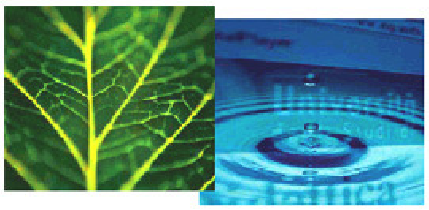
PhD Dissertation



International Doctorate School in Information and Communication Technologies

DISI - University of Trento

MANAGING THE UNCERTAINTY OF THE EVOLUTION OF REQUIREMENTS MODELS

Le Minh Sang Tran

Advisor
Fabio Massacci, Professor
Università degli Studi di Trento

Committee
John Mylopoulos, Professor
Università degli Studi di Trento

Haralambos Mouratidis, Professor
University of Brighton

Ketil Stølen, Professor
University of Oslo, SINTEF ICT

2013

Abstract

"...There are known unknowns: that is to say,
there are things that we now know we don't know..."

Donald Rumsfeld, United States Secretary of Defense

EVOLUTION is an inevitable phenomenon during the life time of a long-lived software systems due to the dynamic of their working environment. Software systems thus need to evolve to meet the changing demands. A key point of evolution is its uncertainty since it refers to potential *future* changes to software artifacts such as requirements models. Thus, the selection of *evolution-resilient* design alternatives for the systems is a significant challenge.

This dissertation proposes a framework for modeling evolution and reasoning about it and its uncertainty in requirements models to facilitate the decision making process. The framework provides *evolution rules* as a means to capture requirements evolution, and a set of *evolution metrics* to quantify design alternatives of the system. This enables more useful information about to what extent design alternatives could resist to evolution. Thus, it helps decision makers to make strategic moves.

Both evolution rules and evolution metrics are backed up with a formal model, which is based on a game-theoretic interpretation, so that it allows a formal semantics understanding of the meaning of the metrics in different scenarios.

The proposed framework is supported by a series of algorithms, which automates the calculation of metrics, and a proof-of-concept Computer Aided Software Engineering (CASE) tool. The algorithms calculate metric values for each design alternative, and enumerate possible design alternatives with the best metric values, *i.e.,* winner alternatives. The algorithms have been designed to incrementally react to every single change made to requirements models in an efficient way.

The proposed framework is evaluated in a series of empirical studies that took place over a year to evaluate the modeling part of the framework. The evaluation studies used scenarios taken from industrial projects in the Air Traffic Management (ATM) domain. The studies involve different types of participants with different expertise in the framework and the do-

main. The results from the studies show that the modeling approach is effective in capturing the evolution of complex systems. It is reasonably possible for people, if they are supplied with appropriate knowledge (*i.e.,* knowledge of method for domain experts, knowledge of domain for method experts, and knowledge of both domain and method for novices), to build significantly large models, and identify possible ways for these models to evolve. Moreover, the studies show that obviously there is a difference between domain experts, method experts, and students on the “baseline" (initial) model, but when it comes to model the changes with evolution rules, there is no significant difference.

The proposed framework is not only applicable to requirements model, but also other system models like risk assessment. The framework has been adapted to deal with evolving risks in long-lived software systems at a high level of abstraction. It thus could work with many existing risk-assessment methods.

In summary, the contribution of this dissertation to the early phase of system development should allow system designers to improve the evolution resilience of long-lived systems.

Acknowledgements

I owe my great thanks to many people, without whom, this dissertation could not be finished.

First and foremost, I profoundly thank my PhD supervisor, Professor Fabio Massacci. I was fortunate enough to be under his guidance. I would like to thank him for all of his advice, his patience, and his guidance over the past five years and beyond. I thank him for his countless hours providing ideas, explanation and feedback, also his listening and helping me to formulate initial ideas. I learnt from him the passion in doing research. During my graduate studies, I have received invaluable support from him to travel widely (summer schools, project meetings and conferences) to communicate with many researchers worldwide.

I am grateful to Professors John Mylopouslous for his support, invaluable comments and discussion since the very first days of my doctorate. His courses – Requirements Engineering and Conceptual Modeling – have provided me the essential background in this work.

I would like to express my gratitude to Professor Ketil Stølen, my mentor during my internship at SINTEF. I appreciate his precious guidance, his time spending as chair of PhD committee, and his encouragement for me to go running, which I now seriously practise. I also would like to thank the many people at SINTEF, particularly Bjørnar Soulhalg, who have helped me a lot too during my internship there.

I am thankful to Professor Haralambos Mouratidis, another PhD committee member. I was very impressed when I received his feedback on the draft of my dissertation. I believed that he had spent a plenty of time to read every word of my dissertation, and gave me a list of extremely helpful suggestions.

I am grateful to Dr. Federica Paci, who co-authored an article which is an essential part of my dissertation. Many thanks to other co-authors who contributed to publications, and to my colleagues and my friends at University of Trento, in particular Bruno Crispo, Jennifer Horkoff, Elda Paja, Tong Li, Katsiaryna Labunets and Luca Allodi for helpful feedback, suggestions, and encouragement. My sincere thanks also go to the staffs Luca Valenzin, Michela Angeli, Elisa Chiarani, Andrea Stenico, and Francesca Belton for helping me with administrative stuffs so that I could channel my time into my work.

I am also in debt to many MSc students who followed the course Security Engineering 2011-2012 at the University of Trento, as well as to the ATM experts and consultants who participated in the empirical studies of the modeling approach reported in this dissertation.

I would like to acknowledge in particular Valentino Meduri, Alessandra Tedeschi, Massimo Felici and Stephane Paul. Without their contribution, I would not be able to finish the evaluation of my proposed approach.

I also wish to express my thanks to anonymous reviewers whose the feedback was daunting to me at the first sight, but indeed was fruitful criticism that helped me to improve my research resulting in the publications, which are included in this dissertation.

Many thanks to all of my friends in Trento, in Viet Nam and elsewhere for the pleasure and nice time which make up my social life. I apologize for not listing your names because it would be unfair for anyone, whom I might forget to mention by mistake.

Thanks to my sweetheart Viet Hung Nguyen for his true love and endless care. I thank him for giving me advice and encouragement, and lifting the spirits up for both of us during this tough time.

Finally, my deepest thanks to my family for their unconditional love, unbounded patience and endurance through years when I was far away. Although they may not completely understand the things that I am working on, they remained supportive and proud of my attainments.

This work has received financial support from the SecureChange FP7 project and the NESSoS FP7 project funded by the European Commission, and University of Trento.

Contents

List of Tables

List of Figures

Acronyms

AHP	Analytic Hierarchical Process
AMAN	Arrival Manager
ANSP	Airspace Navigation Service Provider
ATCC	Air Traffic Control Center
ATCO	Air Traffic Controller
ATM	Air Traffic Management
ATSU	Air Traffic Service Unit
CASE	Computer Aided Software Engineering
CTS	Conflict Tools System
DMAN	Departure Manager
EC	Executive Controller
FAA	United State Federal Aviation Authority
FDPS	Flight Data Processing System
GUI	Graphical User Interface
ICT	Information and Communication Technology
PLC	Planning Controller
RBT	Reference Business Trajectory
RE	Requirements Engineering
SESAR	European Single European Sky ATM Research Initiative
SWIM	System Wide Information Management
TCC	Tactical Controller

Part I

Motivation and Background

CHAPTER 1

INTRODUCTION

"...There are known unknowns: that is to say,
there are things that we now know we don't know..."

Donald Rumsfeld, United States Secretary of Defense

This chapter presents the motivation of this dissertation. It also summarizes the major contributions of this work, as well as published/in submission publications on which the dissertation is built.

THE term *software evolution* has been introduced in 1980 by Lehman in his work[Leh80a; Leh80b], and was widely adopted since the 90s. In the domain of software systems [Pro96; Has+05; LL98; Rus+99; ZO97], evolution refers to a process of continually updating software systems in accordance to changes in their working environments such as business requirements, regulations and standards. Recent studies in software evolution attempt to understand causes, processes, and effects of the phenomenon [AP03; KS99; LaM+08]; or focus on the methods and tools that manage the effects of evolution [Sof05; Sou+11; Has+05]; or offset their effects in advance. The aim of this dissertation focuses on the last setting.

A key observation underpinning this dissertation is that while some changes (*i.e.,* evolution) are unpredictable, many others can be predicted albeit with some uncertainty because they will be the result of a process. A representative example is Air Traffic Management (ATM). The increment in both quantity and complexity of air traffic requires a better collaboration among ATM systems and actors across air spaces and nations. To address this requirement, Europe has launched several long-term projects [1] [2] and programs [3] [4], which include: introducing new cooperation regulations, ATM business processes; improving existing ATM systems (*e.g.,* Arrival Management – AMAN, Department Management – DMAN); introducing new infrastructure for information interchange (*i.e.,* System Wide Information Management – SWIM[Adm09]), and so on. The requirements of these systems may evolve in many directions and at different levels from organizational, architectural, to operational aspects. Such evolutions are known to be possible, yet unknown whether they would happen: the *known unknowns.*

Unfortunately, a company that produces or procures software for these systems cannot wait until all unknowns in the standardization [old: SWIM standardization] become known. The process of tendering and organizational restructuring requires a significant amount of time and planning. Therefore decision makers must essentially bet on the final solution and possibly minimize the risks that the solution (*i.e.,* design alternative, implementation choice) would turn out to be wrong and require last minute acquisitions. For instance, while the European Single European Sky ATM Research Initiative (SESAR) and the United State Federal Aviation Authority (FAA), in [Pro08; Fed09; Adm09], have listed a number of design alternatives for SWIM (both architectural and technical), which satisfy different high level decisions, still there is no means to support the choice of an optimal alternative that could minimize the risk of wrong local choices.

While many approaches have been proposed to perform the management or consistency checking on requirements evolution, there has been less effort on delivering an explicit modeling and reasoning framework to assist decision makers to select a good design alternative. In the realm of requirements evolution, a "good" design alternative is an evolution-resilient one. An *evolution-resilient* design alternative, in the context of this dissertation, is understood as a design alternative that has more chances to be operational even if evolution happens, or requires less modification to be operational due to evolution.

We need to capture what Loucopoulos and Kavakli [LK99] identify as the knowledge about *"what the current state is", "where the desired state to-be is in the future"*, and *"alter-*

[1]http://www.sesarju.eu/
[2]http://www.swim-suit.aero/swimsuit/
[3]http://www.eurocontrol.int/services/arrival-manager
[4]http://www.eurocontrol.int/surveillance/cascade

native designs" for the desired future state. In this respect, it is important to provide a sound quantitative analysis, which is one of the current weaknesses (see Dalal *et al.* [Dal+04] for a discussion) of many existing approaches.

As an effort to bridge the gap, this thesis proposes a framework that provides support for modeling and reasoning about the uncertainty of requirements evolution, and at the utmost importance, assisting the decision makers in the selection of a “good" design alternative with respect to evolution.

1.1 Contributions

The major contribution of this dissertation is a framework dealing with evolution in requirements models. The proposed framework includes:

- *A set of notions to model the uncertainty of requirements evolution.* We propose evolution rules, which include *observable rule* and *controllable rule*, as a mean to capture the uncertainty in requirements evolution. The former kind captures potential evolution and its uncertainty. The latter kind captures different design alternatives that fulfill requirements. These evolution rules are used to capture the evolution in requirements models at high level of abstraction. They could be adapted to different requirements engineering languages.

- *A set of quantitative metrics for reasoning on the evolution uncertainty.* With respect to evolution, we propose three quantitative metrics to assess how well a design alternative could be able to resist to evolution. In other word, we measure the probability by which the implementation of a system using the given design alternative could be operational when evolution happens. The proposed metrics are based on evolution rules. Therefore they could be also applied to abstract requirements model.

- *A series of algorithm automating the reasoning.* To make the proposed framework more feasible in practice, we develop a series of algorithms that automate the reasoning on requirements evolution. To maintain the generality of the framework, we propose hypergraph as a means to express requirements models. Based on hypergraph the algorithms enumerate possible design alternatives and calculate their metrics values. Thus it could help to facilitate the decision making process.

- *A prototype of proof-of-concept CASE tool.* We implement a prototype of a CASE tool as a proof-of-concept for the proposed framework. The CASE tool allows users to customize graphical constructs that are used to model requirements models. It also implements the proposed algorithms to perform automated reasoning on requirements evolution.

- *A modeling and reasoning approach on risk evolution.* We extend the proposed approach for requirements evolution to the realm of risk management. We propose a generic method to select cost-effective countermeasures for software risks based on risk graph. We then adapt evolution rules to cope with evolution in risk graphs, and integrate the proposed metrics to the reasoning process to select cost-effective and evolution-resilient countermeasures for risks in software systems.

1.2 Terminology

Evolution Potential changes make to software artifacts (*e.g.*, requirements models) after deployment of system-to-be.

Evolution uncertainty The uncertainty of whether an evolution actually applies to given circumstances.

Requirements evolution Potential changes make to requirements after deployment of system-to-be.

System-to-be The system as it should be when it will be built and operated.

System-to-be-next After deployment of system-to-be, new requirements or problems may arise. We may need to consider the next system versions are likely to be. The system versions beyond the system-to-be are the systems-to-be-next.

1.3 Structure of the Dissertation

The dissertation is organized into four parts. The first part focuses on the introductory basics consisting of following chapters:

- *Chapter 1: Introduction* . This chapter presents the motivation and the summarize of major contributions of this dissertation.

- *Chapter 2: Research Roadmap* . This chapter discusses the global research questions that drive the research plan for this dissertation. This chapter also summarizes the proposed framework to deal with requirements evolution, as well as how various artifacts in later chapters are fitted into the framework.

- *Chapter 3: State-of-the-Art* . This chapter discusses in detail related works in the field of requirements evolution. Particularly, it presents different approaches in requirements evolution modeling, requirements evolution management, and relevant analysis techniques, such as inconsistency checking, change impact analysis. The chapter also gives a comparison between the proposed framework and its closed approaches in the literature.

  Referred publication(s): This chapter has been partially published in:

  – Michael Felderer, Basel Katt, Philipp Kalb, Jan Jürjens, Martín Ochoa, Federica Paci, Le Minh Sang Tran, Thein Than Tun, Koen Yskout, Riccardo Scandariato, Frank Piessens, Dries Vanoverberghe, Elizabeta Fourneret, Matthias Gander, Bjørnar Solhaug, and Ruth Breu. "Evolution of Security Engineering Artifacts: A State of the Art Survey". In: *International Journal of Secure Software Engineering* (2014). To appear.

- *Chapter 4: Application Scenarios* . This chapter summarizes the two scenarios that we employ to illustrate the proposed framework, and to conduct evaluation studies. The scenarios are both extracted from the ATM domain. The first one, which is taken from the SecureChange FP7 project, concerns the evolution of both business processes and support-tools of Air Traffic Controllers (ATCOs), particularly, changes in the arrival management process and the introduction of Arrival Manager (AMAN). The second scenario is taken from the SWIM project where evolution occurs at the infrastructure for information interchange and collaboration among ATM systems across airspaces and nations.

The second part presents the body of framework that is conveyed in following chapters:

- *Chapter 5: The Proposed Framework* . This chapter presents the framework, which includes a modeling approach (with evolution rules) and reasoning approach (with evolution metrics). Evolution rules have two kinds: *observable rule* to capture the evolution of requirements, and *controllable rule* to capture the reaction of designers to address such evolution by various choices to implement a system. This chapter also presents a game-theoretic interpretation for the semantics of the evolution probability. Evolution metrics are *Max Belief, Residual Disbelief,* and *Max Disbelief,* which are to assess quantitatively design alternatives, and hence, to facilitate the selection of an evolution-resilient design alternative.

Referred publication(s): This chapter has been partially published in:

- Le Minh Sang Tran and Fabio Massacci. “Dealing with Known Unknowns: Towards a Game-Theoretic Foundation for Software Requirement Evolution”. In: *Proceedings of the 23th Conference On Advanced Information Systems Engineering (CAiSE’11)*. 2011, pp. 62–76
- Le Minh Sang Tran and Fabio Massacci. *Dealing with Known Unknowns: a General Approach for Modeling and Reasoning on Requirements Evolution.* Tech. rep. (to be submitted to the Software and Systems Modeling (SOSYM) journal). University of Trento, 2013

• *Chapter 6: Automated Reasoning Support* . This chapter presents a set of algorithms that automates the analysis and the incremental calculation of metrics described in chapter 5. Requirements models and evolution rules are represented by a hypergraph structure. The algorithms perform calculation and propagation values from leaf nodes to the root. The propagated values at the root are used to calculate values of the metrics. This chapters also presents the computational complexity of the proposed algorithms with proofs.

Referred publication(s): This chapter under submission, which is part of:

- Le Minh Sang Tran and Fabio Massacci. *Dealing with Known Unknowns: a General Approach for Modeling and Reasoning on Requirements Evolution.* Tech. rep. (to be submitted to the Software and Systems Modeling (SOSYM) journal). University of Trento, 2013

• *Chapter 7: Unicorn: Tooling and the First (Self) Evaluation* . This chapter provides an overview about the proof-of-concept CASE tool, namely UNICORN, that implements the modeling and reasoning described in previous chapters. UNICORN takes advantage on the Eclipse plugin infrastructure platform to enhance its extendability. The tool is logically organized into two parts, one for modeling and another one for reasoning. Both are developed to maximize the extendability. We also discuss a performance simulation of the proposed algorithms, as well as a self-evaluation study on a large example taken from an industrial project.

Referred publication(s): This chapter has been published in:

- Le Minh Sang Tran and Fabio Massacci. “UNICORN: A Tool for Modeling and Reasoning on the Uncertainty of Requirements Evolution”. In: *CAiSE Forum*. 2013, pp. 161–168

• *Chapter 8: Empirical Evaluation of the Framework with Third-Party* . This chapter presents a series of empirical studies that aim to evaluate the modeling approach of the proposed framework. These studies follow the Goal-Question-Metric template and orient to different types of participants: researchers, domain experts, and students. All the studies lasted for more than a year. The important outcome from these studies shows that the proposed framework is effective in capturing requirements evolution of complex systems.

