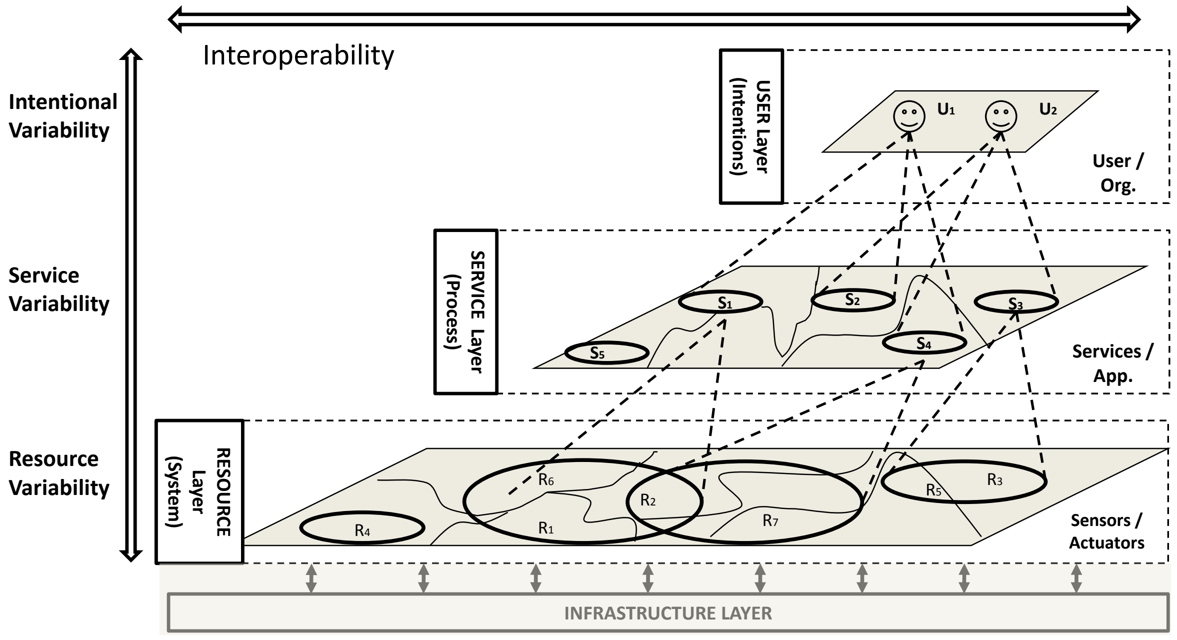


Figure 8.5: Analysis Framework for Interoperability

the Internet and service-based systems) with an aim to analyze intentional interoperability. As a further step in this direction, we aim to link this framework to more practical techniques e.g natural language processing and feature modeling.

8.2.6 Towards a case tool for operationalizing Adaptive Requirements

Approaches addressing these challenges adopt existing requirements methods, modeling languages and techniques to specify requirements monitoring for SAS. A convenient and easy approach to specify requirements monitoring for SAS is still missing. We presented preliminary tool to specify adaptive requirements that leads to monitoring specification. As an ongoing work, we focus on this issue. In fact, we aim to develop a more integrated case tool, where requirements are expressed using the tool set presented i.e. Adaptive RML, and later device operational adaptive requirements specification such that monitoring specification and configurations can be derived automatically.

Initial ideas has been summarized along this line in [OQF$^+$11]. Further steps to consolidate and experimentation are ongoing.

8.2.7 Empirical Study on ARML

Adoption of goal-oriented languages to model requirements for SAS is evident in recent research works [SBW$^+$10]. Common in all such proposals is the use of available RML such as iStar / Tropos and KAOS, the most widely used goal-oriented modeling languages for requirements modeling. However, there has been limited consensus on the necessary concepts needed to perform RE for SAS. To this aim, in this thesis work we presented a visual requirements modeling language – visual RML, Adaptive Requirements Modeling language (ARML), based on the revised core ontology of RE for SAS presented in our recent work [QJP11]. This language has graphical primitives in line with classical goal modeling languages and is formalized via a mapping to Techne [JBEM10a], an abstract formal RML. We presented results from an empirical survey study on the effectiveness and usefulness of this language in this thesis involving experienced subjects. Along this direction, a further step is to conduct an empirical experiment on a larger scale involving subjects from academic institutes.

8.2.8 Continuous Refinement of Requirements at Run-time

The need of continuous RE motivated in this thesis work provides further investigation in this direction. Extending the work on CARE framework, we are investigating a method to support continuous refinement of requirements at run-time. In this we work we have identified that continuous changes requires re-formulation of the requirements problem. For this reason, our proposed revised core ontology of RE for SAS needs to be instantiated at run-time by the SAS itself. To put into practice we are studying the metaphor of "schema evolution", that has been long investigated in database and knowledge representation research. The key idea behind this continuous refinement of requirements is to exploit at best the representation of requirements that a *self-adaptive software* holds with itself at run-time. Change operators needs to be defined along with rules that determines when or where to refine the requirements and who has to be involved e.g. end-user. We adopt ontologies as a convenient practical modeling and reasoning tool to represent requirements that at run-time. Using this representation enables to analyze change not only at instance level but also at concepts level. During operation changes are detected by *self-adaptive software* and correspondingly reasoning is performed over requirements (represented as ontologies) to evaluate the change. As a result, if there is a change in requirements, the *self-adaptive software* itself evolve (refine) them using the change operator involving the end-user. During this continuous refinement, *self-adaptive software* maintains the consistency of its requirements.

Bibliography

[ADG08] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Location-based software modeling and analysis: Tropos-based approach. In Qing Li, Stefano Spaccapietra, Eric S. K. Yu, and Antoni Olivé, editors, *ER*, volume 5231 of *Lecture Notes in Computer Science*, pages 169–182. Springer, 2008.

[ADG09] R. Ali, F. Dalpiaz, and P. Giorgini. A goal modeling framework for self-contextualizable software. In *14th International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD09)*, Amsterdam, The Netherlands, 08/06/2009 2009. Springer, Springer.

[BCR02] V. Basili, G. Caldiera, and H. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, 1:578–583, 2002.

[BCZ05] D. M. Berry, B. H. Cheng, and J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *11th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ05)*, pages 95–100, 2005.

[BGH+07] Kamal Bhattacharya, Cagdas Evren Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In Gustavo Alonso, Peter Dadam, and

Michael Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2007.

[BP10] Luciano Baresi and Liliana Pasquale. Live goals for adaptive service compositions. In *SEAMS '10: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 114–123, New York, NY, USA, 2010. ACM.

[BPG$^+$04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agentoriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[BPS10] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. Fuzzy goals for requirements-driven adaptation. In *18th IEEE Int. Requirements Eng. Conf.*, pages 125–134, Sydney, Australia, 2010.

[Bru08] Yuriy Brun. Building biologically-inspired self-adapting systems. In *Software Engineering for Self-Adaptive Systems*, number 08031 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl, Germany, 2008.

[BSG$^+$09] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger M. Giese, Holger Kienle, Marin Litoiu, Hausi A. Mller, Mauro Pezz, and Mary Shaw. Engineering self-adaptive systems through feedback loops. *Software Engineering for Self-Adaptive Systems*, 5525:48–70, 2009.