Referred publication(s): This chapter has been published in:

- Fabio Massacci, Deepa Nagaraj, Federica Paci, Le Minh Sang Tran, and Alessandra Tedeschi. "Assessing a Requirements Evolution Approach: Empirical Studies in the Air Traffic Management Domain". In: *Proceedings of the 2nd International Workshop on Empirical Requirements Engineering (EmpiRE'12)*. 2012, pp. 49–56
- Fabio Massacci, Federica Paci, Le Minh Sang Tran, and Alessandra Tedeschi. "Assessing a Requirements Evolution Approach: Empirical Studies in the Air Traffic Management Domain". In: *Journal of Systems and Software* (2013). Article in press

The third part shows how the proposed framework could be applied on other aspects of software evolution. Therefore we aim to migrate the proposed framework in requirements evolution into the realm of security risk management. We choose to adapt the framework on the risk assessment process as the importance of early risk identification and mitigation is well known. Since software systems might evolve, their risk pictures consequently might also evolve. Nonetheless, most major approaches in the field do not explicitly support the modeling and analysis on the evolution of risks. This drives the focus in the next part of the dissertation that consists of following chapters:

- *Chapter 9: Early Dealing with Evolving Risks in Software Systems* . Founded on chapter 5, this chapter presents a risk-evolution approach that adapt the proposed framework to early deal with evolving risks. The main objective is to assist the decision makers to select an evolution-resilient countermeasure alternative with respect to evolving risks.

  Referred publication(s): This chapter has been partially published in:

  - Le Minh Sang Tran. "Early Dealing with Evolving Risks in Long-Life Evolving Software Systems". In: *Advanced Information Systems Engineering Workshops – CAiSE Workshops*. 2013, pp. 518–523

- *Chapter 10: Selecting Cost-Effective Risk Countermeasures* . This chapter proposes a risk-graph based method to select cost-effective countermeasure alternative. We extend the risk graph to incorporate the risk mitigation information. We further apply the risk-evolution approach in Chapter 9 to the proposed method to enable a more fine-grain risk evolution modeling. Since risk graph is an abstraction of several model-driven risk analysis techniques, this work provides an evidence that the proposed risk-evolution approach could work on these risk analysis techniques.

The work in this chapter is supported by the Mobility Program of the NESSoS FP7 project, under the supervision of Prof. Ketil Stølen.

Referred publication(s): This chapter has been partially published in:

– Le Minh Sang Tran, Bjørnar Solhaug, and Ketil Stølen. “An Approach to Select Cost-Effective Risk Countermeasures”. In: *Data and Applications Security and Privacy XXVII - 27th Annual IFIP WG 11.3 Conference, (DBSec 2013)*. 2013, pp. 266–273

Finally, the last part provides discussion with respect to research questions and success criteria and conclusion for the dissertation:

- *Chapter 11: Discussion* . This chapter discusses the fulfillment of success criteria, as well as how the proposed framework relates to the literate. This chapter also proposes a potential steps to apply the framework in the software development process.

- *Chapter 12: Conclusion* . This chapter summarizes the major contributions of the dissertation and describes possible future directions based on the results.

1.4 Publications

1.4.1 Publications Reported in the Dissertation

- Le Minh Sang Tran. “Requirement Evolution: Towards a Methodology and Framework.” In: *CAiSE Doctoral Consortium 2011*. London, 2011

- Le Minh Sang Tran and Fabio Massacci. “Dealing with Known Unknowns: Towards a Game-Theoretic Foundation for Software Requirement Evolution”. In: *Proceedings of the 23th Conference On Advanced Information Systems Engineering (CAiSE’11)*. 2011, pp. 62–76

- Le Minh Sang Tran and Fabio Massacci. *Dealing with Known Unknowns: a General Approach for Modeling and Reasoning on Requirements Evolution.* Tech. rep. (to be submitted to the Software and Systems Modeling (SOSYM) journal). University of Trento, 2013

- Le Minh Sang Tran and Fabio Massacci. “UNICORN: A Tool for Modeling and Reasoning on the Uncertainty of Requirements Evolution”. In: *CAiSE Forum.* 2013, pp. 161–168
- Fabio Massacci, Deepa Nagaraj, Federica Paci, Le Minh Sang Tran, and Alessandra Tedeschi. “Assessing a Requirements Evolution Approach: Empirical Studies in the Air Traffic Management Domain”. In: *Proceedings of the 2nd International Workshop on Empirical Requirements Engineering (EmpiRE’12).* 2012, pp. 49–56
- Fabio Massacci, Federica Paci, Le Minh Sang Tran, and Alessandra Tedeschi. “Assessing a Requirements Evolution Approach: Empirical Studies in the Air Traffic Management Domain”. In: *Journal of Systems and Software* (2013). Article in press
- Le Minh Sang Tran, Bjørnar Solhaug, and Ketil Stølen. “An Approach to Select Cost-Effective Risk Countermeasures”. In: *Data and Applications Security and Privacy XXVII - 27th Annual IFIP WG 11.3 Conference, (DBSec 2013).* 2013, pp. 266–273
- Le Minh Sang Tran. “Early Dealing with Evolving Risks in Long-Life Evolving Software Systems”. In: *Advanced Information Systems Engineering Workshops – CAiSE Workshops.* 2013, pp. 518–523

1.4.2 Additional Publications

- Viet Hung Nguyen and Le Minh Sang Tran. “Predicting Vulnerable Software Components using Dependency Graphs”. In: *International Workshop on Security Measurement and Metrics (MetriSec’10).* 2010
- Katsiaryna Labunets, Fabio Massacci, Federica Paci, and Le Minh Sang Tran. “An Experimental Comparison of Two Risk-Based Security Methods”. In: *Proceedings of the ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* 2013
- Le Minh Sang Tran and Fabio Massacci. *An Approach for Decision Support on the Uncertainty in Feature Model Evolution.* Tech. rep. (under submission to CAiSE’14). University of Trento, 2013

CHAPTER 2

RESEARCH ROADMAP

This chapter presents the high level research questions for the whole dissertation, as well as success criteria for research questions. The chapter also provides an overview of the proposed framework in this work to address those questions.

THIS chapter characterizes the problem and the global research questions of this dissertation, and the research activities to address these questions. This chapter is organized as follows. Section 2.1 presents problem characterization and research questions that drive the rest of the dissertation. Section 2.2 shows the evaluation strategies for invented artifacts. Section 2.3 briefly discusses the proposed framework in this work. Finally Section 2.4 summarizes the chapter.

2.1 Problem Characterization and Research Questions

2.1.1 The Need of Managing the Evolution Uncertainty of Requirements Models

Long-lived systems are built to satisfy a set of mandatory requirements in a period of time, and operate under assumptions of working environment and context. However, reality might change and evolve. Nowadays, evolution is an inevitable phenomenon [Set+04] [Lam09b, Chap. 6] during the life time of a long-lived software systems due to the dynamic of their working environments. Consequently, software systems might be unstable or non-operational.

Evolution refers to potential changes *in future* to software artifacts such as requirements models. Requirements evolution could be understand as potential future changes to requirements models, for example, new requirements might arise, some current requirements may become obsoleted. Such changes could cause the systems stop working properly and lead to significant economic loss. Fortunately, we can anticipate possibilities of these changes with the help of expertise knowledge in the domain. They are *known*, but which possibilities of changes could happen is *unknown*. Hence, a key point of evolution is its *uncertainty*. Evolution might or might not occur due to some external factors such as changes in regulation, law, and standards. These factors usually cannot be controlled by either software developers or stakeholder. The uncertainty of evolution however could be partially predicted with the aid of domain expertise knowledge, for instance ATM experts could foresee new standards, regulations in the ATM domain in next few years. The likelihood of occurrence of evolution thus could be estimated.

As requirements might change and evolve over time, systems might be non-operational because some mandatory requirements are not satisfied. It is essential for long-lived systems that they need to be still operational during their lifetime. Requirements evolution thus should be taken into account during the software development process. However, many existing requirements modeling languages (*e.g.,* Tropos, KAOS) provide neither notations nor tools to capture evolution in requirements models.

There are some efforts to extend these original modeling languages to support the evolution in software development [CH11; MD00; FB01]. However, none of them pays attention to the uncertainty of evolution. Additionally, these studies are still lacking of a systematic and quantitative reasoning to support decision makers in order to choose a system design alternative that is more *resilient* to potential evolution.

An *evolution-resilient* design alternative, in the context of this dissertation, is understood as a design alternative that has more chances to be operational even if evolution happens, or requires less modification to be operational due to evolution.

2.1.2 Research Questions and Success Criteria

The ultimate objective is to provide a framework that leverages on the uncertainty of evolution to aid the selection of evolution-resilient design alternatives. For this objective, we target following research questions in this dissertation:

Table 2.1: Success Criteria.

RQ	Success Criteria
RQ1	SC1 A modeling approach that is: **SC1.1** *Able to effectively capture the requirements evolution and its uncertainty.* **SC1.2** *Accompanied with a formal semantics of the evolution uncertainty.* **SC1.3** *Potentially applicable to a variety of requirements and system models.*
RQ2	SC2 A reasoning approach that is: **SC2.1** *To provide a set of metrics with formal semantics for reasoning about evolution uncertainty.* **SC2.2** *Able to automate (with formal analysis and tool-support) the reasoning that can enumerate and quantitatively assess individual design alternatives.* **SC2.3** *Able to support the incremental modeling of evolution.*

RQ1 *How to effectively capture the uncertainty of evolution in requirements models and its uncertainty?*

RQ2 *How to perform reasoning about the evolution uncertainty to support decision makers?*

To satisfy the above questions, we propose a framework that could capture the requirements evolution and evolution uncertainty. The framework provides a quantitative analysis that support the identification and assessment of design alternatives. To facilitate the evaluation of the proposed framework, in Table 2.1 we identify several success criteria with respect to research questions. The details are as follows:

Addressing RQ1. The proposed framework should include a modeling approach that is:

- SC1.1 *Able to effectively capture the requirements evolution and its uncertainty*. The modeling approach should take into account a key issue in requirements evolution, identified by Lam and Loomes [LL98], which is *multiple change*. This issue refers to the situation that there are many parallel, competing changes. To address future changes, a requirements model is revised to a new revision. Due to parallel and competing changes, the new revision could have many variants. When evolution actually happens, only one variant holds.

An important assumption that drives the modeling approach is that we only consider predictable changes (*i.e.,* the *known*) whose occurrences are not sure for 100%, but could be estimated at some level of uncertainty (*i.e.,* the *unknown*). We do not take into account *unknown* changes that could not be predicted, and their occurrences are *unknown* as well. In short, this work focuses on *known-unknown* evolution, *not* unknown-unknown ones, which are analyzed in different direction (see works of Ernst *et al.* [Ern+11; Ern+09]).

- SC1.2 *Accompanied with a formal semantics of the evolution uncertainty*. The evolution uncertainty is represented as evolution probability. This probability has clearly not a frequentist because we cannot measure the frequency of an evolution since it might occur only once. Thus it is necessary to an interpretation of evolution uncertainty to understand the semantics of this uncertainty. Such interpretation and understanding are very helpful for elicitation and evaluation of acquired evolution uncertainty.

- SC1.3 *Potentially applicable to a variety of requirements and system models*. There are many existing Requirements Engineering (RE) languages for modeling requirements. The modeling approach does *not* aim to provide a new RE language, but it should be a set of notions and concepts that could be able to adapt to a particular RE language and make it possible to deal with evolution. Evolution might happen not only in requirements models, but also in other system models such as product line modeling or risk assessment. It thus is better that the modeling approach could be adapted to support the modeling of evolution also for other kinds of artifacts.

Addressing RQ2. The proposed framework should include a reasoning approach that is:

- SC2.1 *To provide a set of metrics with formal semantics for reasoning about evolution uncertainty*. It is important that the reasoning approach could facilitate the selection of evolution-resilient design alternatives. For this purpose, it is necessary to have a set of metrics that quantitatively estimate to what level of evolution-resilience a design alternative could be. The metrics should also be backed up with a formal semantics.

- SC2.2 *Able to automate (with formal analysis and tool-support) the reasoning that can enumerate and quantitatively assess individual design alternatives*. Requirements models are usually big and complex, so does the requirement evolution. It is also important that the reasoning is automated (fully or partially). Thus the proposed reasoning approach should include a set of algorithms that enumerate design alternatives, as well as calculate the metrics for each design alternative.

- SC2.3 *Able to support the incremental modeling of evolution*. While developing complex software systems, the iteration approach is usually preferred than the legacy waterfall approach. Therefore the process of modeling and reasoning about requirements evolution could also have several iterations where changes are identified incrementally. The reasoning approach therefore should be support the incremental modeling of evolution. It should efficiently deal with this incremental behavior rather than start over the analysis from ground up.

2.2 Strategies for Evaluation

In this section we discuss some evaluation activities that are usually taken into consideration while evaluating an RE methodology or artifact. These evaluation activities are described as follows:

E1 *Self-evaluation study* (sometimes called "case study"): the author(s) of the research applies the methodology or use the artifact in a complex, real life scenario. The purpose of this study is to show that the artifact can capture most of the important features of the scenario. This approach is used in [Bry+09; Sol+07; Erd+13].

E2 *Empirical study*: the author(s) of the research conduct empirical study(ies) with third people who apply the methodology or use the artifact in complex, real life scenarios. The purpose of this study is to show how other people could use the proposed methodology (or artifact) to capture most of the important features of the scenarios. This approach is used to study other characteristics of the methodology (or artifact) such as effectiveness, applicability, perceived ease of use, intend to use. This approach is used in [Ncu+07; MR05; Mai+04; Vil+10].

E3 *Formal semantics*: the author(s) of the artifact proposes an *independently* motivated formal artifacts that can be used to characterize some features of the original artifact.

Table 2.2: Evaluation activities to fulfill success criteria.

		Modeling			Reasoning		
		SC1.1	SC1.2	SC1.3	SC2.1	SC2.2	SC2.3
E1	Self-evaluation study	✓		✓	✓		
E2	Empirical study	✓		✓	✓		
E3	Formal semantics		✓		✓	✓	
E4	Tool construction	✓		✓		✓	✓
E5	Algorithmic complexity analysis					✓	✓
E6	Simulation experimental algorithmic analysis					✓	

The purpose of this evaluation is to show that the mathematical underpinnings of the original artifact can be explained and characterized by a known framework. This approach is used in [Jür02; Hau+03; LS06]

E4 *Tool construction*: the author(s) of the artifact constructs a proof-of-concept tool that implements the artifact. This is to partially show the applicability of the artifact. It might be also used to study the empirical applicability of the artifact (*e.g.*, via empirical simulation). This approach is used in [Dal11; Ali10].

E5 *Algorithmic complexity analysis*: the purpose of this evaluation activity is to show the worst case complexity of the approach and to identify the cause of intractability. This approach is used in [Mas+06].

E6 *Simulation experimental algorithmic analysis*: the purpose of this evaluation activity is to provide an approximation to the performance of the proposed algorithms. This approach is used in [Dal11].

Not all evaluation activities are equally suited or equally accepted to prove the efficacy and applicability of a research artifact. For example in RE, a Self-evaluation study (E1) and a Tool construction (E4) are the most popular evaluation techniques followed by a Formal semantics (E3) and frequently by empirical studies [CF+09].

In contrast, for reasoning techniques Formal semantics (E3) and Algorithmic complexity analysis (E5) are studied first followed by Tool construction (E4) and Simulation experimental algorithmic analysis (E6). This diverse preference for evaluation can be explained by the diverse expectations of the corresponding artifacts: a modeling artifact is expected to "ease" the work of the analyst and hence it is crucial for Self-evaluation study (E1) and Empirical study (E2). On the opposite side, a reasoning algorithm should not be sensitive to the human pressing the button, but only to the "shape" of the problem (*i.e.,* problem dependence, not human dependence). In Table 2.2 we have clustered the evaluation activities in two groups (one for modeling and another one for reasoning).

2.3 Framework Overview

Figure 2.1 presents an overview about the proposed framework and various activities to evaluate it. In the figure, the framework artifacts are represented by rectangles, and evaluation activities are depicted as parallelograms connected to relevant artifacts. The input of the framework is a requirements model in a particular RE language and anticipated evolution with its likelihoods of occurrences. The output of the framework is a set of design alternatives quantified by evolution metrics. Below we describe the artifacts of the framework:

- *Evolution Rules* (see Section 5.1) to serve as concepts and notations to capture the evolution in requirements models at high level of abstraction.

- *Game-Theoretic Interpretation* (see Section 5.2) for the evolution uncertainty. The evolution uncertainty, which is further referred to as *evolution probability*, is the belief of domain experts about the chance to be materialized of evolution in future. It is a kind of subjective probability. Thus we need a semantics backing up for evolution probability that could be helpful to facilitate the elicitation and evaluation of these probabilities. This interpretation is also used to provide the semantics of evolution metrics.

- *Evolution Metrics* (see Section 5.3) that leverages on evolution uncertainty to support the decision makers in the selection of evolution-resilient design alternatives for the system. The evolution-resilience is defined as the probabilities to be operational regardless of evolution. The semantics of evolution metrics are also accounted by the game-theoretic interpretation.

- *Hypergraph Requirements Model* (see Section 6.1) We employ hypergraph to represent requirements models. This structure is *not* a substitution of any existing RE languages, but it could capture necessary information for the automated reasoning. Thus it could be transformed from a requirements model expressing in a particular RE language.

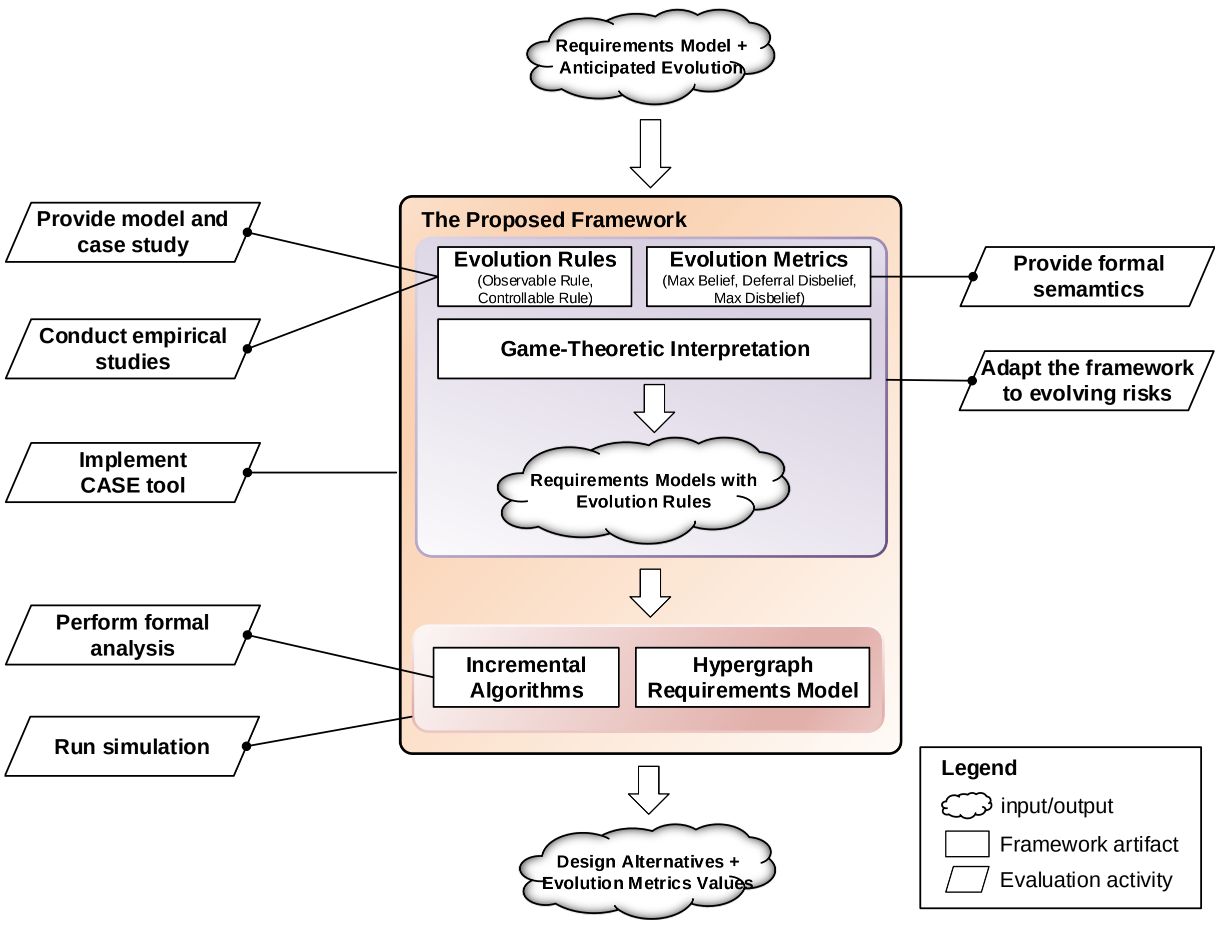


Figure 2.1: The framework overview and associated evaluation activities.

When transforming requirements models into hypergraphs, only requirements and necessary information about requirements satisfaction are transformed.

- *Incremental Algorithms* (see Chapter 6) to automate the enumeration of design alternatives and the calculation of evolution metrics. These algorithms take a hypergraph requirements model as input.

Figure 2.1 also illustrates various activities to evaluate the proposed framework. The order of evaluation activities in the figure does *not* represent the actual order of execution of these activities.

- *Provide formal semantics* (E3, see Section 5.2–5.4). We have provided a formal semantic for the proposed evolution rules and evolution metrics. This is to ensure that they do make sense. We also provided a formal mathematic model to back up the proposed metrics.

- *Perform formal analysis* (E5, see Section 6.3, 6.4). To evaluate the algorithms, we described a set of mathematical properties and their proof of the algorithms. The details of properties and algorithms are presented in Section 6.4. To evaluate that the developed algorithms could be applied to different RE languages, we have proposed several transformation patterns transforming i*, KAOS, and SysML requirements models to hypergraphs. We also adapt the proposed framework in i* language to conduct the empirical studies mentioned above.

- *Implement CASE tool* (E4, see Chapter 7). To further evaluate the framework, we have implemented a CASE tool, namely UNICORN. The tool consists of a Graphical User Interface (GUI) for hypergraph requirement model, and an implementation of the algorithms.

- *Provide model and case study* (E1, see Section 7.4, 7.5). We have conducted a self-evaluation study to evaluate the proposed framework. We applied the proposed framework to model the evolution in a requirements model of a case study taken from an industrial project. Notably, here the term *"case study"*, which is also used by other RE researchers, is different from the case study used in the field of empirical experiments (see further Table 3.1).

- *Run simulation* (E6, see Section 7.4). We have run a simulation on an implementation of the algorithms. The simulation is to study the empirical performance of the algorithms with respect to the complexity of input hypergraphs (or the complexity of requirements models).

- *Conduct empirical studies* (E2, see Chapter 8). To evaluate the high level semantics of the evolution rules we have conducted an empirical study with domain experts to obtain their feedback about using evolution rules to model requirements evolution. To evaluate the effectiveness of the proposed framework, we have conducted other empirical studies with researchers, domain experts, and master students.

- *Adapt the framework to evolving risks* (E1, see Chapter 9 , 10). We demonstrate the applicability of the proposed framework in other field of software engineering rather than RE by apply the proposed framework to deal the evolution of risks in long-lived system.

2.4 Chapter Summary

This chapter discussed the research objective of this dissertation. The objective was refined into three research questions. The focus of the research questions was to propose a framework that supports modeling and reasoning about the uncertainty of evolution in requirements models; and how to evaluate the proposed framework. Based on the research questions, we have elaborated an overview of the proposed framework with various artifacts needed for the framework, as well as different aspects to evaluate it. The next Chapter 3 will provide a background about different evolution perspectives and summarize studies that are relevant for this dissertation.

CHAPTER 3

STATE-OF-THE-ART

This chapter provides a background about different evolution perspectives that we address in this work. The chapter also briefly summarizes relevant studies in the field of requirements evolution, as well as empirical studies on requirements from which we learn to conduct evaluation study for the proposed framework. We also briefly review studies in the risk assessment area where we later adapt the proposed framework to early address evolving risks.

PAST studies do strongly impact present work. Thus in this chapter we briefly review recent studies relating to the purpose to understand the status of the art. Based on this understanding we build the proposed framework. This chapter is organized as follows. Section 3.1 presents a background about different evolution perspectives. Section 3.2 reviews past studies in the field of requirements evolution. Section 3.3 reviews empirical studies on requirements. Section 3.4 review studies in the field of risk analysis. Section 3.5 concludes the chapter.

3.1 Evolution Perspectives

Evolution in a requirements model is basically studied within a time period. This study period may be short (one year), or very long (ten years or more) depending on the life time of the software system. Depended on the scenarios about how to study evolution, we have different evolution perspectives. Lund *et al.* [Lun+11a, Chap. 15] describe three evolution perspectives in risk analysis, including: *maintenance, before-after, continuous evolution.* Out of these, the maintenance evolution perspective mostly relates to the outdate of a document (*e.g.,* requirements model) of an existing system. Since this work mostly focuses on the early solution of requirements evolution, this evolution perspective is not being considered in this work.

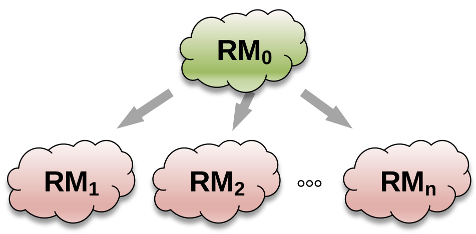
We can adapt the last two evolution perspectives in the proposed framework to deal with evolution in requirements models as follows.

- *Before-After evolution perspective* predicts future contexts by anticipating planned and unplanned changes in the current requirements models at the end of the study period. Multiple possibilities of evolutions are considered. Each can have its own likelihood of occurrence. For example, the ATM 2000+ Strategic Agenda [Eur03] and SESAR Initiative [SES08] have outlined the direction of the ATM developments in a period 2010 to 2020 to have one or more of variants of new queue management tools including Arrival Manager (AMAN), Departure Manager (DMAN) or Surface Manager (SMAN).

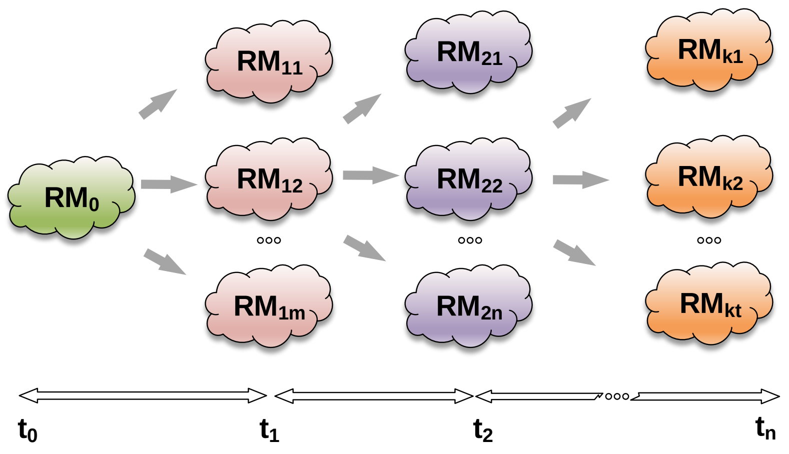
- *Continuous evolution perspective* predicts the evolution of the current context over time based on planned gradual changes. In this perspective, the entire study period is divided into several milestones. At each milestone, potential changes in the requirements model are anticipated. Multiple possibilities of evolution are also allowed.

The major difference between this continuous evolution perspective and the one with similar name described in [Lun+11a, Chap. 15] is that: while the latter considers only one possibility of evolution at certain time points in future; the former enables multiple possibilities of evolution at certain time points. The former will collapse to the latter if there is only one possibility of evolution in each time point. Therefore, the definition of continuous evolution perspective here is the generalization of Lund *et al.* [Lun+11a]'s one.

Figure 3.1 visualizes the evolution perspectives. In this figure, a requirements model is depicted as a cloud. Figure 3.1(a) illustrates the *before-after* evolution perspective where a requirements model might evolve to one of possibilities models at the end of the study period. Meanwhile, Figure 3.1(b) exemplifies the *continuous* evolution perspective. In this illustration, at time t_0 the original requirements model is RM_0, which can evolve to one of RM_i at time t_1. The evolution continuously happens. And at the end of the study period, time t_n, the origin model could be one of RM_{kj}.



(a) Before-after evolution



(b) Continuous evolution

Figure 3.1: The evolution perspectives of requirements evolution.

3.2 Studies on Requirements Evolution

Lam and Loomes [LL98] present the EVE (Evolution Engineering) framework, which includes a meta model and a process model to deal with requirements evolution. They classify changes into the following types:

- Environment change (E-change): these are changes that occur within the environment where the target software system is working, *e.g.*, the introduction of new laws, changes in business agreement.
- Requirement change (R-change): these are changes in requirements and derived from environment changes, *e.g.*, new requirements arrive, current requirements are modified or deleted.
- Viewpoint change (V-change) also called ‘impact’: these are impacts on the ‘life’ of stakeholders once R-changes are implemented. For example, the introduction of AMAN will change the way the arrival sequence is computed (*i.e.*, instead of computed by the

PLC); the SWIM will change the way of communication from telephone-based to an ICT-based infrastructure among ATM actors.

- Design change (D-change): these are changes in the design of the system implied by requirement change, *e.g.,* the introduction or removal of a function, control.

The authors also mention seven key issues that need to be addressed by requirements evolution studies. Those issues include:

- *Modeling evolution*: this is the most important and a fundamental issue. This refers to the way to model, represent, and reason about evolutions.
- *Change analysis*: this is to study about the nature of change to answer 'why' and 'where from' questions about changing requirements.
- *Impact assessment approaches*: this is to assess the impact of changing requirements on other domains, such as social, environmental and cultural issues.
- *Risk assessment*: this refers to risk assessment in relation to requirements changes.
- *Multiple change*: this is the ability to consider many changes that are parallel, and possibly competing. These changes may be originated from environmental changes.
- *Extended traceability*: this is concerned with establishing relationships between engineering artifacts during software maintenance.
- *Tool support*: this is concerned with tool to provide support for requirements evolution such as change management tool, traceability tool, and so on.

We can broadly classify relevant studies about requirements evolution into *impact of evolution*, and *reaction on evolution* with respect to their major contributions. Studies on the impact of evolution aim at identifying potential consequences on artifacts (such as models, specifications) and violation of consistency or security properties. Studies on the reaction to evolution propose reactions to requirements evolution. Bellow we briefly review the related studies.

3.2.1 Studies on Impacts of Evolution

Russo *et al.* [Rus+99] propose an analysis and revision approach to restructure requirements to detect inconsistency and manage changes. The main idea is to allow evolutionary changes to occur first, and then in the next step, verify their impact on requirements satisfaction. The restructuring includes three activities: (1) decomposing the specification into parts, (2) representing those parts within viewpoints (in four templates: hierarchic tree, input-output flow, data-flow diagram, state transition diagram), and (3) enriching the viewpoints with rules that express relationships between different specification fragments. The impact analysis includes steps to check consistency and completeness, tracking changes in the original requirements specification. However, their approach is manually performed.

Hassine *et al.* [Has+05] present an approach to change impact analysis, which refers to identify potential consequences of a change, or estimate what needs to be modified to accomplish a change. Their approach applies both slicing and dependency analysis at the Use Case Map specification level to identify the potential impact of requirements changes on the overall system. The resulting output includes scenarios and components that subject to change. Chechik *et al.* [Che+09] and Lin *et al.* [Lin+09] focus on change propagation. Chechik *et al.* [Che+09] assist users in propagating changes across requirements, design, and implementation artifacts. They use the UML model-based approach and provide automated propagation for changes between requirements and design models via relationships between them. The relationship between models are formalized by OCL rules. The automated change propagation between models localizes the regions in models that should be modified. However, the work is limited to activity diagrams and sequence diagrams of UML language. Lin *et al.* [Lin+09] capture requirements changes as series of changes in specifications. They propose algorithms for managing all possible atomic requirements changes to a sequence-based specification. The algorithms address atomic requirements changes and push them through changes in specifications, maintain old specifications over time and evolved into new specifications with least rework.

Fabbrini *et al.* [Fab+07] work on requirements expressed in natural language. They address the inconsistency between requirements that belong to different evolutionary stages (or evolution steps). Their approach employs Formal Concept Analysis theory to verity the requirements consistency. Requirements of different evolutionary stages have a traceability link to others. By checking the inconsistency, they may detect the error in evolution from one requirement in a stage to another requirement in a different stage.

The authors in [Pro12; Ber+11] present the SECMER methodology for requirements evolution management developed in the context of the SecureChange project [1]. The methodology addresses the before-after evolution perspective and provides support for:

- Modeling requirements evolution: requirements models of the 'before' or 'after' situation are depicted in the Si* language.
- Change management based on evolution rules: security properties are modeled as patterns by Si* language. Any changes made to the requirements model is checked with the argument validity to detect the violations or fulfilment of security properties. If a security property is violated, an alert prompting human intervention is automatically issued with possibly suggested corrective actions.
- Argumentation-based security analysis: this provides evidence if a security property is preserved by evolution or not.

3.2.2 Studies on Reaction on Evolution

Apart from the above approaches that aim to manage requirements evolution (such as checking inconsistency, propagating changes, and change impact) at design time. There exists a number of approaches aiming to support the system evolution driven by requirements evolution [Bri+06; ZO97; Sou+11; Ern+11]. Some of these works focus on the design phase while others target the deployment and execution phases.

Brier *et al.* [Bri+06] present a manual change analysis process in which a situation 'before-the-change' is changed into a situation 'after-the-change'. They adopted the diagrammatic notation from Problem Frame to model these situations. The approach however did not go further with any specific reasoning for change rather captures the part of before model that is changed.

Zowghi and Offen [ZO97] work at meta level to capture intuitive aspects of managing changes to requirements models. Their approach involves modeling requirements models as theories and reasoning changes by mapping changes between models.

Among the work focusing on run-time, Souza *et al.* [Sou+11] propose a systematic method for adaptive software system. In this approach, the dynamic behavior of the system is governed by of a set of (in)equations called *qualitative differential constraints*. The authors characterize the controllability space for a software system defined in terms of a requirements model, variation points, control variables and indicators. All of these are correlated with qualitative differential constraints.

[1]http://www.securechange.eu/

Table 3.1: Overview of Research Methodology

Evaluation Method	**Description**
Case Study	Monitor a phenomenon in its real context
Experiment	Investigate a testable hypothesis
Survey	Collect standardized information from a specific population
Ethnography	Study a community and community's members social interactions
Action Research	Study the experience while solving a problem
Assertion	Use ad-hoc evaluation techniques
Lessons Learned	Examine qualitative data from complete projects
Benchmarking	Test performance running several tests
Screening	Feature-based evaluation done by a single individual
Effects Analysis	Use expert opinion to assess the quantitative effects of methods/tools
Project Monitoring	collect and store data during project development
Field Study	Monitor and collect data about different projects simultaneously
Literature Research	Analyze papers and other documents publicly available
Legacy Data	Examine data from completed projects trying to identify trends

Ernst *et al.* [Ern+11] focus on unknown-unknown evolution, *i.e.,* evolution that we do not know what it is, and when it happens. Instead of finding a solution anticipating evolution, the authors in [Ern+11] study a class of algorithms using *AI Truth Maintenance Systems* (ATMS) to find new solutions that use as much as possible of the old solution (*i.e., maximize familiarity*), and minimize the number of tasks that need to be implemented (*i.e., minimize effort*).

3.3 Studies on Empirical Evaluation

In order to have a glance on the types of empirical studies so that we can learn to conduct the evaluation for the proposed framework (see Chapter 8 later), we first overview the existing empirical research methodologies and introduce a set of terms in the field of empirical research. Then, we discuss the works reporting empirical studies on requirements evolution.

3.3.1 Empirical Research Methodologies

Different taxonomies [RH09; Eas+07; Kit96; ZW98; Bas+86] have been proposed to classify empirical research methodologies in software engineering. In Table 3.1 we summarize the major research methodologies from the taxonomies in [RH09; Eas+07; Kit96; ZW98; Bas+86].