[CdLG$^+$09] Betty H.C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. *Software Engineering for Self-Adaptive*

Systems: A Research Roadmap, pages 1–26. Springer-Verlag, lncs 5525 edition, 2009.

[CGI$^+$08] Betty H. C. Cheng, Holger Giese, Paola Inverardi, Jeff Magee, and Rogério de Lemos. Software engineering for self-adaptive systems: A research road map. In *Software Engineering for Self-Adaptive Systems*, volume 08031 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl, Germany, 2008.

[CHG$^+$04] ShangWen Cheng, AnCheng Huang, David Garlan, Bradley R. Schmerl, and Peter Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. In *ICAC*, pages 276–277. IEEE Computer Society, 2004.

[CRC07] A. Cooper, R. Reimann, and D. Cronin. *About Face 3*. Wiley, 2007.

[CSBW09a] Betty Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In Andy Schürr and Bran Selic, editors, *International Conference on Model-Driven Engineering Languages and Systems*, volume 5795 of *Lecture Notes in Computer Science*, pages 468–483, Denver, Colorado, 2009.

[CSBW09b] Betty H. C. Cheng, Peter Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *MoDELS*, volume 5795 of *Lecture Notes in Computer Science*, pages 468–483. Springer, 2009.

[Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.

[DGM09] Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. An architecture for requirements-driven self-reconfiguration. In Pascal van Eck, Jaap Gordijn, and Roel Wieringa, editors, *CAiSE*, volume 5565 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2009.

[DGM12] Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Adaptive Socio-Technical Systems: a Requirements-driven Approach. *Requirements Engineering*, 2012. To appear.

[DNGM$^+$08] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engg.*, 15(3-4):313–341, 2008.

[DPT07] Giovanni Denaro, Mauro Pezzè, and Davide Tosi. Designing self-adaptive service-oriented applications. In *ICAC*, page 16. IEEE Computer Society, 2007.

[DvLF93a] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[DvLF93b] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.

[FF95] S. Fickas and M. S. Feather. Requirements monitoring in dynamic environments. In *RE '95: Proceedings of the Second IEEE International Symposium on Requirements Engineering*, page 140, Washington, DC, USA, 1995. IEEE Computer Society.

[FFLP98] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *IWSSD '98: Proceedings of the 9th international workshop on Software specification and design*, page 50, Washington, DC, USA, 1998. IEEE Computer Society.

[FFVLP98] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *IWSSD '98: Proceedings of the 9th international workshop on Software specification and design*, page 50, Washington, DC, USA, 1998. IEEE Computer Society.

[FHS$^+$06] Jacqueline Floch, Svein O. Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjørven. Using architecture models for runtime adaptability. *IEEE Software*, 23(2):62–70, 2006.

[FS01] Anthony Finkelstein and Andrea Savigni. A framework for requirements engineering for context-aware services. In *In Proc. of 1st International Workshop From Software Requirements to Architectures (STRAW 01)*, pages 200–1, 2001.

[GEL$^+$06] Eli Gjørven, Frank Eliassen, Ketil Lund, Viktor S. Wold Eide, and Richard Staehli. Self-adaptive systems: A middleware managed approach. In Alexander Keller and Jean-Philippe Martin-Flatin, editors, *SelfMan*, volume 3996 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2006.

[GMNS03] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models. *J. Data Semantics*, 1:1–20, 2003.

[HED93] S. D. P. Harker, K. D. Eason, and J. E. Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *Proc. Int. Symp. Req. Eng.*, pages 266–272, 1993.

[HFS04] Svein O. Hallsteinsen, Jacqueline Floch, and Erlend Stav. A middleware centric approach to building self-adapting systems. In Thomas Gschwind and Cecilia Mascolo, editors, *SEM*, volume 3437 of *Lecture Notes in Computer Science*, pages 107–122. Springer, 2004.

[HSSF06] Svein O. Hallsteinsen, Erlend Stav, Arnor Solberg, and Jacqueline Floch. Using product line techniques to build adaptive systems. In *SPLC*, pages 141–150. IEEE Computer Society, 2006.

[IMW07] Florian Irmert, Meyerhöfer, and Markus Weiten. Towards runtime adaptation in a soa environment. In *Proceedings of (ECOOP'2007) Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'07)*, pages 17–26, Berlin, Germany, July 2007.

[JBEM10a] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos. Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling. In *18th IEEE Int. Requirements Eng. Conf.*, 2010.

[JBEM10b] Ivan J. Jureta, Alex Borgida, Neil A. Ernst, and John Mylopoulos. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *18th IEEE Int. Requirements Eng. Conf.*, pages 115–124, Sydney, Australia, 2010.

[JBEM10c] Ivan J Jureta, Alex Borgida, Neil A Ernst, and John Mylopoulos. Techne: Towards a New Generation of Requirements

Modeling Languages with Goals, Preferences, and Inconsistency Handling. In *International Conference on Requirements Engineering (RE)*, Sydney, Australia, September 2010.

[JMF08] I. J. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *16th IEEE Int. Requirements Eng. Conf.*, pages 71–80, 2008.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[KM07] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. *Future of Software Engineering, 2007. FOSE '07*, pages 259–268, May 2007.

[LLW$^+$05] Sotirios Liaskos, Alexei Lapouchnian, Yiqiao Wang, Yijun Yu, and Steve M. Easterbrook. Configuring common personal software: a requirements-driven approach. In *13th IEEE International Conference on Requirements Engineering, (RE'05), Paris, France*, pages 9–18, 2005.

[LLY$^+$06] Sotirios Liaskos, Alexei Lapouchnian, Yijun Yu, Eric Yu, and John Mylopoulos. On goal-based variability acquisition and analysis. In *14th IEEE Int. Requirements Eng. Conf.*, pages 79 –88, 2006.

[LMM10a] S. Liaskos, S. McIlraith, and J. Mylopoulos. Goal-based Preference Specification for Requirements Engineering. In *18th IEEE Int. Requirements Engineering Conf.*, 2010.

[LMM10b] Sotirios Liaskos, Sheila A. McIlraith, and John Mylopoulos. Integrating preferences into goal models for requirements engineering. In *18th IEEE Int. Requirements Eng. Conf.*, pages 135–144, Sydney, Australia, 2010.

[LSSZ10] Chiara Leonardi, Luca Sabatucci, Angelo Susi, and Massimo Zancanaro. Ahabs leg: Exploring the issues of communicating semi-formal requirements to the final users. In Barbara Pernici, editor, *Advanced Information Systems Engineering*, volume 6051 of *LNCS*, pages 455–469. Springer, 2010.

[McC93] John McCarthy. Notes on formalizing context. In *Proceedings of the 13th international joint conference on Artifical intelligence*, volume 1, pages 555–560, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[MHM09] Daniel Laurence Moody, Patrick Heymans, and Raimundas Matulevicius. Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax. In *17th IEEE Int. Requirements Eng. Conf.*, pages 171–180, 2009.

[MNF$^+$10] Alessandro Marchetto, Cu D. Nguyen, Chiara Di Francescomarino, Nauman A. Qureshi, Anna Perini, and Paolo Tonella. A design methodology for real services. In *PESOS '10: Proceedings of the 2010 ICSE Workshop on Principles of Engineering Service Oriented Systems*, 2010.

[MPP08] Mirko Morandini, Loris Penserini, and Anna Perini. Towards goal-oriented development of self-adaptive systems. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'08)*, pages 9–16, New York, NY, USA, 2008. ACM.