Runeson and Host [RH09] identify four classes of empirical research methods: *case study*, which is an empirical inquery that investigates a contemporary phenomenon within a real-life context; *survey* that is a collection of standardized information from a specific population by means of a questionnaire or interview; *experiment* is an investigation of a testable hypothesis when one or more independent variables are manipulated to measure their effect on one or more dependent variables; *action research* aims to solve a real-world problem while simultaneously studying the experience of solving the problem. Easterbrook *et al.* [Eas+07] also count *ethnographic studies* among the major research methodologies. Ethnographic research studies based on field observations a community of people to understand how the members of that community make sense of their social interactions. Kitchenham [Kit96] also consider case study, experiment and survey as classes of empirical research methodologies, but it also identify *screening*, *effects analysis*, and *benchmarking* as classes of research methods. Screening is a feature-based evaluation done by a single individual who not only determines the features to be assessed and their rating scale but also does the assessment. Effects analysis is a method that uses expert opinion to assess the quantitative effects of different methods and tools. Benchmarking is a process of running a number of standard tests/trials using a number of alternative tools/methods (usually tools) and assessing the relative performance of the tools in those tests.

Zelkowitz and Wallace [ZW98] enrich the taxonomies proposed in [RH09; Eas+07; Kit96] with new classes of empirical research methodologies: *project monitoring*, *assertion*, and *field study*, *literature research*, *legacy data*, and *lessons learned*. Project monitoring focus on the collection and storage of data that occurs during project development. An assertion is an experiment where the designer of a new technology is both experimenter and subject of study[2]. A field study monitors and collects data about different projects simultaneously. Literature research analyzes papers and other documents publicly available. Legacy data is method that examines data from completed projects trying to identify trends. Lessons learned examine data from complete projects to identify qualitative aspects that can be used to improve further developments.

3.3.2 Empirical Studies on Requirements Evolution

Below we briefly review studies that are closely related to the empirical studies, which are later described in Chapter 8.

[2]A subject of study is an agent that is studied and collected data on.

Villela *et al.* [Vil+10] present on quasi-experiment in the field of Ambient Assisted Living to study the adequacy and feasibility of PLEvo-Scoping method [Vil+08]. That method is based on a software evolution model to help requirements engineers and product managers identify the unstable features of an embedded system and their potential needed adaptations. It allows to identify and prioritize likely future adaptation needs and to select solutions to deal with them. Their quasi-experiment follows the Goal-Question-Metric template [BR88] and involves three kinds of roles: method expert, stakeholder, and domain expert. The quasi-experiment took place in the form of two two-days workshops where two groups consisting of three domain experts applied PLEvo-Scoping. The first part of each workshop was dedicated to the presentation of the application domain, and the quasi-experiments task. Both quantitative and qualitative measures were used to evaluate the adequacy and the feasibility of the method. However, due to the small number of subjects, the authors were not able to perform any statistical tests.

McGee and Greer [MG11] conducted a case study [RH09] to assess if a change taxonomy proposed by the authors helps to understand the consequences of requirements change, why and when it happens. The study was conducted during the execution of a software development project in the government sector and involved 15 software developers and analysts. Data on requirements changes were collected during the different phases of the software development life cycle. The quality of changes was assessed by a researcher and a project manager. The authors defined quantitative metrics to answer their research questions like the number of changes and the cost of changes and used hypothesis testing to evaluate the hypotheses related to their research questions.

Another study on requirements evolution by Herrmann *et al.* [Her+] is one of the pioneer in specifying the delta requirements without having to describe complete system in details. Herrmann *et al.* investigates the applicability of TORE, a requirements engineering approach to identify delta requirements for an engineering tool. Delta requirements refer to changes in requirements identified when comparing the as-is system with the system-to-be. The study measures improvements in the as-is-analysis, the to-be-analysis, and the prioritization of requirements.

Maiden et al. [Ncu+07; MR05; Mai+04], have presented several case studies in the ATM domain to validate RESCUE, a scenario-driven requirements engineering process. In the studies, the authors ran several creativity workshops with ATM experts with different expertise to study how RESCUE helps to discover stakeholder and system requirements. The workshops were organized in three main phases: a *training* phase about RESCUE, a *brainstorming* phase, and then an *application* phase where the experts applied RESCUE to discover requirements for different ATM tools (*e.g.*, DMAN, CORA-2, and MSP). During the workshops, color-coded idea cards, post-it notes, A3 papers have been used to collect the results. The authors claimed that, although not all the workshop sessions were a success, the overall process definitely was – as it helped to set up a common understanding and facilitated the interaction among people involved.

3.4 Studies on Selecting Risk Countermeasures

Mehr and Forbes [MF73] suggest that "risk management theory needs to merge with traditional financial theory in order to bring added realism to the decision-making process". In line with the suggestion, Cost-benefit analysis (CBA) is often used with risk management to assess the effectiveness of risk countermeasures [AP01; Boa+01; Sen00]. Major CBA steps include: *a)* develop measures to mitigate a certain problem *b)* develop measure alternatives *c)* estimate the impact and cost of each measure *d)* compare the benefit and costs for each measure alternative *e)* conduct a sensitive analysis of the uncertainty of estimated benefit and cost *f)* recommend a cost-effective measure alternative for implementation.

In risk management, decision on different risk countermeasure alternatives has been emphasized in many studies [Sto+02; Nor10; WHO09]. The guideline in [Sto+02] proposes cost-benefit analysis to optimally allocate resources and implement cost-effective controls after identifying all possible countermeasures. This encompasses the determination of the impact of implementing (and not implementing) the countermeasures, and the estimated costs of them. Another guideline [WHO09] provides a semi-quantitative risk assessment. The probability and impact of risks are put into categories, which are assigned with scores. The differences between the total score for all risks before and after any proposed risk reduction strategies relatively show the efficiency among strategies, and effectiveness of their costs. It also suggests that the economic costs for baseline risks should be evaluated using one of the following methods: Cost-Of-Illness, Willingness-To-Pay, Qualified-Adjusted Life Years, Disability-Adjusted Life Years. These methods have been designed to assess cost of risks, but not cost of countermeasures.

Butler [But02] proposes the Security Attribute Evaluation Method (SAEM) to evaluate alternative security designs. It employs a four-step process, namely benefit assessment, threat index evaluation, coverage assessment, and cost analysis. This approach focuses mostly on the consequence of risks rather than cost of countermeasures.

Chapman and Leng [CL04] describes a decision methodology to measure the economic performance of risk countermeasure alternatives. The methodology is based on two kinds of analysis (baseline and sensitivity), four methods of economic evaluation, and a cost-accounting framework. The cost is broken down into several dimensions and types. The advantage is to provide a clear economic justification among countermeasure alternatives. However, it does not differentiate alternatives based on their suitability to mitigate risks. In other words, the methodology focuses on the cost-difference aspect but does not take into account the benefit-difference (in terms of level of risks reduced) among alternatives.

There exist studies on Real Options Thinking [Kul+99; AK99; LS07] to articulate and compare different security solutions in terms of their business value. These solutions however are on the management aspect such as postpone, abandon, or continue to invest in security. Norman [Nor10] advocates the use of Decision Matrix to agree on countermeasure alternatives. A Decision Matrix is a simple spreadsheet, which contains a list of countermeasures, and a list of risks, which those countermeasures mitigate. For each countermeasure, there are estimates with respect to cost, effectiveness, and convenience. The countermeasure effectiveness is measured by metrics contained within the Sandia Vulnerability Assessment Model. That approach is however not clearly defined, and all metrics are developed as spreadsheets, which are complicated to implement and follow.

Houmb *et al.* [Hou+12] introduce SecInvest, a security investment support framework that derives a security solution fitness score to compare alternatives and decide whether to invest or to take the associated risk. SecInvest relies on an eight-step trade-off analysis, which employs existing risk assessment techniques for risk level. SecInvest scores alternatives with respect to their cost and effect, trade-off parameters, and investment opportunities. However, that approach does not provide a systematic way to assess the effects of alternatives on risks, either not take into account the dependency among countermeasures in an alternative.

Existing risk assessment methods, such as some mentioned above, mostly perform on a target software system at a particular point in time. However, when software evolves, risks might also evolve. Lund *et al.* [Lun+11b] and Solhaug and Seehusen [SS13] propose general techniques and guidelines for managing risk in changing systems. In particular, they proposed a risk assessment method for long-lived evolving system that includes assessment steps, language for the modeling and documentation of changing risks, and techniques for tracing evolution from requirements models to risk models. This is one of the pioneers in the field of evolving risks.

3.5 Chapter Summary

This chapter presented a background understanding about the state-of-the-art. In the subsequent chapters, we will discuss about the application scenarios from which we exemplify the notions and concepts of the proposed framework. These scenarios also serve for the evaluation purpose of the framework.

CHAPTER 4

APPLICATION SCENARIOS

This chapter describes application scenarios taken from industrial projects in the ATM domain. These scenarios are used to exemplify all principle and concepts in the proposed framework, and to conduct a series of empirical studies to evaluate the proposed framework.

IN order to improve the comprehension of the proposed framework, we try to exemplify as much as possible principles and concepts. All examples are taken from industrial projects in the ATM domain. In this chapter we discuss application scenarios from which examples are built. To the brevity, we only focus on parts of systems rather the entire projects.

Software systems in the ATM domain usually are very long lasting lifetime and evolvable. Due to the continuous increasing of the air traffic, as well as the increasing of security, dependability, and performance requirements, these software systems have to face to many kinds of changes during their long lifetime. These systems must be able to accommodate for changes in the controlled process, such as improved aircraft performance, to host new controller supporting tools, and be compatible with possible new control procedures and rules applied by the controllers. Therefore, good designs that early consider (as much as possible) potential future changes will increase the reliability of ATM systems and provide smooth operations in air traffic controller.

Table 4.1: Technical documents of the scenario.

Name	Document Title	Description
SC-D1.1[(1)]	Description of the scenarios and their requirements	describes in detail the requirements for the ATM scenario. Changes concerning to the introduction of AMAN are also elaborated.
SWIM-D1.2.1[(2)]	Information Content and Service Requirements	describes an overview of SWIM, ATM information content requirements and services requirements.
SWIM-D1.6.1[(2)]	SWIM Prototype Requirements for Iteration	describes the system context that the SWIM will face and support, including a set of usecases, scenarios where SWIM integrates with other systems. Requirements for the prototype iteration are also elaborated.
SWIM-D2.3.1[(2)]	SWIM-SUIT information models and services	describes existing ATM information systems, and future SESAR ATM system, as well as the role of SWIM network in the SESAR ATM architecture. Evolution of the SWIM services is also elaborated.
SWIM-TECH[(3)]	Segment 2 Technical Overview	describes in detail the functional architecture of SWIM, including architecture options, design solutions, and technologies.

Sources:
(1)`http://www.securechange.eu/content/deliverables`
(2)`http://www.swim-suit.aero/swimsuit/projdoc.php`
(3)`http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/atc_comms_services/swim/documentation/media/Segment%202/SegmentTechnicalOverview_10709.pdf`

We employ these scenarios in this dissertation because we have good contacts with practitioners (each has more than 10-year experience in the ATM domain in the context of the Secure Change project[1]. This would ease the conduction of the empirical evaluation, and make the outcomes of the evaluation more realistic.

Table 4.1 presents a list of technical documents from which ATM scenarios are taken. These documents are provided by Deep Blue Srl, an Italian consultancy company specialized in human factors, safety and evaluation of ATM concepts and systems, which actively participates in the SESAR Initiative.

[1]`http://www.securechange.eu/`

This chapter is organized as follows. Section 4.1 describes the SWIM scenario from which illustrative examples for the framework are taken. Section 4.2 briefly presents the ATM application scenarios that we employ to conduct a series of empirical evaluation studies to evaluate the proposed framework.

4.1 The SWIM Scenario

The System Wide Information Management (SWIM) [Adm09] is an information management infrastructure, which connects all ATM actors (both human and non-human) from aircrafts to ground facilities. It allows seamless information interchange and the capability of finding the most appropriate source of information while catering for information security requirements.

FAA has proposed a logical architecture of SWIM that consists of several functional blocks. Among these blocks, we focus on the possible evolution of the Enterprise Information System Security (ISS-ENT), and External Boundary Protection (BP) [Adm09, section 5.6]. To keep the running example simple and illustrative, we only present a small subset of requirements and requirements evolution of both ISS-ENT and BP. Interested readers are referred to [Adm09, section 5.6] for further details.

The ISS-ENT plays as part of the underlying infrastructure used by enterprise systems. The ISS-ENT includes two sub parts: Identities and Keys Management Infrastructure (IKMI) and Intrusion Detection System (IDS). The IKMI concerns the management of keys and identities of system entities including all SWIM's actors (both human and non-human). The IDS detects if there is intrusion within the network and thereby prevents potential attacks as much as possible. The two corresponding requirements of IKMI and IDS are: *Enable identities and keys management infrastructure* (RQ-E1), and *Enable intrusion detection* (RQ-E2).

The BP controls the connection and information exchange between entities within enterprise systems, and with external entities. The corresponding requirement of BP is *Enable external boundary protection* (RQ-B1).

Domain experts in the field of ATM have foreseen the potential changes in the working environment where the number of actors (both airspace applications and ATM users) might quickly increase. Consequently, both the ISS-ENT and BP might need to scale up with a large number of entities to be managed. Due to such changes, the experts have identified several possibilities that might occur. Then, changes in requirements and components to fulfill requirements are identified in each possibility.

Table 4.2: The evolution of ISS-ENT (including IKMI and IDS), BP, and their possible design alternatives.

Part	Possibility	Belief	Requirements	Alternatives
IKMI	IKMI-P1	10%	RQ-E1	OpenLDAP (C1)
				Oracle Identity Directory (C2)
	IKMI-P2	20%	RQ-E1, RQ-E3	C2
	IKMI-P3	30%	RQ-E1, RQ-E4	C1, Ad-hoc Singe-sign on (C3)
				C2
	IKMI-P4	40%	RQ-E1, RQ-E3, RQ-E4	C2
IDS	IDS-P1	60%	RQ-E2	Host-based IDS (C4)
				Network-based IDS (C5)
	IDS-P2	40%	RQ-E2, RQ-E5	C5
BP	BP-P1	40%	RQ-B1	Ad-hoc BP (C6)
				Common-gateway BP (C7)
	BP-P2	40%	RQ-B1,RQ-B2	C6, C2, Centralized Policy Decision Point (C8)
				C7, C2,C8
	BP-P3	20%	RQ-B1,RQ-B2,RQ-B3	C7, C2,C8

Where: RQ-E1: *Enable identities and keys management infrastructure*, RQ-E2: *Enable intrusion detection*, RQ-E3: *Support scalable IKMI*, RQ-E4: *Support Single sign-on (SSO)*, RQ-E5: *Support scalable, centralized intrusion detection*, RQ-B1: *Enable external boundary protection*, RQ-B2: *Support scalable BP*, RQ-B3: *Provide BP overall security assessment*
Note: all numbers in this table are imaginary for the illustration purpose only.

Table 4.2 summarizes evolution possibilities, changes, and components to fulfill requirements in ISS-ENT and BP. The first column *Part* indicates different parts, namely, IKMI, IDS, and BP, where evolution might occur. Since these parts are relatively independent, we assume the evolution among them is independent. The second column *Possibility* lists possible scenarios (or possibilities) that evolution might happen in future. Each possibility is associated with an expertise belief that the corresponding scenario will happen; this belief is denoted in the third column *Belief.* Notably, all belief numbers in this table are imaginary for the illustration purpose. The two next columns (*Requirements* and *Alternatives*) describe requirements with respect to evolution, and different design alternatives (or implementation choices) to fulfill the requirements. Each alternative occupies one row in the table.

According to Table 4.2, IKMI could evolve to one of the four possibilities, namely IKMI-P1, IKMI-P2, IKMI-P3, and IKMI-P4. The first possibility IKMI-P1 has one requirement RQ-E1 and indicates the scenario where no change happens in future. Its belief is 10%. It has two design alternatives C1 and C2 respectively. In this table, we refer to C*i* as *component.* The second possibility IKMI-P2 indicates a future scenario that has the requirement RQ-E1 and a new requirement *Support scalable IKMI* (RQ-E3). The third possibility IKMI-P3 indicates a scenario that has RQ-E1 and a new requirement *Support Single sign-on (SSO)* (RQ-E4). The fourth possibility IKMI-P4 indicates a scenario that has RQ-E1 and both new requirements RQ-E3 and RQ-E4.

Similarly, the IDS has two possibilities (IDS-P1 and IDS-P2); the BP has three possibilities (BP-P1, BP-P2 and BP-P3). The first possibility of each part indicates a future scenario where no change happens.

As IKMI, IDS, BP are independent, we have in total 24 possibilities to be considered for SWIM. These possibilities are associated with expertise beliefs about the probabilities that they might happen. The decision maker now faces the question: “how to select a design alternative that is better resilient to evolution than others?".

4.2 The AMAN Scenario

The context of this scenario is the evolution in air traffic management procedures planned by the SESAR research programme, which is building the future European air traffic management system. The scenarios focus on the introduction of a new queue management tool, AMAN, and the introduction of a new data transport infrastructure, SWIM, that will replace the current phone-communication lines.

Before the introduction of the AMAN, the flight arrival management operations are performed by the Sector Team composed by two controllers, the Tactical and Planning Controllers. This is done with the support of the CWP (Controller Working Position). The controllers have to compute the arrival sequence for the flights and give clearances for landing to the pilots flying in their sector on the basis of the information displayed by the CWP such as air traffic, radar data, weather condition, etc provided by different ATM actors. The communication among these actors takes places over a dedicated and secure communication line.

After the introduction of AMAN, the AMAN provides support to controllers by automatically generating the arrival sequence. The AMAN may also provide other functionalities, such as generation of advisories for aircrafts, or metering capabilities for a runway, or support runway allocation (at airports with multiple runway configurations). At the organizational level, the introduction of the AMAN requires the introduction of a new type of controller, namely, the Sequence Manager who will monitor and modify sequences generated by AMAN, and will provide information and updates to Sector Team. At the operational level, all ATM actors (including AMAN) communicate via SWIM, a new network for the management and sharing of information. This communication would provide authenticity, integrity and availability that should be comparable with the one provided by the dedicated communication lines (*e.g.,* phone) currently used by controllers.

4.3 Chapter Summary

This section described two application scenarios in the ATM domain. Both were taken from real industrial projects. The first scenario was about SWIM, an information management infrastructure that connects all ATM subsystems from aircrafts to ground facilities. In this scenario we focused on the enterprise security services and boundary protection services. We employed this scenario to illustrate the proposed framework (see chapter 5).

The second scenario was about the AMAN scenario, which we mostly focused on the evolution in requirements model at high level abstraction concerning the introduction of AMAN. We employed this scenario to conduct evaluation studies for the framework (see chapter 8).