[NE00] Bashar Nuseibeh and Steve M. Easterbrook. Requirements engineering: a roadmap. In *ICSE - Future of SE Track*, pages 35–46, 2000.

[Nie04] L. Nielsen. Engaging personas and narrative scenarios. *A study how a user-centered approach influenced the perception of the design process in the e-business group at AstraZeneca. Copenhagen Business School Editor, Fredriksberg, Denmark*, 2004.

[OFMA08] Marc Oriol, Xavier Franch, Jordi Marco, and David Ameller. Monitoring adaptable soa-systems using salmon. In *Workshop on Service Monitoring, Adaptation and Beyond (Mona+)*, pages 19–28, 2008.

[OGT$^+$99] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.

[OQF$^+$11] Marc Oriol, Nauman A. Qureshi, Xavier Franch, Anna Perini, and Jordi Marco. Requirements monitoring for adaptive service-based applications, 2011.

[OR04] Andrea Omicini and Giovanni Rimassa. Towards seamless agent middleware. In *WETICE*, pages 417–422. IEEE Computer Society, 2004.

[Pap08] Mike P. Papazoglou. The challenges of service evolution. In Zohra Bellahsene and Michel Léonard, editors, *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[Per09] Anna Perini. *Wiley Encyclopedia of Computer Science and Engineering*, chapter Agent-Oriented Software Engineering. Hoboken: John Wiley & Sons, Inc., January 2009. dx.doi.org/10.1002/9780470050118.ecse006.

[PPSM07a] Loris Penserini, Anna Perini, Angelo Susi, and John Mylopoulos. High variability design for software agents: Extending Tropos. *TAAS*, 2(4), 2007.

[PPSM07b] Loris Penserini, Anna Perini, Angelo Susi, and John Mylopoulos. High variability design for software agents: Extending Tropos. *TAAS*, 2(4), 2007.

[PQS$^+$11] Anna Perini, Nauman A. Qureshi, Luca Sabatucci, Alberto Siena, and Angelo Susi. Evolving requirements in socio-technical systems: Concepts and practice. In *Conceptual Modeling (ER'11)*, 2011.

[QJP11] Nauman A. Qureshi, Ivan Jureta, and Anna Perini. Requirements engineering for self-adaptive systems: Core ontology and problem statement. In *23rd Intl. Conf. on Advanced Information Systems Engineering (CAiSE'11)*, volume 6741 of *LNCS*, pages 33–47. Springer, 2011.

[QLP11] Nauman A Qureshi, Sotirios Liaskos, and Anna Perini. Reasoning about adaptive requirements for self-adaptive systems at runtime. In *Second Int. Workshop on Requirements@Run.Time (RE@RunTime), held at (RE'11)*, Trento, Italy, 2011.

[QMNP10] Nauman A. Qureshi, Mirko Morandini, Cu D. Nguyen, and Anna Perini. Tool-supported development process for adaptive systems, 2010. SE Research Group Technical Report (TR-FBK-

SE-2010-2), FBK, Trento, Italy. http://se.fbk.eu/files/TR-FBK-SE-2010-2.pdf.

[QP08] Nauman A. Qureshi and Anna Perini. Towards seamless adaptation: An agent-oriented approach. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 471–472, Washington, DC, USA, 2008. IEEE CS.

[QP10] Nauman A. Qureshi and Anna Perini. Requirements engineering for adaptive service based applications. In *18th IEEE Int. Requirements Eng. Conf.*, pages 108–111, Sydney, Australia, 2010.

[QPEM10] Nauman A Qureshi, Anna Perini, Neil A Ernst, and John Mylopoulos. Towards a continuous requirements engineering framework for self-adaptive systems. In *First Int. Workshop on Requirements@Run.Time (RE@RunTime), held at (RE'10)*, pages 9–16, Sydney, Australia, 2010.

[QSP11] Nauman A. Qureshi, Norbert Seyff, and Anna Perini. Satisfying user needs at the right time and in the right place: a research preview. In *Proceedings of the 17th international working conference on Requirements engineering: foundation for software quality*, REFSQ'11, pages 94–99, Berlin, Heidelberg, 2011. Springer-Verlag.

[Rob09] William Robinson. A Roadmap for Comprehensive Requirements Monitoring. *Computer*, 43(5):64–72, 2009.

[SBH$^+$07] Pete Sawyer, Nelly Bencomo, Danny Hughes, Paul Grace, Heather J. Goldsby, and Betty H. C. Cheng. Visualizing the analysis of dynamically adaptive systems using i* and dsls. In *REV '07: Proceedings of the Second International Workshop on Requirements Engineering Visualization*, page 3, Washington, DC, USA, 2007. IEEE Computer Society.

[SBW$^+$10] Pete Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-aware systems a research agenda for re for self-adaptive systems. In *18th IEEE Int. Requirements Eng. Conf.*, pages 95–103, Sydney, Australia, 2010.

[SFRG08] Giovanna Di Marzo Serugendo, John Fitzgerald, Alexander Romanovsky, and Nicolas Guelfi. A generic framework for the engineering of self-adaptive and self-organising systems. In *Organic Computing - Controlled Self-organization*, number 08141 in Dagstuhl Seminar Proceedings. Schloss Dagstuhl, Germany, 2008.

[SGM10] N. Seyff, F. Graf, and N. Maiden. Using mobile re tools to give end-users their own voice. In *International Conference on Requirements Engineering (RE)*, pages 37–46, Sydney, Australia, 2010.

[SKS07] Masayuki Shibaoka, Haruhiko Kaiya, and Motoshi Saeki. Goore: Goal-oriented and ontology driven requirements elicitation method. In *Advances in Conceptual Modeling - Foundations and Applications*, pages 225–234, Auckland, New Zealand, Nov. 2007. Springer. LNCS 4802.

[SLRM10] Vitor E. S. Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness requirements for adaptive systems, 2010. Technical Report DISI-10-049, DISI, Universit‘a di Trento, Italy.

[SMPS09] A. Siena, J. Mylopoulos, A. Perini, and A. Susi. Designing law-compliant software requirements. In *Conceptual Modeling (ER’09)*, pages 472–486, 2009.

[SR07] Jan Sudeikat and Wolfgang Renz. Toward requirements engineering for self - organizing multi- agent systems. In *SASO*, pages 299–302. IEEE Computer Society, 2007.

[SS07] Wassiou Sitou and Bernd Spanfelner. Towards requirements engineering for context adaptive systems. In *COMPSAC ’07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, pages 593–600, Washington, DC, USA, 2007. IEEE Computer Society.

[ST09] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:14:1–14:42, May 2009.

[SYN07a] M. Salifu, Yijun Yu, and B. Nuseibeh. Specifying monitoring and switching problems in context. *15th IEEE International Requirements Engineering Conference (RE ’07)*, pages 211–220, Oct. 2007.

[SYN07b] M. Salifu, Yijun Yu, and B. Nuseibeh. Specifying monitoring and switching problems in context. In *15th IEEE Int. Requirements Eng. Conf.*, pages 211–220, 2007.

[TCW$^+$04] Gerald Tesauro, David M. Chess, William E. Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O. Kephart, and Steve R.

White. A multi-agent systems approach to autonomic computing. In *AAMAS*, pages 464–471. IEEE Computer Society, 2004.

[TJWW07] W.T. Tsai, Z. Jin, P. Wang, and B. Wu. Requirement engineering in service-oriented system engineering. In *e-Business Engineering, 2007. ICEBE 2007. IEEE International Conference on*, pages 661–668, Oct. 2007.

[vL01a] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. 5th IEEE Int. Symposium on Requirements Eng.*, page 249. IEEE Computer Society, 2001.

[vL01b] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *5th IEEE International Symposium on Requirements Engineering*, page 249, 2001.

[WSB+10] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty Cheng, and Jean-Michel Bruel. Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15:177–196, 2010.

[Yu95] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.

[Yu97] E. Yu. Towards modeling and reasoning support for early requirements engineering. In *Proc. 3rd IEEE Int. Symposium on Requirements Eng.*, pages 226–235, 1997.

[ZJ97a] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM T. Softw. Eng. Methodol.*, 6(1):1–30, 1997.

[ZJ97b] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.

[ZJW03] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.

[ZLKM08] Qin Zhu, Lin Lei, Holger M. Kienle, and Hausi A. Muller. Characterizing maintainability concerns in autonomic element design. *IEEE International Conference on Software Maintenance (ICSM 2008)*, pages 197–206, 28 2008-Oct. 4 2008.

Appendix A

Empirical Survey Material

Below we provide the material we used for the empirical survey study.

SURVEY INTRODUCTION

INTRODUCTION:

What are self-Adaptive Systems (SAS)?
Software Systems which are able to alter their behavior in response to changes in the external environment such as: change in end-user needs, variation in operational context and resource availability. Software engineering for such system is complex. In particular, requirements engineering (RE) for SAS requires attention to specify requirements and perform analysis over them.

Example:
Consider music download software (e.g. a juke box). It accepts a variety of users requests with different parameters (bandwidth, format) and preferences about the choice of music (e.g. rock, instrumental etc.). The software manages the incoming request by being continuously available thereby balancing the load with respect to its scalability i.e. by adding more memory and disk space from a pool of servers.

Objective of the Survey:
The aim of this survey is to solicit feedback on the need of having a visual requirements modeling language to model requirements of self-adaptive systems.

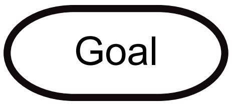
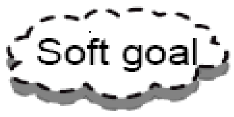
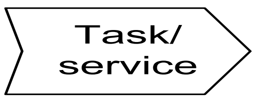
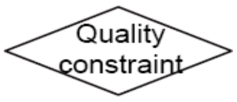
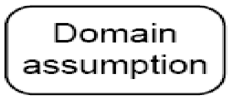
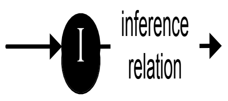
We ask you to do the following tasks:

1- **Modeling Task**: You are asked to play the role of an analyst.
2- **Reasoning Task**: You are asked to play the role of a running SAS.
3- **Visual Notation Analysis Task**: You are required to analyze the visual notations based on your experience.
4- **Final Feedback**:

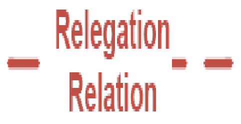
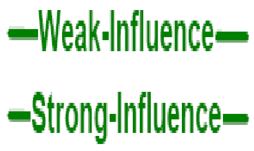
VISUAL NOTATION GUIDE

ADAPTIVE REQURIEMENTS MODELING LANGUAGE (ARML)

<u>Quick guide to notations:</u>

Visual Notation	Concepts & Relations	e.g.
	**Definition:** A **Goal** represents a desired state of affairs, the achievement of which can be measured and is definitively concluded. **Example:** *"Meeting to be Scheduled"*	**See** **Fig.1,2,3** **on Page 7**
	**Definition:** A **Soft goal** represents a desired state of affairs, the achievement of which can only be estimated, not definitively concluded. **Example:** *"Convenience"*, *"Easy"*	**See** **Fig.7** **on Page 7**
	**Definition:** A **Task** corresponds to an activity or an action whose achievement leads to the definitive conclusion of its means. A service can operationalize task. **Example:** *"Download music"*, *"Show song listed as most viewed"*	**See** **Fig.4,5,6** **on Page 7**
	**Definition:** A **Quality constraint** is a desired value of non-binary measurable properties of the system-to-be that constrains a goal or a soft goal. **Example:** *"Music download speed must not be less than 128kbps/sec"*	**See** **Fig.8** **on Page 7**
	**Definition:** A **Domain assumption** is a condition within which the system-to-be will be performing tasks in order to achieve the goals, quality constraints, and satisfy as best as feasible the soft goals. **Example:** *"If the user has paid the subscription she can download the music from the online database"*	**See** **Fig.4,5,8** **on Page 7**
	**Definition:** An **<Inference> relation** stands between a requirement that is the immediate consequence of another set of requirements, the former is called the conclusion, the latter the premises. Alternatively, inference relation can be used to connect the refined requirement to the requirements that refine it. **Possible Operationalization:** AND – Inference (Conjunctions) **Fig. 1** OR – Inference (Disjunctions) **Fig. 2** Simple – Inference (Refinement) **Fig. 3** Means-End (as in case of i*/Tropos) **Fig. 3, 6** **Example:** *"Req1: Generate revenue from the audio player"* has <inference>	**See** **Fig. ALL** **on Page 7**

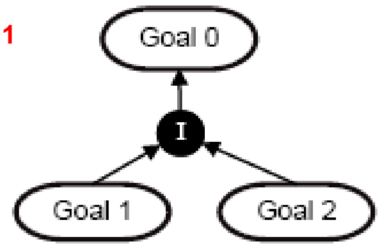
	relation with two requirements: *“Req1.1: Music is available to subscribers”*, *“Req1.2: Display ads in the player”*.	
- - -Conflict- - -	**Definition:** A **<Conflict> relation** stands between all members (two or more) of a minimally inconsistent set of requirements. **Example:** *“Req1: Music is available to subscribers”* is in <Conflict> with *“Req2: Music is available to users”* *Rationale:* Since it is not possible to maintain the player free to all users and make music available to subscribers only.	**See** **Fig.8** **on Page 7**
- - -pref- - ▷▷	**Definition:** A <**Preference**> is a binary relation that exists between two requirements and it defines the stakeholder evaluations of requirements that determine the desirability of a requirement. **Possible Operationalization:** Ranking/Priorities (0---10) Fuzzy Scale (0.1 ---- 1.0) **Example:** *“The bitrate of music delivered via the online audio player should be at least 256kb/s”* is <Preferred> over *“the bitrate of music delivered via the online audio player should be at least 128kb/s”*	**See** **Fig.9** **on Page 7**
O Is-Optional	**Definition:** An <**is-Optional> relation** is unary that states the evaluation of stakeholder of requirement, which may be desirable. Functional requirements, which are “nice to have”. **Operationalization:** Boolean (T/F) (Yes/No) **Example:** *“Color printing of a meeting schedule”* <is-Optional>.	**See** **Fig.9** **on Page 7**
M Is-Mandatory	**Definition:** An <**is-Mandatory> relation** is unary that states the evaluation of stakeholder of requirement, which must be satisfied. Functional requirements. **Operationalization:** Boolean (T/F) (Yes/No) **Example:** *“Each Participant must have meeting schedule available”* <is-Mandatory>.	**See** **Fig.10** **on Page 7**
_ _Association Link _	**Definition:** An <**Association> link** is used to define a link between two elements. **Example:** *“High level Context (e.g. Outdoor)”* is <associated> to *“an ontology concept (e.g. place)”*.	**See** **Fig.11** **on Page 7**