In the next chapter, we are going to describe the proposed framework where we discuss how we model evolution in requirements models and the formal semantic of the modeling approach.

Part II

Framework Details

CHAPTER 5

THE PROPOSED FRAMEWORK

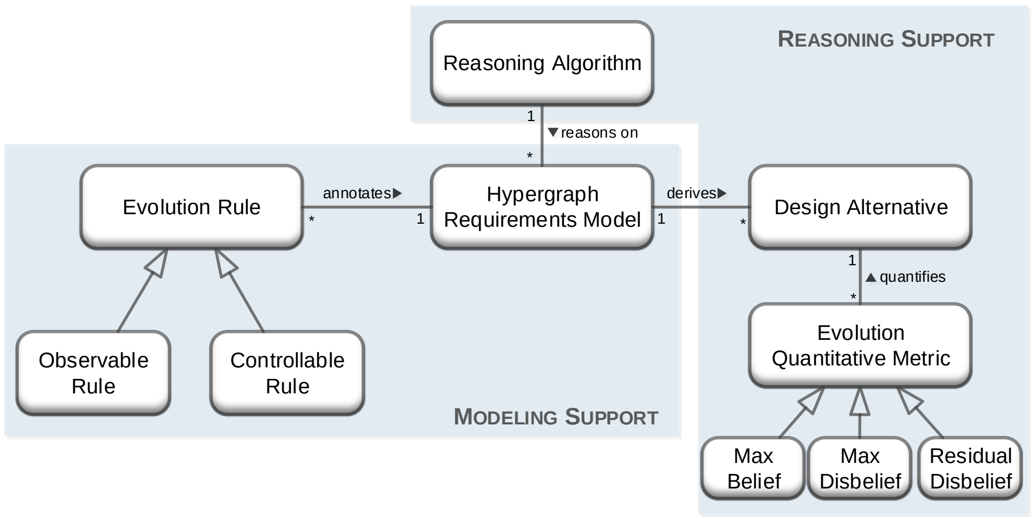
The chapter proposes a framework that tackles the fundamental issue of modeling and reasoning about requirements evolution to aid the selection of evolution-resilient design alternative. The modeling support captures requirements evolution in terms of controllable and observable rules. The reasoning support provides three quantitative metrics to identify which requirements must be implemented to guarantee the best chances of success (Max Belief) or minimize the risk of wasting money (Residual Disbelief and Max Disbelief). The formal semantics of the evolution uncertainty and evolution metrics, as well as the underpinning formal mathematical model of the metrics are also discussed. The applicability of the framework is illustrated with an example from the development of the SWIM platform in Air Traffic Control.

REQUIREMENTS evolution is unavoidable for any long-lived system due to changes in business objectives, regulations, standards, environment or threats. In many cases, these changes are not completely unknown. For instance, the ongoing discussion in a standard body might feature two or three proposals, albeit it might not be clear which one will finally win. Therefore, when having a number of possible design alternatives for the system, decision makers need to select one that is evolution-resilient.

The proposed framework aims at dealing with the uncertainty of the requirements evolution to support the selection of such design alternatives. The intuition is to provide: a *modeling support* mechanism to capture the requirements evolution of the system 'as-is', and an automated *reasoning support* mechanism to aid the selection of an evolution-resilient system design alternative.

Figure 5.1 briefs the conceptual model of the proposed framework. Requirements evolution is modeled by *Evolution Rule*. There are two kinds of rules: *Observable Rule* and *Controllable Rule*. The former kind captures potential changes and their uncertainty (*i.e.,* likelihood of occurrence). The latter kind captures different design alternatives that fulfill requirements. *Hypergraph Requirements Model* is a representation used to express requirements models. The evolution rules are also incorporated into this hypergraph. We introduce several *Reasoning Algorithms* that reason on the hypergraph to derive possible *Design Alternatives* and calculate different *Evolution Quantitative Metrics* for each alternative. There are three quantitative metrics: *Max Belief, Residual Disbelief,* and *Max Disbelief.*

The rest of this chapter is organized as follows. Section 5.1 describes evolution rules to capture requirements evolution. Section 5.2 discusses an interpretation of evolution uncertainty based on game-theoretic approach. Section 5.3 presents evolution metrics and their semantics that support the reasoning on evolution uncertainty. Section 5.4 discusses the formal rule and revised evolution metrics for complex scenarios of evolution in requirements evolution. Finally Section 5.5 summarizes this chapter.



The conceptual model of the framework is illustrated using UML class diagram. The round rectangles denotes entities. The shaded background groups entities for evolution modeling support, and entities for evolution reasoning support.

Figure 5.1: The conceptual model of the proposed framework.

5.1 Modeling the Requirements Evolution

This section describes how we capture the evolution in requirements models for a simple evolution scenario – the *before-after evolution* perspective (see Section 3.1) where an original requirements model could evolve to many possibilities during the study period. More complex evolution scenarios will be discussed later in Section 5.4.

Requirements evolution includes both planned and unplanned changes. The unplanned changes are unexpected, but happen in a system-to-be at a certain time in future (*i.e.,* a system to-be-next). They might happen due to external factors and are not under the control of domain experts or the company who builds the system. These factors could be either changes in regulation, business goals or agreements; or new demands in the market; or new concepts or changes in standards, and so on. Though unplanned changes are out of control, they can be foreseen at some level of uncertainty, which we call *evolution probability*. The evolution probability is the belief by which a change (or changes) might occur in future.

The occurrence of planned changes, as name suggested, is expected. Still it might not be 100% for sure due to some unexpected reasons and/or aforementioned factors. For instance, we may plan to integrate a new function into the SWIM next year, but then we might not do that due to some financial issues. Hence they still have a likelihood of occurrence (*i.e.,* evolution probability), albeit this likelihood is usually high (more than 90%, for example) as they are intentional.

These planned and unplanned changes with their evolution probabilities are captured by *observable rules*, which are described as follows.

Definition 5.1 (observable rule)**.** *An observable rule of a requirements model (or sub model) RM, denoted as $r_o(\textsc{rm})$, is a set of triples (i.e.,* ***evolution possibilities****) $RM \xrightarrow{p_i} RM_i$ where p_i is the probability that RM evolves to RM_i. The possibilities in a rule are complete and mutual exclusive.*

$$r_o(\textsc{rm}) = \left\{ RM \xrightarrow{p_i} RM_i \;\middle|\; p_i > 0 \wedge \sum_{i=1}^{n} p_i = 1 \right\} \tag{5.1}$$

where n is the number of evolution possibilities.

To satisfy the requirements of the system-to-be and potential systems-to-be-next, several design alternatives could be developed. A design alternative and its related concepts are described as follows:

Definition 5.2. *Given a requirements model RM:*

- *A **mandatory requirement** is a requirement that must be satisfied in order to ensure that the system is working properly.*
- *A **design alternative** D is a set of model entities such that their implementation will satisfy all mandatory requirements.*
- *A **primitive design alternative** D^* is a design alternative where removing any member of the set makes it no longer a design alternative.*
- *The **complete set of primitive design alternatives** $\Sigma_{RM}^{D^*}$ is the set of all primitive design alternatives.*
- *The **complete set of design alternatives** is the set of all design alternatives.*

These design alternatives of a requirements model (either for system-to-be or systems-to-be-next) are captured by *controllable rules*, as described as follows.

Definition 5.3 (controllable rule)**.** *A controllable rule of a requirements model (or sub model) RM, denoted as $r_c(\textsc{rm})$, is a set of pairs $RM \rightarrow D_j^*$, where D_j^* is a primitive design alternative of RM.*

$$r_c(\textsc{rm}) = \left\{RM \rightarrow D_j^* \mid D_j^* \in \Sigma_{RM}^{D^*}\right\} \tag{5.2}$$

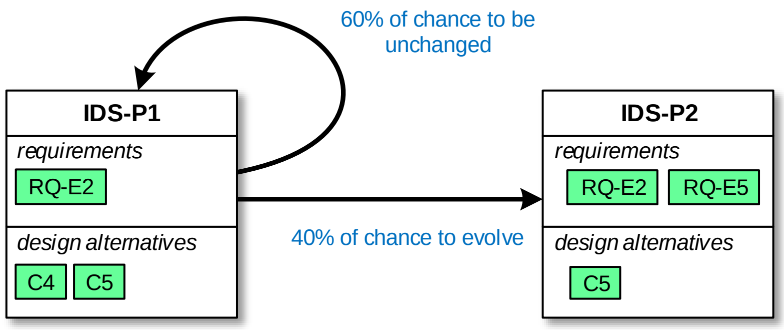
We simplify notation and use the arrow ($\rightarrow$) to denote both observable and controllable rules rather than a different symbol. The label on top with the probability should be enough to distinguish them.

Example 5.1 (evolution rules) Figure 5.2 visualizes the evolution rules of the IDS part within the ISS-ENT (see Table 4.2, Section 4.1). The requirements model is denoted by a rectangle with three compartments that respectively depict the name of the requirements model, a list of its mandatory requirements, and a list of primitive design alternatives.

In this example, the name of requirements model is IDS-P1. It currently has one requirement: RQ-E2. To fulfill this requirement, we have two design alternatives: Host-based IDS (C4), or Network-based IDS (C5). Due to the evolution described in Section 4.1, the IDS could either remain unchanged (first possibility, IDS-P1), or will evolve to a scenario (second possibility, IDS-P2) that has two requirements: RQ-E2 and *Support scalable, centralized intrusion detection* (RQ-E5). To fulfill these requirements, there is one design alternative: C5. The expertise belief for IDS-P1 to occur is 60%, and for IDS-P2 is 40%.

The observable rule corresponding to Figure 5.2 is written as follows:

$$r_o(\textsc{ids}) = \left\{\textsc{ids-p}1 \xrightarrow{0.6} \textsc{ids-p}1, \textsc{ids-p}1 \xrightarrow{0.4} \textsc{ids-p}2\right\}$$



A (part of) requirements model is represented as a rectangle with three compartments: the top for the model name, the middle for list of requirements, and the bottom for list of primitive design alternatives. The solid arrow connects two requirements models (or to the model itself) to indicate the possibility that the source model might evolve to the target model with a certain level of belief (probability).

Figure 5.2: The observable and controllable evolution.

The controllable rules for these possibilities are written as follows:

$$r_c(\text{IDS-P1}) = \{\text{IDS-P1} \rightarrow \{\text{C4}\}, \text{IDS-P1} \rightarrow \{\text{C5}\}\}$$
$$r_c(\text{IDS-P2}) = \{\text{IDS-P2} \rightarrow \{\text{C5}\}\}$$

Importantly, IDS-P1 might also have another design alternative such as $\{\text{C4}, \text{C5}\}$. This design alternative is not a primitive one and belongs to the complete set of design alternatives. However, in the controllable rule we capture only the primitive design alternatives as specified above. ■

5.2 Semantics of Evolution Probability: a Game-Theoretic Interpretation

In the proposed framework, the evolution uncertainty is represented as evolution probabilities captured in observable rules. The high level semantics of evolution probability is the likelihood that an evolution might happen in given circumstances. However, we still need a more detail semantics for evolution probabilities, which turns to be necessary to aid the elicitation of these values. In other words, the semantics behind the claim that *"the probability to have a head (or tail) while tossing a coin is 50%"* is well-known and measurable, which is if

```
Game has $n$ round, each round is about a software component $C_i$
FOR i = 1 to n
    Domain Expert announces $p_i \in [0,1]$
    Company announces decision $d_i$: 1: make now, -1: buy later
    The Market announces $r_i \in [0,1]$
    The cumulative asset value of Company is
        $A_i = A_{i-1} + d_i(r_i - p_i)$
```

Figure 5.3: The protocol of the game explaining the sematic of the evolution probability.

one tosses a coin 100 times, he/she has the head (or tail) of the coin about 50 times. So what does it mean for saying *“the probability of an evolution is 50%"*?

To answer such question, this section describes a game similar to one discussed by Shafer *et al.* [Sha+09] to account for the evolution probability. In this game there are three players: Domain Expert, Company, and Market. The sketch of this game is denoted in Figure 5.3.

In this protocol, Company wants to have software component C_i, and asks the Domain Expert for his opinion about the value of C_i in future. This pricing information also includes the expert’s estimate that the component might turn out to be useless.

- At the first step, Domain Expert announces p_i, which is his opinion about a fair price of C_i. So near-zero value of p_i means C_i will likely be useless, according to Domain Expert.

- At the second step, Company has to select between two options:

  - Either “buy later": Company decides to buy C_i later from suppliers at the Market’s price and sets aside p_i in its books to buy C_i.

  - Or “make now": Company decides to tender in order to make C_i now at price p_i.

- At the third step, Market announces r_i, which is the real price of software component C_i.

At the end of a round, the Company ends up with either a software asset or a monetary asset. The value of the software asset is determined by the market price. Table 5.1 exhibits all possible round-ending cases where Company could either win or lose the round upon to the decision made in the second step.

Table 5.1: The asset value of Company based on different decisions

	“make now" ($d_i = 1$)	“buy later" ($d_i = -1$)
$r_i < p_i$	software costs p_i but values only $r_i < p_i$ (lose)	stashed cash p_i exceeds price r_i on market (win)
$r_i > p_i$	software has value $r_i > p_i$ cost paid to produce it (win)	cash on books $p_i < r_i$ money needed to buy software on market (lose)

The law of large numbers here corresponds to say that if unlikely events happen then Company has a strategy to multiply the value of its assets by a large amount. Suppose $p_i \ll r_i$, then the belief on events that should *not* happen (smaller p_i means that the component should be useless) is wrong: these allegedly unlikely events do indeed happen. By systematically betting against the Domain Expert and making the software at p_i, the Company can multiply the value of its assets. The p_i is the evolution probability.

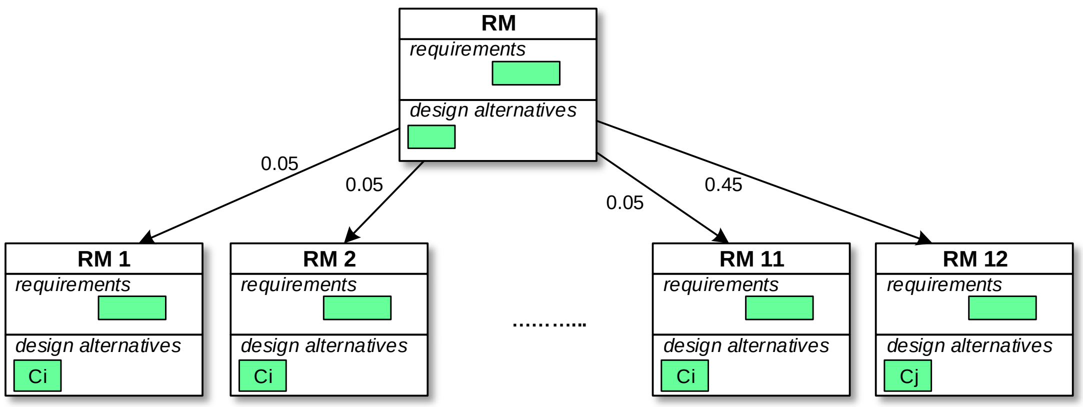
5.3 Semantics of Reasoning about the Evolution Uncertainty

This section introduces three quantitative metrics to evaluate design alternatives with respect to evolution uncertainty, as well as the underpinning mathematical model of these metrics. The three quantitative metrics, namely, *Max Belief*, *Residual Disbelief*, and *Max Disbelief* are as follows.

Max Belief (*MaxB*): measures the maximum belief that a design alternative *D* would be usable after evolution happens. According to the discussed game semantics, Max Belief is the maximum amount of money to allocate if you want to “make now" a call of tender (and trust the domain expert).

Residual Disbelief (*ResD*): is the complement of total belief that a design alternative *D* is usable after evolution happens. It is the (maximum) amount of money that you would be able to save if you postpone the decision (“buy later” so that you do not spend money on potential useless software).

Max Disbelief (*MaxD*): measures the maximum belief such that a design alternative *D* is useless after evolution happens. Max Disbelief partially supports the action “buy later” that postpones the implementation of *D*. It is the max belief in the evolution where *D* is utterly useless. Thus, it could be used as an approximation of Residual Disbelief.



The requirements model RM might evolve to one of twelve possibilities, where RM evolves to RM12 with highest probability 0.45. So, Cj has probability of 0.45 to be useful. Ci occurs in all remaining possibilities, which make for 0.55. However, the probability that an individual possibility where Ci is useful will actually happen is only 0.05.

Figure 5.4: An example of the long tail problem.

The Max Belief and Residual Disbelief might differ substantially in some cases, in particular in presence of a heavy tail of probabilities. This problem, firstly coined by Anderson [And04], is present when a larger than normal population rests within the tail of the distribution.

Example 5.2 (long tail) Figure 5.4 shows an example of long tail where a requirements model RM might evolve to a number of possibilities with very low probabilities (say, eleven possibilities from RM1 to RM11 with 0.05 each), and another extra possibility with a dominating probability (say, RM12 with 0.45). From RM1 to RM11, Ci is a design alternative suitable for all of them. Cj is a design alternative only for RM12. Suppose that Ci and Cj are disjoint each other. According to the metric definitions, Ci is quantified with a triplet of Max Belief, Residual Disbelief, and Max Disbelief: $\langle 0.05, 0.45, 0.45\rangle$; and Cj is quantified with $\langle 0.45, 0.55, 0.05\rangle$. Clearly, if ones' preferences are based on risk minimization, then buying later Ci could be a good choice. Yet this would be missing the opportunity to make now Cj. ■

Some people might put their bets on the long tail [And04], while others do the other way round [Elb08]. We do not take a stand in this debate. Instead, we provide values of both Max Belief (*i.e.*, the head) and Residual Disbelief (*i.e.*, the tail). One can use these values to select the “best-for-him/her" option.