	**Definition:** A <**Relegation> relation** is n-array relation that stands between one or more requirements, to relax or to suspend conditions imposed over them. A mandatory requirement can have a <relegation> relation with an optional requirement. **Operationalization:** Chaining Rules Fuzzy Temporal Branching Logic functions (e.g. as in case of Relax Language) Relegation function: responsible to relax the conditions of the constraints over the requirements or to suspend it. **Example:** "*Req1: to download music*" has <Relegation> relation with the "*Req2: to Stream the Song online*". Rationale: Since download music is for subscribers and to reduce load on the server it can relax streaming of songs.	**See** **Fig.9** **on Page 7**
	**Definition:** An <**Influence> relation** is said to exist between a set of requirements, where satisfaction of one requirement warrants the satisfaction of the other. This determines the satisfaction of the requirements set. There are two types, weak-influence (where partial satisfaction is possible) and strong-influence (where there is no possibility to satisfy the latter requirement). **Possible Operationalizations:** Priority (Quantitative Rankings) Cost (possibly with a cost function) Probabilistic (using Statistical method) Contribution links (++,--, -, +) as in case of i* / Tropos **Example:** "*Req: to subscribe and make payment*" have <Strong-Influence> over the "*Req to download the music*". "*Req: subscribe and make payment*" have <weak-Influence> over the "*Req: to listen music online*"	**See** **Fig.10** **on Page 7**
	Definition: A **Resource** is an entity either tangible or intangible referred to by one or more instances of the information communicated during elicitation by the stakeholder. **Classification:** Tangible Intangible **Example:** Tangible Resource: "*Physical e.g. Mobile phone*" Intangible Resource: "*Virtual or Data e.g. Bank Balance, Agenda*"	**See** **Fig.12** **on Page 7**

$\xrightarrow{\text{Requires}}$	**Definition:** A <**Requires**> relation is a binary relation that exists between a task and a resource. **Possible Operationalization:** Resource Locator Function that determines all the required resources needed by the tasks in a particular context. **Example:** *"Task: Download song"* <requires> *"Resource: internet connection"*	**See** **Fig.12** **on Page 7**
C_1 [Context]	**Definition:** A **Context** is defined as a set of information (condition) that is presupposed (or believed to be true) by the stakeholders to hold when they communicate particular requirements. **Possible Operationalization:** Class Diagrams Conditions States Explicit specification of the states and conditions represented in the requirements model to be monitored. Contextualization Function: which tells the specific states or conditions to be true, which describes a situation in an environment **Classification:** Information related to particular user's and system's environment states, conditions or assumptions **Example:** *"System states (e.g. searching a song)", "User states (e.g. Listening to music)", "User Location (e.g. at home)", "Device Status (e.g. Battery is low)"*	**See** **Fig.11** **on Page 7**
@{ Ontology Concept }	**Definition:** An **Ontology Concept** defines an entity and its characteristics or essential features in a particular domain of discourse. **Example:** *"Frame rate in Music Ontology"*	**See** **Fig.11** **on Page 7**

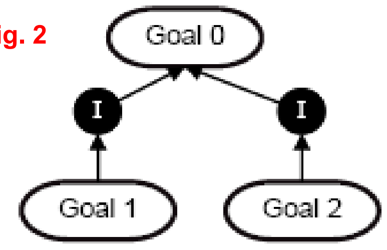
Requirement Refinement Patterns in ARML and Use of Visual Notation

Fig. 1



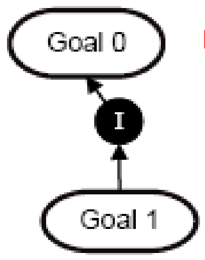
AND-Inference

Fig. 2



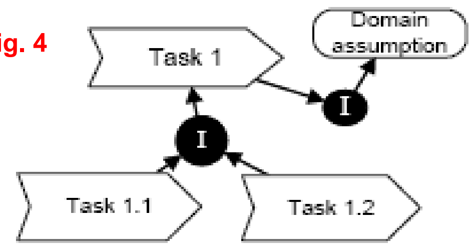
(High Variability) OR-Inference

Fig. 3



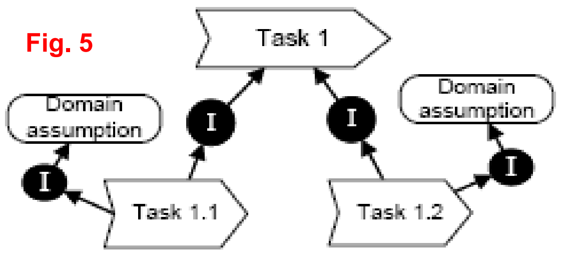
Simple-Inference

Fig. 4



AND-Inference

Fig. 5



(High Variability) OR-Inference

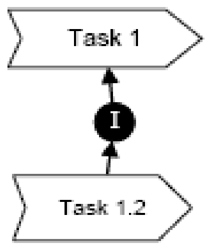
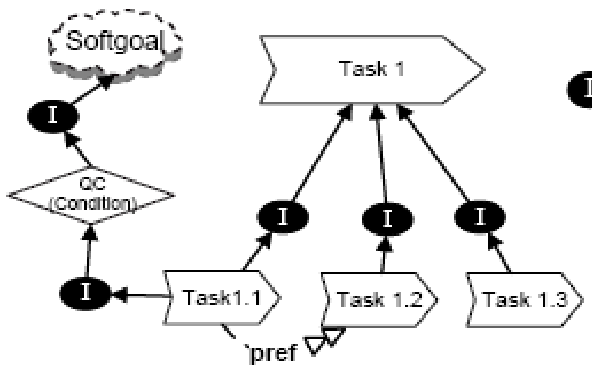


Fig. 6

Simple-Inference

Fig. 7



Preferred Task w.r.t Quality Constraint (QC) and Soft goal (SG)

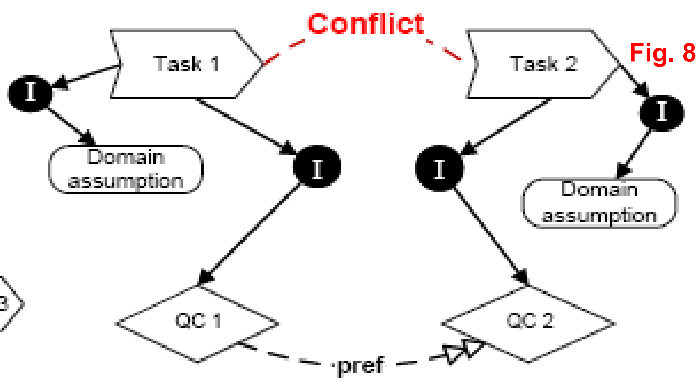


Fig. 8

Conflicting Tasks (Inconsistent Tasks)

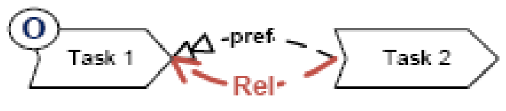
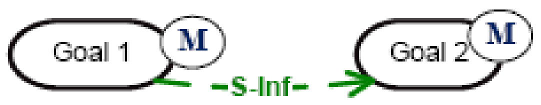


Fig. 9

Relegation Relation



Influence Relation

Fig. 10

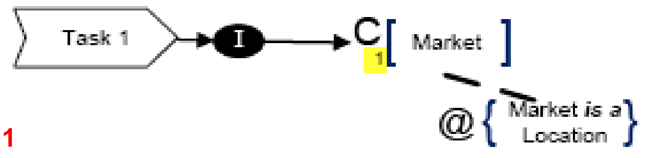


Fig. 11

Task inferring Context which is defined in an Ontology

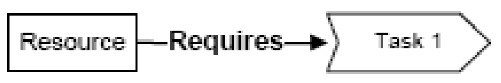
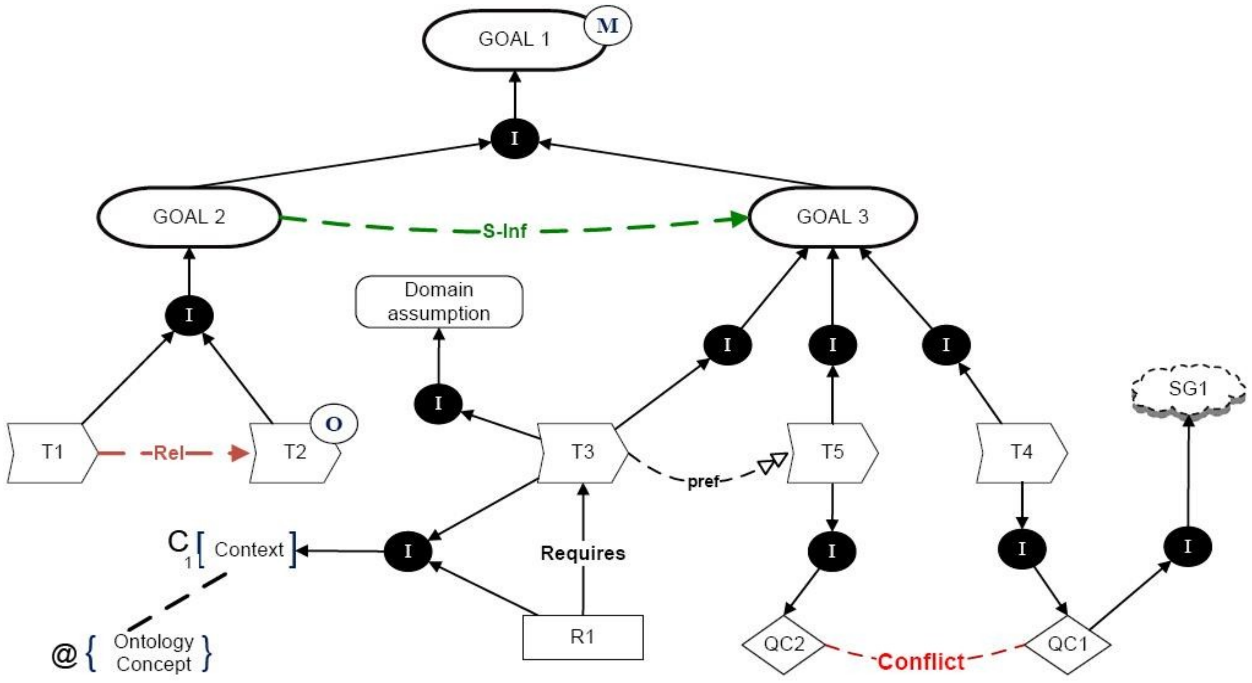


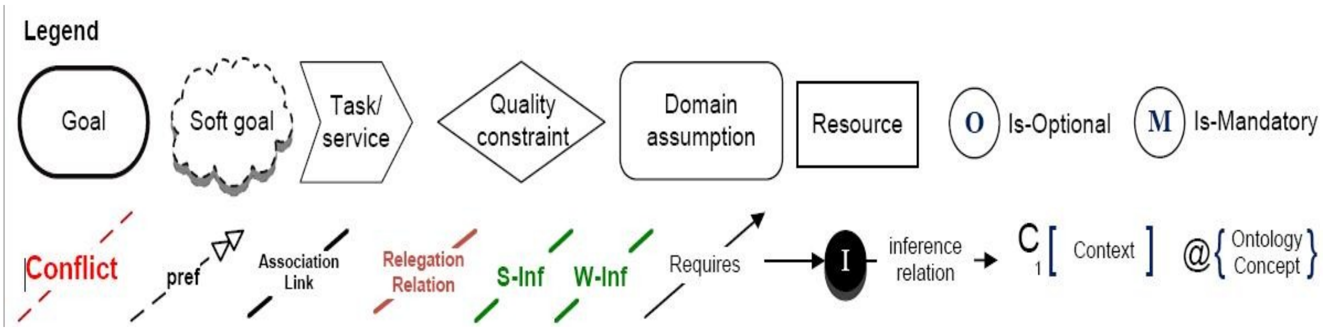
Fig. 12

Requires Relation

A graph of modeling notations (overall picture):



Legend:



PRE-QUESTIONNAIRE

Please State your Previous Experience:

Software modeling experience in general (specify in years)	_____
Software requirements modeling using any goal modeling language (specify in years)	_____

MATERIAL & TASKS

FOR

MODELING

Modeling Task: You are asked to play the role of an analyst:

a. **TASK # 1:**

   i. Answer the Questions
      - Read the given requirements of “iComp” system
      - Read the given partial Model of “iComp” system in ARML

b. **TASK # 2:**

   i. Change the given model as per the given requirement

   ii. Answer the questions
      - Read the given partial Model of “iComp” system in ARML