For a requirements model RM with observable rule $r_o(\text{RM}) = \{\langle RM \xrightarrow{p_i} RM_i\rangle | i = 1..n\}$. Let D be a design alternative of RM , we have:

$$\begin{aligned}
MaxB(D) &\overset{def}{=} \max_{\{\langle RM \xrightarrow{p_i} RM_i\rangle \in r_o(RM) | \exists D^* \in \Sigma^{D^*}_{RM_i} : D^* \subseteq D\}} p_i \\
ResD(D) &\overset{def}{=} 1 - \sum_{\{\langle RM \xrightarrow{p_i} RM_i\rangle \in r_o(RM) | \exists D^* \in \Sigma^{D^*}_{RM_i} : D^* \subseteq D\}} p_i \\
MaxD(D) &\overset{def}{=} \max_{\{\langle RM \xrightarrow{p_i} RM_i\rangle \in r_o(RM) | \nexists D^* \in \Sigma^{D^*}_{RM_i} : D^* \subseteq D\}} p_i
\end{aligned} \tag{5.3}$$

These metrics provide a quantitative assessment that supports the selection of design alternatives for a requirements model RM at design time. The optimal design alternative should have a good trade-off between its Max Belief, Residual Disbelief, and Max Disbelief measures. The default implementation for the preference criteria is *"higher max belief, lower residual disbelief and lower max disbelief"*. Such decision orientation can be explained using the game discussed in Section 5.2. In this way, the system implemented by this alternative has a higher chance of success, and minimizes the risk of wasting money with less modification when evolution happens. Risk averse principals might choose another alternative. From an algorithmic perspective it is only important to have a preference relation between triplets of metrics.

Example 5.3 (quantitative metrics) With reference to Table 4.2, Table 5.2 reports all possible combinations of design alternatives of each part and their metrics' values. The first column is the identifier of the global design alternative obtained by combining the corresponding design alternatives of individual parts, which are reported in the next three columns. The last three columns are the metrics' values of a global design alternative. Notably, we eliminate some redundant combinations such as *{*C1, C6, C2, C8*}* because both {C1} and {C2} are design alternatives of IKMI. Therefore it is not necessary to have them both in a global design alternative.

According to Table 5.2, the global design alternatives #5–#12 have the highest Max Belief, the global alternative #12 has the lowest Residual Disbelief, and the lowest Max Disbelief as well. Thus, the alternative #12 is the best one according to the selection criteria discussed previously. ■

Table 5.2: Qualitative metrics for design alternatives of the running example.

No.	Design Alternatives			Max Belief	Residual Disbelief	Max Disbelief
	IKMI	IDS	BP			
1	C1	C4	C6	2.4%	97.6%	9.6%
2	C1	C4	C7	2.4%	97.6%	9.6%
3	C1	C5	C6	2.4%	96.0%	9.6%
4	C1	C5	C7	2.4%	96.0%	9.6%
5	C2	C4	C6	**9.6%**	76.0%	9.6%
6	C2	C4	C7	**9.6%**	76.0%	9.6%
7	C2	C4	C6,C2,C8	**9.6%**	52.0%	6.4%
8	C2	C4	C7,C2,C8	**9.6%**	40.0%	6.4%
9	C2	C5	C6	**9.6%**	60.0%	9.6%
10	C2	C5	C7	**9.6%**	60.0%	9.6%
11	C2	C5	C6,C2,C8	**9.6%**	20.0%	4.8%
12	C2	C5	C7,C2,C8	**9.6%**	**0.0%**	**0.0%**
13	C1,C3	C4	C6	7.2%	90.4%	9.6%
14	C1,C3	C4	C7	7.2%	90.4%	9.6%
15	C1,C3	C5	C6	7.2%	84.0%	9.6%
16	C1,C3	C5	C7	7.2%	84.0%	9.6%

5.4 Formal Rules for Complex Evolution Scenarios

The requirements evolution in real world systems could be very complicated in two dimensions: scale and time. The former appears when the requirements model is too large and complex. It is therefore mostly impossible to analyze the evolution in the entire requirements model as a whole. The latter determines the case that evolution might occur multiple times in a requirements model over time. Additionally, an evolution might depend on the previous occurrence of evolution. This section discusses how these complex evolution scenarios could be addressed by using evolution rules described in previous sections. This section also provides the underpinning mathematical models beyond these scenarios.

5.4.1 Evolution in Large Requirements Model

Requirements models for real systems are typical too large to study evolution in requirements model. The current design principle of software systems typically divides a big requirements model into several sub models (or sub parts) representing separated functional modules. Hence instead of studying evolution in a big requirements model as a whole, we could analyze evolution in its sub models. Evolution in these sub models could be latter combined to represent evolution in the global requirements model. To the simplicity we assume that evolution in a part is independent with evolution in others.

Local evolution rules in independent sub parts could be combined to achieve the global evolution rules for the global requirements model. The rationale of this combination is the effort to reuse the notions of Max Belief, Residual Disbelief, and Max Disbelief without any extra treatment. In the following we discuss how to combine evolution rules from two independent sub parts.

Given two independent sub parts SM and SM' of a requirements model. The evolution rules for these parts are respectively $r_o(SM)$, $r_o(SM')$ (observable rules), and $r_c(SM)$, $r_c(SM')$ (controllable rules) as follows:

$$r_o(SM) = \left\{ SM \xrightarrow{p_i} SM_i \middle| p_i > 0 \wedge \sum_{i=1}^{n} p_i = 1 \right\} \tag{5.4}$$

$$r_o(SM') = \left\{ SM' \xrightarrow{p'_j} SM'_j \middle| p'_j > 0 \wedge \sum_{j=1}^{m} p'_j = 1 \right\} \tag{5.5}$$

$$r_c(SM) = \left\{ SM \rightarrow D^*_t \middle| D^*_t \in \Sigma^{D^*}_{SM} \right\} \tag{5.6}$$

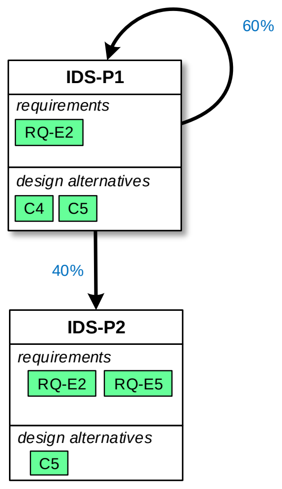
$$r_c(SM') = \left\{ SM' \rightarrow D^*_k \middle| D^*_k \in \Sigma^{D^*}_{SM'} \right\} \tag{5.7}$$

Let RM be the combined model of two parts SM and SM'. The evolution rules of RM are formulated as follows:

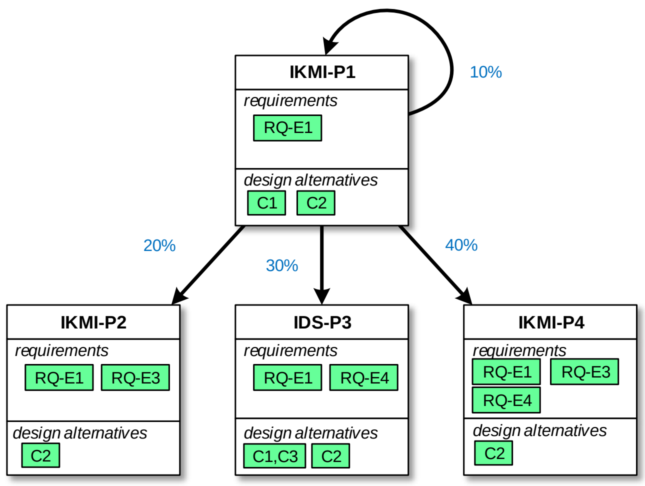
$$r_o(\text{RM}) = \left\{ SM \cup SM' \xrightarrow{p_i \cdot p'_j} SM_i \cup SM'_j \middle| SM \xrightarrow{p_i} SM_i \in r_o(SM) \wedge SM' \xrightarrow{p'_j} SM'_j \in r_o(SM') \right\} \tag{5.8}$$

$$r_c(\text{RM}) = \left\{ SM \cup SM' \rightarrow D^*_i \middle| D^*_i \in \Sigma^{D^*}_{SM \cup SM'} \right\} \tag{5.9}$$

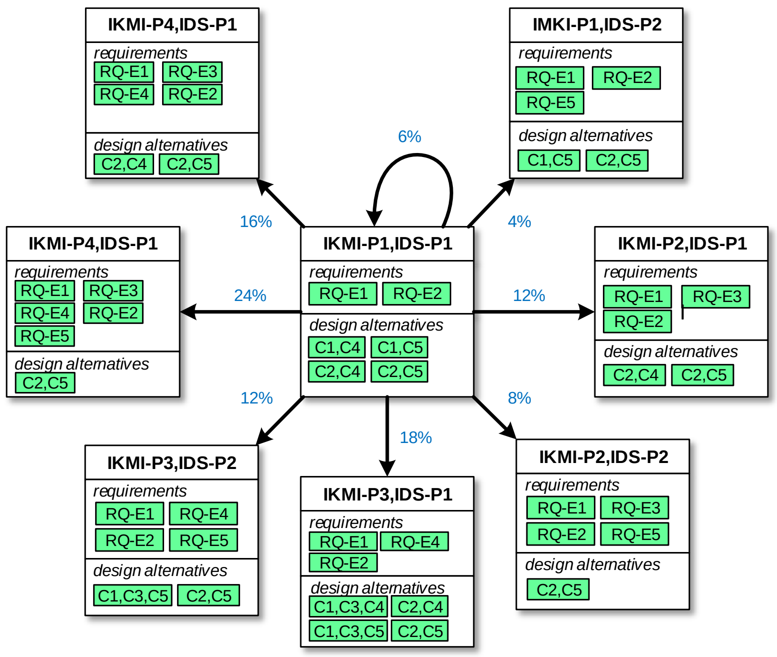
Example 5.4 Recall to Table 4.2, Figure 5.5 illustrates an example of combining evolution rules in two sub parts IKMI and IDS. ■



(a) IDS observable rule



(b) IKMI obervable rule



(c) IKMI+IDS observable rule

Figure 5.5: Example of combining two observable evolution rules.

5.4.2 Continuous Evolution in Requirements Models

The continuous evolution in requirements models is described in the continuous evolution perspective previously discussed in Section 3.1. Here, during the study period, the requirements models keep evolving at multiple time points. We refer to this evolution perspective

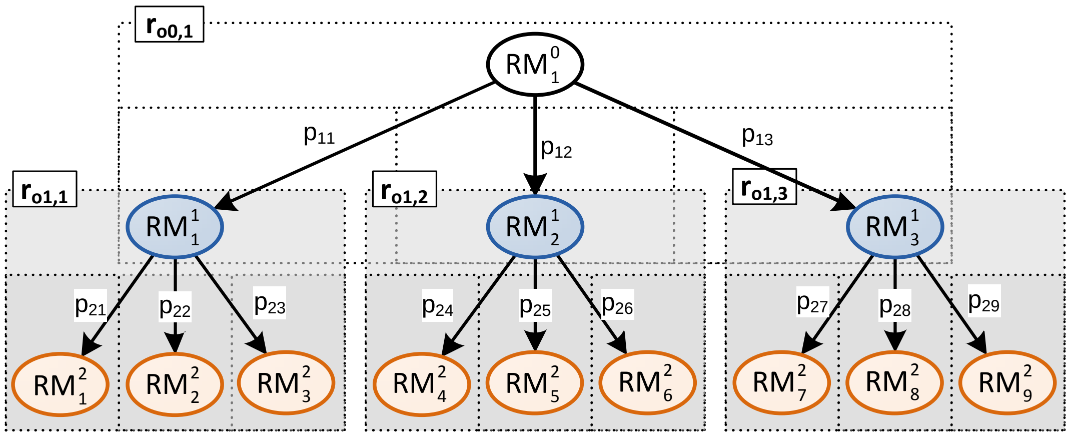


Figure 5.6: Observable rules in a continuous evolution requirements model.

as *multi-step evolution* (or *continuous evolution*) where in each step, the evolution is captured by also using the evolution rules described in Section 5.1. For the sake of brevity, we can assume that there is only one observable rule in a requirements model at a certain time regardless of its complexity. This makes sense because if there are many observable rules in different parts of the model, they eventually could be merged into a global, unique one for the whole model as discussed in Section 5.4.1. Thus the continuous evolution could be captured seamlessly by using the evolution rules where the original requirements model in a step s is one of a possibility requirements model in the accessor step $s-1$.

Example 5.5 Figure 5.6 illustrates a two-step evolution, in which observable rules are denoted as dotted boxes. The original model lays on top part of a box, and all potential evolutions are in sub boxes laid at the bottom. There are directed edges connecting the original model to potential evolutions. The label on each edge represents the probability such that original model evolves to target model. In Figure 5.6, an original requirements model RM_1^0 evolves to either RM_1^1, RM_2^1 or RM_3^1. Subsequently, RM_i^1 evolves to RM_j^2, where i=1..3 and j=1..9. ■

We extend formulae in (5.3) to formulate the calculation of Max Belief, Residual Disbelief, Max Disbelief in the sense of continuous evolution. Given a requirements model RM, and its design alternative D, we respectively denote $MaxB^*(D|RM)$, $ResD^*(D|RM)$, and $MaxD^*(D|RM)$ as the Max Belief, Residual Disbelief, and Max Disbelief of D with respect to the continuous evolution of RM, as the following formulae show.

$$MaxB^*(D|RM) = \begin{cases} \textsf{is-DA}(D, RM) & RM \text{ does not evolve,} \\ \max\limits_{\left\{\langle RM \xrightarrow{p_i} RM_i\rangle \in \textsf{possibility}(D,\ RM)\right\}} p_i \cdot MaxB^*(D|RM_i) & \text{otherwise.} \end{cases} \tag{5.10}$$

$$ResD^*(D|RM) = \begin{cases} 1 - \textsf{is-DA}(D, RM) & RM \text{ does not evolve,} \\ 1 - \sum\limits_{\left\{\langle RM \xrightarrow{p_i} RM_i\rangle \in \textsf{possibility}(D,\ RM)\right\}} p_i \cdot (1 - ResD^*(D|RM_i)) & \text{otherwise.} \end{cases} \tag{5.11}$$

$$MaxD^*(D|RM) = \begin{cases} 1 - \textsf{is-DA}(D, RM) & RM \text{ does not evolve,} \\ \max\limits_{\left\{\langle RM \xrightarrow{p_i} RM_i\rangle \in r_o(RM) \setminus \textsf{possibility}(D,\ RM)\right\}} p_i \cdot MaxD^*(D|RM_i) & \text{otherwise.} \end{cases} \tag{5.12}$$

where

- $\textsf{is-DA}(D, RM)$ is a function that returns 1 if D is a design alternative of RM, or returns 0 otherwise;
- $\textsf{possibility}(D, RM) = \left\{\langle RM \xrightarrow{p_i} RM_i\rangle \in r_o(RM)\ |\textsf{is-da}(D, RM_i)\right\}$ is the set of evolution possibilities of the evolution rule of RM such that D is a design alternative in the evolved models RM_i.

5.5 Chapter Summary

This chapter has presented a new concept that is *evolution rule*. There are two kinds of evolution rules: *observable rule* and *controllable rule*. We employed evolution rules to capture the uncertainty of evolution in requirements model by the concept *evolution probability*. Requirements models in this chapter were treated as high level of abstraction, which were collection of entities and relations. Therefore we could use evolution rules to capture the uncertainty of evolution in requirements model expressing in many RE languages.

The evolution probability were a kind of subjective probability. Its semantics was further explained by a game-theoretic interpretation. Based on evolution probability, we proposed three different evolution metrics: Max Belief, Residual Disbelief, and Max Disbelief to estimate the resilience of a design alternative with respect to evolution. These metrics were useful measures to support the selection of a good evolution-resilient design alternative.

We also discussed two complex scenarios of evolution: evolution in big requirements models, and continuous evolution. In the former, we suggested to study evolution in independent sub-models of the original one then combined them later. In the latter, we extended the formulae of evolution metrics to address the continuous evolution in requirements models.

In the next chapter, we are going to describe a series of algorithms to automate the calculation of proposed evolution metrics for design alternatives of requirements models.

CHAPTER 6

AUTOMATED REASONING SUPPORT

This chapter describes a mechanism to automate the calculation of the quantitative metrics in the proposed framework. For this purpose, we employ hypergraph as a means to capture requirements models where evolution rules are incorporated. Based on this, we develop a series of algorithm to do the calculation. The algorithms support the incremental calculation that automatically reacts on each change made to the requirements model so that users could immediately see the change of metrics at the minimum calculation effort. Moreover, we also present a formal analysis on these algorithms.

THE previous chapter has described the framework to address the *known-unknown* in requirements evolution. The framework has proposed a couple of quantitative metrics, as well as mathematic formulae to calculate them. However, it is almost impossible to do this manually for evolution in a large scale requirements model. Therefore having an automated reasoning support is an essential criterion of success of the proposed framework. This chapter describes a series of algorithms to calculate metric values. These algorithms are designed to support the incremental calculation so that any change made by designers could be immediately reflected in change to metric values in an efficient way.

Moreover, since the proposed framework does not stick to any particular language, we thus need a universal means to represent requirements model in which evolution rules are incorporated. For this purpose, we employ hypergraph as a means to convey requirements models. The hypergraph does not intentionally substitute any RE languages. Hence, it only contains enough information for the reasoning purpose.

This chapter is organized as follows. Section 6.1 presents hypergraph,and how we recast some existing requirements modeling languages to hypergraph. Section 6.2 describes the algorithms to incrementally calculate the quantitative metrics. Section 5.4 analyzes the computational complexity of the algorithms. Section 6.4 provides formal proofs for proposition, lemmas, and theorems.

6.1 Hypergraph Requirements Model

In order to automate the reasoning about requirements evolution, we employ directed hypergraph [Aus+83] as a structure to represent requirements models. With reference to Figure 5.1 (Section 5), the *Hypergraph Requirements Model* is a representation used to express requirements models. A requirements model in an existing modeling language (*e.g.,* KAOS [Lam09a], i*[Yu99], SysML [Sys]) could be transformed into a hypergraph for the execution of the algorithms. This transformation preserves the information about requirements satisfaction, but removes other information to keep the hypergraph simple. In this section, we describe a formalization of hypergraph (Section 6.1.1), and hypergraph equivalences of graphical constructs in some modeling languages (Section 6.1.2).

6.1.1 A Formalization of Hypergraph

A hypergraph, according to [Aus+83], comprises of nodes and hyperarcs. Nodes represent requirements, and hyperarcs represent relations between nodes. We define *refines* relations that connect a source requirement node to a target requirement node; the target requirement is satisfied if the source one is satisfied. If a target requirement requires more than one source requirement in order to be satisfied, we employ an extra *compound node* in the middle. Then a *refines* relation is divided into:

- *component-edge refines* connects a source requirement node to a target compound node.
- *target-edge refines* connects a source requirement node, or a source compound node to a target requirement node.