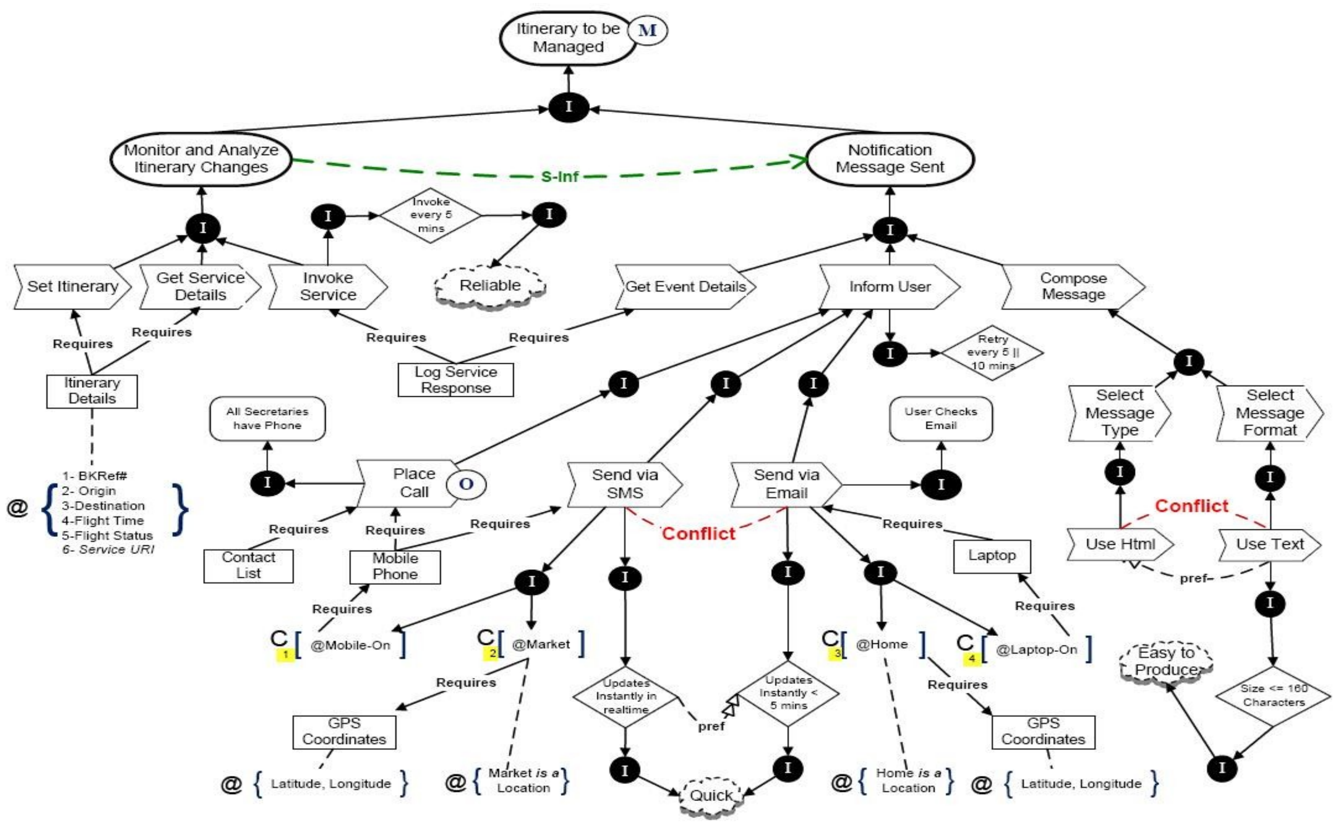
c. **Feedback:** Post-Modeling Questionnaire

Modeling Task:

iComp: System Requirements:

An "*Intelligent Travel Companion*" (hereafter, iComp) is a self-adaptive system. It manages end-user's travel tasks. It supports the end-user to specify their preferences and receive timely information about their booking confirmation (e.g., confirmed, canceled, in progress). The booking preference is collected using the iComp GUI. Once end-user has specified her travel itinerary (e.g. booking reference number (BKRef#), time of the flight, origin, destination and service source etc.), iComp starts to monitor the relevant service for checking the flight status. The notification message about the flight status must be sent to the end-user as per her preferences i.e. via Email, or Call, or SMS (i.e. most preferred method) instantly (i.e. in less than an hour before the flight) on her device (i.e. laptop or mobile) depending the user context (i.e. home or market or office etc.) and system context (e.g. the Internet is available, service is available etc.). The notification message must be sent to end-users device by selecting a suitable message format (i.e. size, scaling, format) depending as per her device context (e.g. smart phone). In case there are some problems (i.e. user is not accessible, network is not available, or device is not reachable), the notification message must be ensured as sent by adopting an alternative method e.g. sending to alternative contact (if given) e.g. office secretary, or apply a retry strategy (e.g. attempt to resend the message after every 5 minutes than 10 minutes etc.) until the message is confirmed to be delivered.

Partial Model of "iComp" in ARML:



A – Task # 1: (Act as an Analyst)

Start time _________ **End time** __________

See the model of "iComp" given on the previous Page (page 14).

Answer the following questions:

Which resources are used to quickly notify the user in case of flight status is ("Cancelled")?

Resources: ____

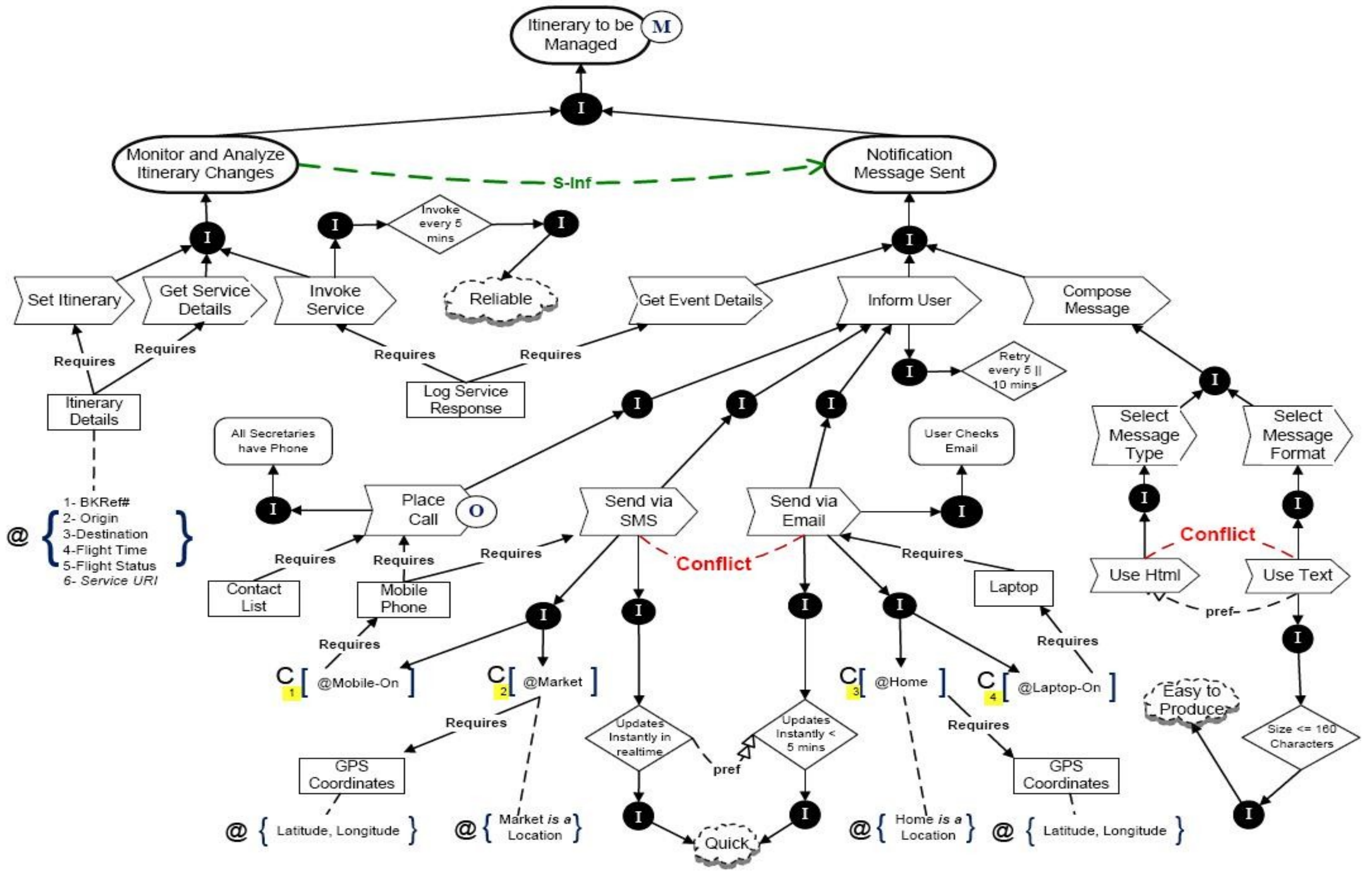
Which tasks are involved in notifying the user when the flight status is changed ("Cancelled")?

Tasks: ____

In which context the user is notified more quickly about the change in flight status?

Contexts: ____

Partial Model of "iComp" in ARML:



B – Task # 2: (Act as an Analyst)

Start time _________ **End time** __________

Try to change the given model as per the following requirement. (See page 16)

Requirement: “User prefers to be notified via Tweeter service.”

HINT: Use the visual modeling notations guide if needed (on Page4-7).

Answer the following questions:

Which task is influenced by adding the new task to “send via tweeter”?

Influence Relation among tasks: ____

What are the resources added by you?

Resources: ____

In which context the notification method “send via tweeter” will be more preferred?

Contexts: ____

What are the new ontology concepts you added?

Ontology Concepts: ____

C – Feedback: Post-Modeling Questionnaire:

According to your modeling experience (Task: A, B). Answer the following questions:

Start time _________ **End time** __________

SCALE: **1 – Strongly agrees, 2 – Agree, 3 – Not certain, 4 – Disagree, 5 – Strongly disagree**

	1	2	3	4	5
Modeling task is clear					
Modeling in ARML is difficult					
Using the visual notations is difficult					
Visual notations are adequate to model the requirements of SAS					
It is useful to explicitly model context, resource, ontology concepts					
It is useful to model influence, relegation and requires relations among modeling elements					

General Comments (If any):

MATERIAL & TASKS

FOR

REASONING

Reasoning Task: You are asked to play the role of a running SAS.

a. **TASK # 1:**

   i. Answer the question (descriptive)
      - Read the given scenario

   ii. Answer the questions(multiple-choice)
      - Read the given scenario

b. **Feedback:** Post-Reasoning Questionnaire

A – Task # 1: (Act as a running SAS)

Read the following scenario; use the “iComp” model on page (16), if required.

Start time _________   **End time** __________

<u>**Scenario** [Flight Changed]:</u>

If the flight is “Cancelled” and there is no option to fly the same day; *iComp* must reason and adapt to the candidate solution to inform the user as soon as possible.

(i) Answer the following question:

What elements from the requirements model are required by *iComp* to reason in such a situation? (Consider yourself as *iComp*, and textually describe the reasoning to be performed).

(ii) Answer the following question:

Start time _________ **End time** __________

Read the above scenario [Flight Changed], and select the most relevant option to the following question.

Which option describes reasoning by *iComp* about a particular concept? (select multiple, if applies)

1. Context (Answer :_______)
   a. User Context is: Outdoor (e.g. User is at the airport)
   b. System Context is: Checking Preference and Services to Notify
   c. Communication services are limited (e.g. Internet is slow at airport)
   d. User Context is: Indoor (e.g. User is at home)
   e. System Context is: Service invocation failed.

2. Resource (Answer :_______)
   a. Wifi
   b. Itinerary
   c. Mobile Device
   d. GPS coordinates
   e. Laptop

3. Influence Relation (Answer :_______)
   a. Service invocation task has <weak-influence> Send via SMS task
   b. Get user context task has <strong-influence> on inform user task
   c. Preferred message format has <strong-influence> on inform user task
   d. Read log task has <weak-influence> on send via email task.

4. Relegation Relation (Answer :_______)
   a. Send via SMS task <relegate> compose message task
   b. Inform user task <relegate> get user context task
   c. Get system context task <relegate> get user context task

Any other concepts you think are relevant for reasoning (please specify):

B – Feedback: Post-Reasoning Questionnaire:

According to your reasoning experience (Task A (i), A (ii)), answer the following questions:

Start time _________ **End time** __________

SCALE: **1 – Strongly agrees, 2 – Agree, 3 – Not certain, 4 – Disagree, 5 – Strongly disagree**

	1	2	3	4	5
It is difficult by SAS to reason on concepts and relation depicted in the model					
Concepts and relations are sufficient enough to reason on by SAS					
Concepts and relations are relevant for reasoning by SAS					
It is useful for SAS to reason on context, resource, ontology concepts					
It is useful for SAS to reason on influence, relegation and requires relations					

General Comments (If any):

TASK

FOR

VISUAL NOTATIONS ANALYSIS

TASK # 3: Visual Notations Analysis Task:

Read above the quick guide to the notations (on page 4-7). After reading the definitions and the associated notation, you are required to answer the following questions.
(Act as a novice user to analyze the visual notations to judge their effectiveness)

Start time _________ **End time** __________

SCALE: **1 – Strongly agrees, 2 – Agree, 3 – Not certain, 4 – Disagree, 5 – Strongly disagree**

	1	2	3	4	5
The Visual notation represents the concept they refer					

State your opinion: Why is it so? [Optional]

Visual notations are visually distinct from one and other					

State your opinion: If you think they are not distinct, why? [Optional]

Visual notations help in expressing the intended meaning					

State your opinion: If not, give reason, Why? [Optional]

The visual notations are of ample visual expressiveness (e.g. color, size and dimensions)					

State your opinion: Why visual notations are not visually expressive? [Optional]

Visual notations proposed are not overloaded (i.e. used to represent multiple concepts)					

State your opinion: If you think they are overloaded, please give reason, why? [Optional]

The visual notations proposed are easy to remember					

State your opinion: If not, please give reason, why? [Optional]

The proposed visual notations are helpful in modeling requirements of SAS					

State your opinion: If not helpful, Why? [Optional]

The visual notations does not increase too much the complexity of the model					

State your opinion: If you think complexity of the model increases, give reason, why? [Optional]

The visual notations are easy to use					

State your opinion: Which of the proposed visual notation is not easy to use? [Optional]

Post-visual notation analysis questions:

How much % of the time did you approximately spend in?

Reading the ARML concepts and relations	______%
Understanding the use of visual notations	______%
Reasoning on answering the questions	______%

General Comments (If any):

FINAL FEEDBACK QUESTIONNAIRE

Final Feedback Questionnaire:

Start time _________ **End time** __________

SCALE: **1 – Strongly agrees, 2 – Agree, 3 – Not certain, 4 – Disagree, 5 – Strongly disagree**

	1	2	3	4	5
The modeling guidelines are clear enough to me					
Concepts and relations in ARML are useful enough to model requirements of SAS					
Concepts and relations in ARML are adequate enough to model requirements of SAS					
I had no problem in modeling Context and Resources using ARML notations					
I had no problems in modeling Influence and Relegation relations using ARML notations					
The modeling effort to change the model is high					
The reasoning effort using in practice is high					
Concepts like context and resources are useful in modeling requirements of SAS					
The obtained model is useful enough to guide the running SAS to reason about execution situations					
Visual notation are effective to model the requirements for SAS					

Your Experience with ARML (Cross out the relevant**):**

Time was enough to perform the required tasks	YES ( ) / NO ( )
I understand what I have to do and guidelines were clear enough	YES ( ) / NO ( )