Then a controllable rule is implicitly represented when a target node is reached by *refines* hyperarc(s) (*i.e.,* the target requirement has various design alternatives). Selecting different incoming arcs means selecting different design alternatives for a target node.

To represent observable rules, we need to introduce a new type of nodes – *observable node*, and a new type of relations – *evolves* relation. An *observable node* bijectively indicates an observable rule; and an *evolves* relation connects an observable node and a requirement node to represent for the occurrence of one evolution possibility.

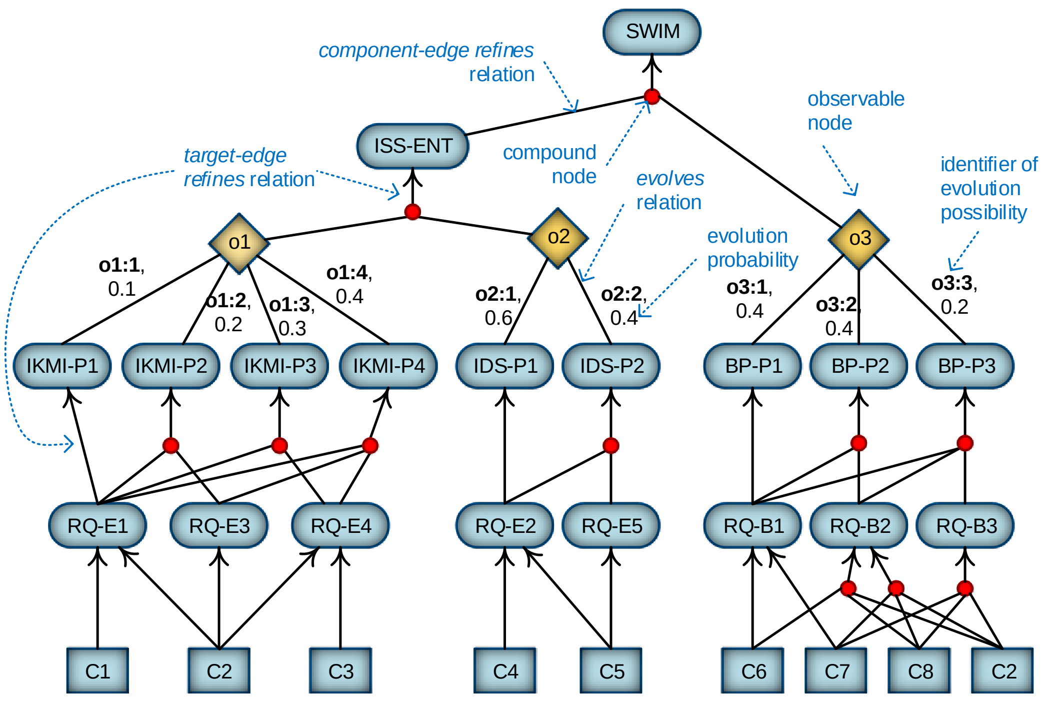
The *observable node* is also able to participate in a *refines* relation as a source requirement node. We then need to extend the concept of the *component-edge refines* and the *target-edge refines* mentioned above. Hence, a *component-edge refines* could connect a source observable node to a target compound node; and a *target-edge refines* could connect a source observable node to a target requirement node.

To this extent, we formally define a hypergraph as follows.

Definition 6.1 (hypergraph requirements model)**.** *A hypergraph requirements model $\mathscr{H}_{RM}$ is a tuple $\langle N \cup N_c \cup N_o, H \cup H_c \cup H_o, root, \mu \rangle$ where:*

- *N is a set of requirements nodes.*
- *N_c is a set of compound nodes.*
- *N_o is a set of observable nodes.*
- *$H \subseteq (N \times N) \cup (N_c \times N) \cup (N_o \times N)$ is a set of* target-edge refines *relations.*
- *$H_c \subseteq (N \times N_c) \cup (N_o \times N_c)$ is a set of* component-edge refines *relations.*
- *$H_o \subseteq N \times N_o$ is a set of* evolves *relations*
- *$root \in N \cup N_o$ is the root node of the hypergraph.*
- *μ is a function that assigns the evolution probability to an* evolves *relation.*

Example 6.1 (hypergraph requirements model) Figure 6.1 presents the hypergraph of SWIM scenario (see Section 4.1). In hypergraph, different graphical notations are used to emphasize the distinction between leaf requirements (rectangles) and others (round rectangles). Leaf requirements might also be referred to as components. Diamonds represent *observable nodes*. To visualize relationships, plain edges are *component-edge refines* relations; arrows are *target-edge refines* ones (target requirements correspond to arrow heads).



Rectangles represent *leaf* requirements; round rectangles denote *intermediate* requirements. Diamonds are observable nodes. Red circles depict compound nodes.

Figure 6.1: The hypergraph requirements model of the SWIM scenario.

The *evolves* relations are also plain edges, but decorated with the evolution probability and the identifier of the evolution possibility to distinct with *component-edge refines* relations. ■

We distinguish each evolution possibility by a unique identifier, which is the combination of the name of the observable node and the index of the evolution possibility in the corresponding observable rule. An example of such identifiers is illustrated in Figure 6.1 as o3:3, where o3 is the name of the observable node.

We revise the definition of a *design alternative* for an arbitrary node in the hypergraph as follows.

Definition 6.2 (design alternative in hypergraph)**.** *Let t be a node in a hypergraph $\mathcal{H}_{RM}$.*

- *D is **a design alternative** of t if and only if D be a set of leaf nodes in $\mathcal{H}_{RM}$ and:*
    - *$t \in D$, or*
    - *$\exists \langle x, t\rangle \in H \cup H_o$ such that D is a design alternative of x, or*
    - *$t \in N_c$ such that for all $\langle x, t\rangle \in H_c$, D is a design alternative of x.*

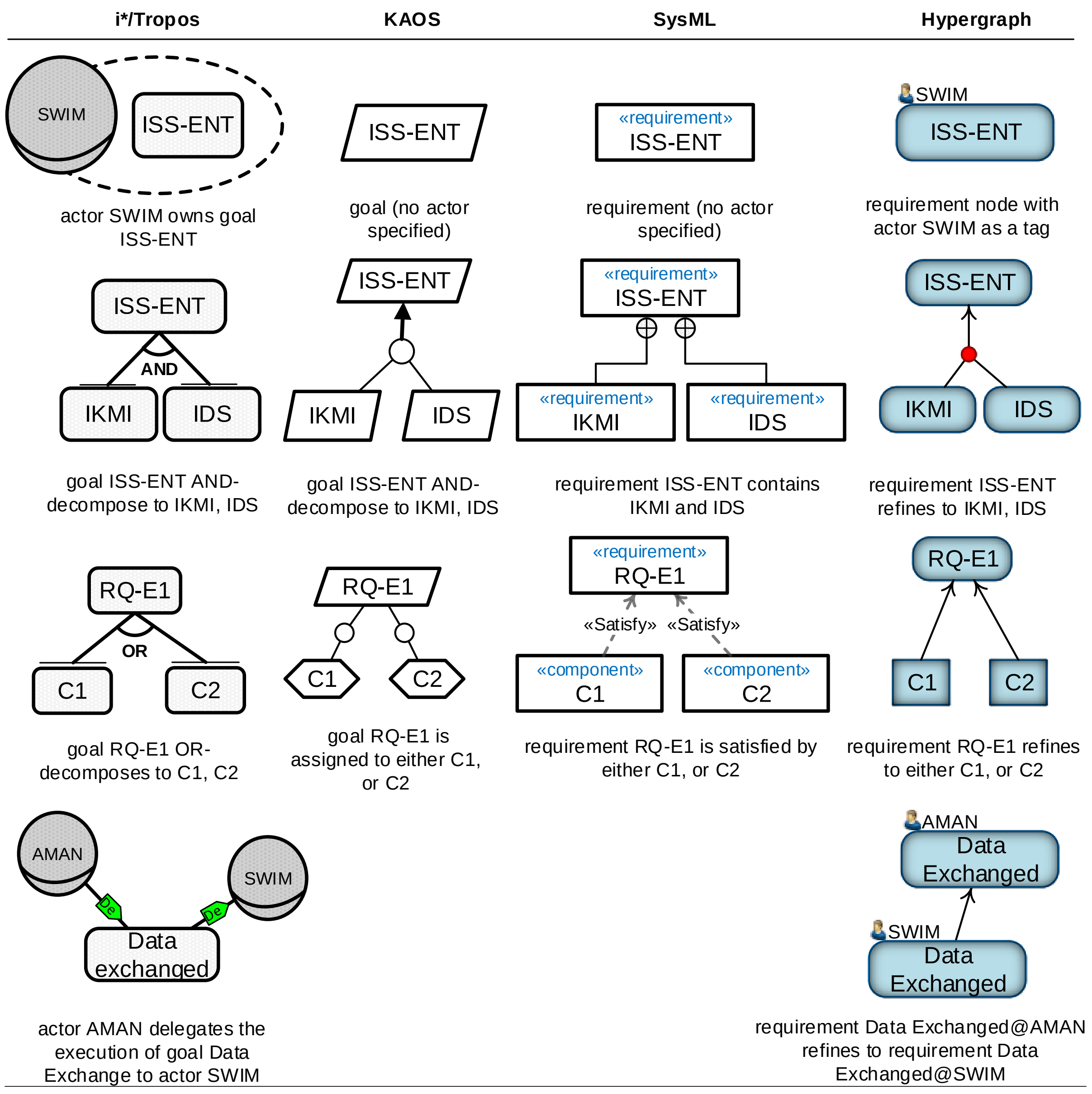
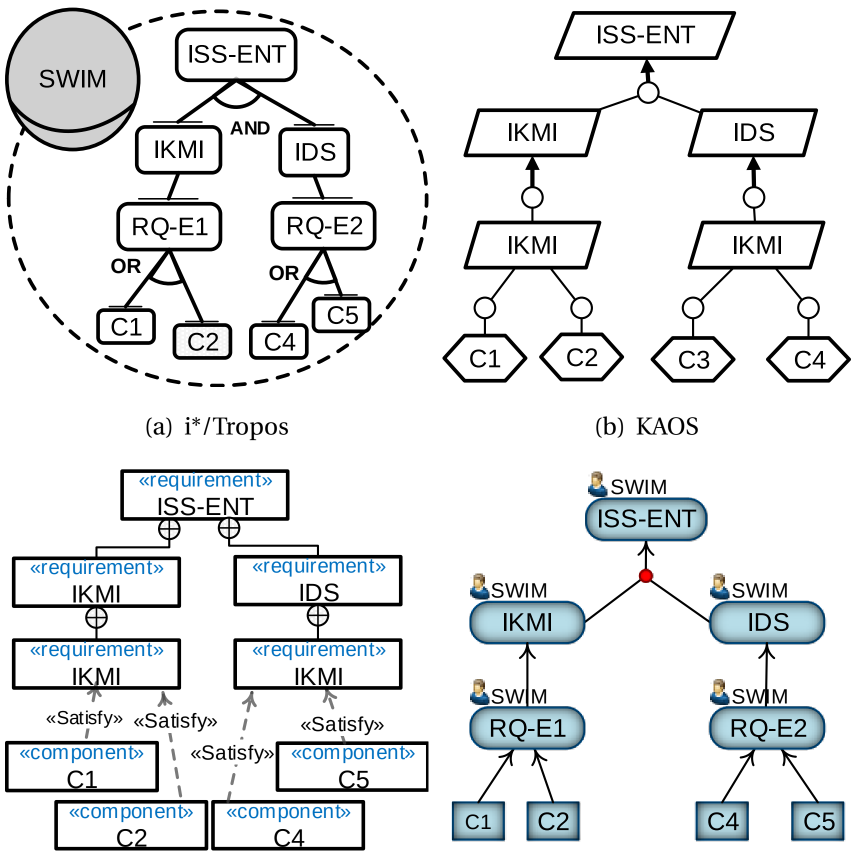

i*/Tropos
KAOS
SysML
Hypergraph
SWIM
ISS-ENT
actor SWIM owns goal ISS-ENT
goal (no actor specified)
«requirement» ISS-ENT
requirement (no actor specified)
requirement node with actor SWIM as a tag
AND
IKMI
IDS
goal ISS-ENT AND-decompose to IKMI, IDS
goal ISS-ENT AND-decompose to IKMI, IDS
requirement ISS-ENT contains IKMI and IDS
requirement ISS-ENT refines to IKMI, IDS
RQ-E1
OR
C1
C2
goal RQ-E1 OR-decomposes to C1, C2
goal RQ-E1 is assigned to either C1, or C2
«Satisfy»
«component»
requirement RQ-E1 is satisfied by either C1, or C2
requirement RQ-E1 refines to either C1, or C2
AMAN
De
Data exchanged
Data Exchanged
actor AMAN delegates the execution of goal Data Exchange to actor SWIM
requirement Data Exchanged@AMAN refines to requirement Data Exchanged@SWIM

Figure 6.2: Some modeling constructs in different modeling languages and their equivalence in the hypergraph.

- *D^* is* ***a primitive design alternative*** *of t if and only if: D^* is a design alternative of t, and no proper subset of D^* is a design alternative of t.*



(a) i*/Tropos (b) KAOS

(c) SysML (d) Hypergraph

Figure 6.3: The requirements model of ISS-ENT modeled by existing languages and by hypergraph.

6.1.2 Hypergraph Representation for Existing Modeling Languages's Constructs

Modeling constructs in different requirements modeling languages can be casted to hypergraph's constructs. In this section, we select three languages i*/Tropos[Yu96], KAOS[Lam09b], and SysML[Sys] to illustrate the equivalences between some constructs of these languages and those of hypergraph.

Figure 6.2 presents a set of transformation patterns of three RE languages: i*/Tropos, KAOS, and SysML to hypergraph. By using these patterns we can convert a requirements model in the these languages to hypergraph. As we only need information concerning the fulfillment of requirements, we transform such information only and ignore unnecessary other information. Concretely, a goal/task (i*/Tropos, KAOS) or a requirement (SysML) is casted to a requirement node in hypergraph. The ownership between an actor and a goal (i*/Tropos) is denoted as a tag associated to the requirement node in hypergraph. *AND-decompose* relations (i*/Tropos, KAOS), and *containment* relations (SysML) are casted to *refine* relations. An *OR-decompose* relation is casted into several *refines* ones. *Assign* relations (KAOS) and *satisfy* relations (SysML) are also casted to *refine* relations.

The transformation from a requirements model to hypergraph could be done automatically via pattern-based transformation engine, the VIATRA2 framework described in [VB07].

As an illustration for the transformation, Figure 6.3 exhibits equivalent forms of the requirements model for ISS-ENT in i*/Tropos (a), KAOS (b), SysML (c), and hypergraph (d).

6.2 Algorithms for Design Alterative Identification and Metric Calculation

The basic idea behind the algorithms is to propagate metric values from leaf nodes to the root node in a hypergraph and cache their value at intermediate nodes. By term *propagate*, we mean a bottom-up approach where we first calculate metrics for leaf nodes, then calculate (or generate) metric values of a parent node based on its children. To be clear, recall to Definition 6.1, in a relation $\langle s,t\rangle \in H \cup H_c \cup H_o$ of a hypergraph, we refer to the source node s as child node, and the target node t as parent node.

This idea allows for an incremental calculation of metrics where there are changes in the graph. It is important for practical usage as we envision the framework to support the design process where the designer experiments with alternatives, or refines evolution probabilities or adding new evolution as additional information from the environment becomes available. We only need to re-calculate the metric values at nodes affected by changes and propagate the new calculation up to the root, while leaving metric values at other nodes untouch. This is essentially better than redoing the whole calculation from ground up.

The intermediate data structure used for the metric calculation is a list of design alternatives annotated with extra information. We refer to this list as *Design Alternatives Table* (*DAT*). The definition of an annotated design alternative is as follows.

Definition 6.3 (annotated design alternative)**.** *An annotated design alternative $\mathcal{D}$ of a node t is a tuple $\langle D, mb, rd, md, \phi\rangle$, where:*

- *D is a design alternative of t,*

- *mb, rd, md are the values of Max Belief, Residual Disbelief, and Max Disbelief of D, respectively.*

- *ϕ is a set of identifiers of evolution possibilities that D supports (i.e., D is a design alternative in these possibilities).*

Each node has its own *DAT* containing all of the annotated design alternatives for itself. As the root node is normally the top requirement for a requirements model, the *DAT* of the root node contains all of the annotated design alternatives to be selected for the system.

Aside from *DAT*, the algorithms use some additional data structures for the calculation such as `VISIT` to maintain the processing status of a node (*i.e.,* 1-processed, 0-otherwise); `READY` to determine whether a node is ready to process (*i.e.,* there is enough data for processing). These data structures are also used to avoid reprocessing a processed node. Table 6.1 summarizes them and a list of algorithms needed for the metric calculation. The metric calculation is divided into following phases:

- *Generating the DATs* (section Section 6.2.1): generate *DATs* for all nodes in a hypegraph by propagating *DATs* from the leaf nodes to the top.

- *Calculating metrics for a design alternative* (Section 6.2.2): calculates metric values for a design alternative based on *DAT* of the root node.

- *Updating the DATs due to incremental changes* (Section 6.2.3): incrementally updates *DATs* with respect to changes in a hypergraph. It avoids from-scratch regeneration for incremental changes made to the hypergraph once *DATs* have been properly generated.

We further discuss the computational complexity of algorithms in Section 6.3.

6.2.1 Generating the DATs

The algorithm `generateDAT()` (see Algorithm 1) takes a hypergraph as input and generates *DAT* for every node in two steps. First, the algorithm sets up ready-to-process status for every node by updating `READY`. Concretely, for every leaf node x, it is immediately ready (*i.e.,* `READY[`x`]` $= 0$) and queued to `Q`. For each of other nodes, its ready-to-process status is set to the number of incoming edges to it. Second, the algorithm invokes `propagateDAT()` to generate all *DAT*s.

Table 6.1: Data structures and algorithms for the metric calculation.

The names of data structures are in uppercase; the names of algorithms are in lowercase.

Name	Description
`VISIT[x]`	maintains the processing status of a node (1 if a node is processed).
`READY[x]`	maintains the readiness-for-process of a node (0 if a node is ready).
`DAT[x]`	maintains the *DAT* of the node `x`.
`Q`	is a queue holding a list of ready-to-process nodes.
`generateDAT()`	generates *DAT* for every node from scratch.
`propagateDAT()`	propagates *DATs* from nodes in `Q` to the root.
`updateDAT()`	handles the incremental changes to update the *DATs* of affected nodes.

```
procedure generateDAT( $\mathcal{H}_{RM}$: hypergraph)
precondition
postcondition
    DAT[x]: is initialized for every node x in $\mathcal{H}_{RM}$
begin
  makeQempty();
  for each node x $\in \mathcal{H}_{RM}$ do
    if x is leaf then
      READY[x] $\leftarrow$ 0;
      enqueue(Q, x);
    else
      READY[x] $\leftarrow$ number of incoming edges to x
  propagateDAT($\mathcal{H}_{RM}$);
end
```

Algorithm 1: Generating *DAT* for every node in a hypergraph.

The algorithm `propagateDAT()` (see Algorithm 2) takes a hypergraph as input, generates *DAT* for queued nodes in `Q`, and propagates these *DAT*s to the root node. To simplify the presentation of the algorithm, following operators are defined:

- $\overline{p}$: the complement value of a probability p.

$$\overline{p} = 1 - p \tag{6.1}$$

```
procedure propagateDAT($\mathcal{H}_{RM}$: hypergraph)
precondition
  Q : contains all ready-to-process nodes.
  READY: is properly set to the readiness of every node.
postcondition
  DAT : contains proper DATs of all nodes.
begin
  markAllNodesUnvisited(); {for every node x, set VISIT[x] ←0}
  while $Q \neq \emptyset$ do
    x ←dequeue(Q);
    if (VISIT[x] = 0) {check whether node is ready, but not visited}
      VISIT[x] ←1; {mark node visited}
      if x is leaf node then
        DAT[x] ←$\{\langle\{x\}, 1, 0, 0, \emptyset\rangle\}$;
      else
        DAT[x] ←$\emptyset$ ;
        {calculate DAT[x] based on incoming edges of x, i.e., x's children}
        if x is an observable node then
          for each $\langle z, x\rangle \in \mathcal{H}_{RM}$ do DAT[x] ←DAT[x] merge $\mu(\langle z, x\rangle)\cdot$ DAT[z];
        else if x is a requirement node then
          for each $\langle z, x\rangle \in \mathcal{H}_{RM}$ do DAT[x] ←DAT[x] $\cup$ DAT[z];
        else {x is a compound node}
          for each $\langle z, x\rangle \in \mathcal{H}_{RM}$ do DAT[x] ←DAT[x] join DAT[z];
      for each $\langle x, t\rangle \in \mathcal{H}_{RM}$ do {outgoing edges of x}
        READY[t] ←READY[t] − 1;
        if READY[t] = 0 then
          enqueue(Q, t);
end
```

Algorithm 2: Propagating DAT in a hypergraph.

- π_n: extracts the nth (1-based) element in a tuple. To improve the readability, we substitute n by a symbolic name. For example, let $\mathcal{D} = \langle D, mb, rd, md, \phi\rangle$ be an annotated design alternative, $\pi_2.\mathcal{D}$ and $\pi_{mb}.\mathcal{D}$ both return the Max Belief value of $\mathcal{D}$.

- $p \cdot DAT(x)$: represents the multiplication of a probability value to all annotated design alternatives in a DAT(x).

$$\begin{aligned} p \cdot DAT(x) = \{ \Big\langle D_i, p \cdot mb_i, 1 - p \cdot \overline{rd_i}, p \cdot md_i, \phi_i \Big\rangle \mid \\ \langle D_i, mb_i, rd_i, md_i, \phi_i \rangle \in DAT(x)\} \end{aligned} \tag{6.2}$$

- consolidate: this operator takes two inputs, $\mathscr{D}$ as an annotated design alternative, and $DAT(x)$ as design alternatives table at node x. The operator updates the metrics' values of $\mathscr{D}$ with respect to $DAT(x)$. We calculate the metrics of $\mathscr{D}$ with reference to their definitions as of (5.3). The consolidate operator is formulated as follows.

$$\mathsf{consolidate}_{DAT(x)}\mathscr{D} \overset{\text{def}}{=} \langle \pi_D.\mathscr{D}, mb, rd, md, \phi \rangle \tag{6.3}$$

in which:

$$\begin{aligned} mb &= \max\left\{\pi_{mb}.\mathscr{D}_i | \mathscr{D}_i \in \mathsf{support}_{DAT(x)}(\mathscr{D})\right\} \\ rd &= 1 - \sum_{rd \in \mathsf{rd}_{DAT(x)}(\mathscr{D})} \overline{rd} \\ md &= \max\left\{\pi_{mb}.\mathscr{D}_i | \mathscr{D}_i \in DAT(x) \setminus \mathsf{support}_{DAT(x)}(\mathscr{D})\right\} \\ \phi &= \bigcup_{\phi_i \in \Phi_{DAT(x)}(\mathscr{D})} \phi_i \end{aligned}$$

where:

- $\mathsf{support}_{DAT(x)}(\mathscr{D}) = \{\mathscr{D}_i \in DAT(x) | \pi_D.\mathscr{D}_i \subseteq \pi_D.\mathscr{D}\}$ is a set of annotated design alternatives in $DAT(x)$, which are subsumed by $\mathscr{D}$. An annotated design alternative $\mathscr{D}_i$ is subsumed by $\mathscr{D}_j$ if $\pi_D.\mathscr{D}_i \subseteq \pi_D.\mathscr{D}_j$.
- $\Phi_{DAT(x)}(\mathscr{D}) = \left\{\pi_\phi.\mathscr{D}_i | \mathscr{D}_i \in \mathsf{support}_{DAT(x)}(\mathscr{D})\right\}$ is a set of identifiers of evolution possibilities where $\pi_D.\mathscr{D}$ is one of their design alternatives.
- $\mathsf{rd}_{DAT(x)}(\mathscr{D}) = \left\{\min \pi_{rd}.\mathscr{D}_i | \mathscr{D}_i \in \mathsf{support}_{DAT(x)}(\mathscr{D}) \wedge \pi_\phi.\mathscr{D}_i \in \Phi_{DAT(x)}(\mathscr{D})\right\}$

We overload the consolidate operator to take one input, a $DAT(x)$. This operator updates the metrics' values of every annotated design alternative $\mathscr{D}$ in $DAT(x)$.

$$\mathsf{consolidate}(DAT(x)) \overset{\text{def}}{=} \left\{\mathsf{consolidate}_{DAT(x)}\mathscr{D}_i | \mathscr{D}_i \in DAT(x)\right\} \tag{6.4}$$

- join: combines two annotated design alternatives $\mathscr{D}_i = \langle D_i, mb_i, rd_i, md_i, \phi_i \rangle$ and $\mathscr{D}_j = \langle D_j, mb_j, rd_j, md_j, \phi_j \rangle$, as the following formula shows:

$$\mathscr{D}_i \,\mathsf{join}\, \mathscr{D}_j = \left\langle D_i \cup D_j, mb_i \cdot mb_j, 1 - \overline{rd_i} \cdot \overline{rd_j}, 0, \phi_i \cup \phi_j \right\rangle \tag{6.5}$$

Note that the Max Disbelief of the result design alternative is set to 0 because this value will be calculated later via the consolidate operator. We also overload the join operator to combine two *DAT*s of two nodes x_1, x_2, namely $DAT(x_1)$ and $DAT(x_2)$ respectively. In this operation, we combine each annotated design alternative of $DAT(x_1)$ to each of $DAT(x_2)$.

$$DAT(x_1)\,\mathsf{join}\,DAT(x_2) \overset{\text{def}}{=} \mathsf{consolidate}\left\{\mathcal{D}_i\,\mathsf{join}\,\mathcal{D}_j | \mathcal{D}_i \in DAT(x_1), \mathcal{D}_j \in DAT(x_2)\right\} \tag{6.6}$$

- merge: merges two *DAT*s of two nodes x_1 and x_2.

$$DAT(x_1)\,\mathsf{merge}\,DAT(x_2) \overset{\text{def}}{=} \mathsf{consolidate}(DAT(x_1) \cup DAT(x_2)) \tag{6.7}$$

In the Algorithm 2, the `propagateDAT()` dequeues a node x from `Q` and processes it. For each dequeued node x, it checks processing status `VISIT[x]` to ensure that every node is processed at most once (line 11–12).

When x is a leaf node, it is the only one design alternative for itself (*i.e.,* we implement it and it is satisfied). Hence, $DAT(x)$ has only one entry, which is $\langle\{x\}, 1, 0, 0, \emptyset\rangle$ (line 13–14). The Max Belief is 1 because of 100% chance of it being the case, both Residual Disbelief and Max Disbelief are assigned to 0 because of 0% chance of it being useless. The last element is an empty set since there is no evolution possibility.

When x is not a leaf node, the algorithm computes the $DAT(x)$ based on the *DAT* of its children (line 17–23) as follows:

$$DAT(x) = \begin{cases} \underset{\langle z,x\rangle \in H_o}{\mathsf{merge}}\, \mu(\langle z,x\rangle)\cdot DAT(z) & x \text{ is an observable node} \quad (6.8a)\\ \bigcup\limits_{\langle z,x\rangle \in H} DAT(z) & x \text{ is a requirement node} \quad (6.8b)\\ \underset{\langle z,x\rangle \in H_c}{\mathsf{join}}\, DAT(z) & x \text{ is a compound node} \quad (6.8c)\end{cases}$$

Note that (6.8a–6.8c) are the net results of the `for` loops at line 19, 21, and 23 respectively because the *DAT* of the parent node is initialized with the empty set at line 16.

Then, the algorithm iterates every parent node t of x to update the ready-to-process status of t by counting down `READY[t]` by 1. If t is ready (*i.e.,* `READY[t]` $= 0$), t is queued to `Q` for processing in subsequence loops, see line 24–27.

Example 6.2 This example shows how to apply (6.8a–6.8c) in the propagation of *DAT*s. We exemplify the *DAT* propagation in a part of the hypergraph represented in Figure 6.1. Concretely, we propagate *DAT* from leaf nodes C2,C6–C8 to the observable node O3. At the beginning, *DATs* of leaf nodes are initialized as follows:

$$DAT(\mathrm{C}i) = \{\langle\{\mathrm{C}i\}, 1, 0, 0, \emptyset\rangle\}, \text{ where: } i \in \{2, 6, 7, 8\}$$

Applying (6.8b), we have $DAT(\text{RQ-B1})$ calculated as follows:

$$\begin{aligned} DAT(\text{RQ-B1}) &= DAT(\text{C6}) \cup DAT(\text{C7}) \\ &= \{\langle\{\text{C6}\}, 1, 0, 0, \emptyset\rangle, \langle\{\text{C7}\}, 1, 0, 0, \emptyset\rangle\} \end{aligned}$$

Applying (6.8c) then (6.8b), we have $DAT(\text{RQ-B3})$ calculated as follows:

$$DAT(\text{RQ-B3}) = \{\langle\{\text{C2,C7,C8}\}, 1, 0, 0, \emptyset\rangle\}$$

Similarly, we have DAT of BP-Pi ($i = 1...3$) calculated as follows:

$$\begin{aligned} DAT(\text{BP-P1}) &= \{\langle\{\text{C6}\}, 1, 0, 0, \emptyset\rangle, \langle\{\text{C7}\}, 1, 0, 0, \emptyset\rangle\} \\ DAT(\text{BP-P2}) &= \{\langle\{\text{C2,C6,C8}\}, 1, 0, 0, \emptyset\rangle, \langle\{\text{C2,C7,C8}\}, 1, 0, 0, \emptyset\rangle\} \\ DAT(\text{BP-P3}) &= \{\langle\{\text{C2,C6,C7,C8}\}, 1, 0, 0, \emptyset\rangle, \langle\{\text{C2,C7,C8}\}, 1, 0, 0, \emptyset\rangle\} \end{aligned}$$

Applying (6.8a), we have $DAT(\text{O3})$ calculated as follows:

$$\begin{aligned} DAT(\text{O3}) = \{&\langle\{\text{C6}\}, 0.4, 0.6, 0.4, \{\text{O3:1}\}\rangle, \langle\{\text{C7}\}, 0.4, 0.6, 0.4, \{\text{O3:1}\}\rangle \\ &\langle\{\text{C2,C6,C8}\}, 0.4, 0.2, 0.2, \{\text{O3:1,O3:2}\}\rangle, \\ &\langle\{\text{C2,C7,C8}\}, 0.4, 0, 0, \{\text{O3:1,O3:2,O3:3}\}\rangle \\ &\langle\{\text{C2,C6,C7,C8}\}, 0.4, 0, 0, \{\text{O3:1,O3:2,O3:3}\}\rangle\} \end{aligned}$$

■

6.2.2 Calculating Metrics for a Design Alternative

When the algorithm `generateDAT()` accomplishes, the *DAT* of the root node holds a list of all possible annotated design alternatives obtained by hypergraph traversing. This facilitates the selection of design alternatives by their metric values.

Besides these alternatives, we are also able to calculate the metrics of an arbitrary design alternative. Let D be a design alternative, $DAT(x_0)$ be the DAT of the root node x_0, the metric values of D are obtained by consolidating an annotated design alternative constructed from D as follows:

$$\langle D, mb, rd, md, \phi\rangle = \mathsf{consolidate}_{DAT(x_0)} \langle D, 0, 1, 1, \emptyset\rangle \tag{6.9}$$

The metric values of D are the corresponding values of the annotated alternative: $MaxB(D) = mb$, $ResD(D) = rd$, and $MaxD(D) = md$. Notably, D could also be a set of arbitrary components. If D is not a valid design alternative in any evolution possibility, the Max Belief and Residual Disbelief of D are 0 and 1 respectively with reference to (6.3).

```
procedure updateDAT($\mathcal{H}_{RM}$ : hypergraph, n : node)
precondition
  DAT : has been generated by propagateDAT().
postcondition
  DAT : is updated with respects to changes in node n
begin
  makeQemtpy();
  markAllNodesUnvisited();
  enqueue(Q, n);
  {firstly, we determine which nodes should have their DAT updated.}
  while $Q \neq \emptyset$ do
    $x \leftarrow$ dequeue(Q);
    if (VISIT[x] = 0)
      VISIT[x] $\leftarrow$ 1;
      for each $\langle x, t \rangle \in \mathcal{H}_{RM}$ do {outgoing edges of x}
        READY[t] $\leftarrow$ READY[t] + 1;
        enqueue(Q, t);
  {secondly, update the DAT of identified node}
  READY[n] = 0; {mark node n ready for recalculating DAT}
  enqueue(Q, n);
  propagateDAT($\mathcal{H}_{RM}$);
end
```

Algorithm 3: Handling incremental changes.

6.2.3 Updating the DATs due to Incremental Changes

Once the *DATs* of all nodes have been properly generated, any changes made to the graph will trigger an incremental update of *DATs*. We develop the algorithm `updateDAT()` (see Algorithm 3) to handle incremental changes in a hypergraph. It takes two inputs: a hypergraph, and a node from which it and its parents need *DATs* updated. There are following kinds of changes in a hypergraph:

- *Add/remove a node*: in the former case, we pass the added node to the algorithm to have its *DAT* generated. In the latter case, all relations from/to the removed node should also be removed. As a result, this case is handled indirectly by handling the removal of these relations.

- *Add/remove/modify an* evolves *relation*: in all cases, we pass the observable node associated with the changed *evolves* relation to the algorithm to update the *DATs* of the observable node and its parents.

- *Add/remove a* refines *relation*: for *component-edge refines* relation, we pass the parent compound node to the algorithm; for *target-edge refines* relation, we pass the parent requirement node.

We do not explicitly consider the kind of changes that users modify a node including requirement node, observable node, and compound node. A node in hypergraph does not have any attributes that directly impact to the calculation of *DATs*. Thus we do not need to handle the modification of nodes in the incremental update of *DATs*. For the similar reason, we also do not consider the modification of *refines* relations in this algorithm.

Similar to the generation of *DAT*, Algorithm 3 also has two steps. For the first step, the algorithm identifies nodes and their number of children, which need their *DATs* updated due to the change (line 7–17). This step starts by visiting the input node n. For a visiting node x, the number of unprocessed children of each x's parent node (*i.e.,* `READY[]`) is increased by 1. Then, the algorithm recursively visits all x's parent nodes. It employs the data structure `VISIT` to ensure that every node in the input hypergraph is visited at most once. For the second step, it invokes `propagateDAT()` to update *DATs*.

6.3 The Complexity of Algorithms

This section analyzes the computational complexity of the algorithms discussed in Section 6.2. To improve the readability of the section, we move all proofs of proposition, lemmas, and theorems in this section to the consecutive one (see further Section 6.4)

While propagating *DAT*s from leaf nodes up to the root node, the number entries of the *DAT* (*i.e., DAT* size) of a parent node is increasing rapidly (linear at observable nodes and requirement nodes, and exponential at compound nodes). Consequently, the complexity of the algorithms described in previous sections is exponential in both time and space.

Proposition 1. *The complexity of the algorithm* `generateDAT()` *and the algorithm* `updateDAT()` *is* $O(n^2 \cdot \max|DAT|^4)$, *where* $|DAT|$ *is the DAT size of a node in the hypergraph.*

Proof. See Section 6.4.1. □

If the input graph forms a k-ary fully connected lattice structure, it could have approximately $\frac{n+1}{k}$ leaf nodes. Thus, the maximum number of design alternatives, which equals

$\max|DAT|$, is $2^{\frac{n+1}{k}}$. As a result, the complexity of the `propagateDAT()` is $O(n^2 \cdot 2^{4n})$. This phenomenon is due to design alternatives competing across different evolution possibilities.

We address this exponential problem by preventing the explosion of the *DAT* size. Instead of keeping all annotated design alternatives in a *DAT*, we only keep annotated design alternatives $\mathscr{D}_i$ of which $\pi_D.\mathscr{D}_i$ is a primitive design alternative within its evolution possibility as the following formula shows:

$$\mathsf{filter}\, DAT(x) = \left\{ \mathscr{D}_i \in DAT(x) \,\middle|\, \forall \mathscr{D}_j \in DAT(x) : \pi_\phi.\mathscr{D}_i \supseteq \pi_\phi.\mathscr{D}_j \rightarrow \pi_D.\mathscr{D}_i \subseteq \pi_D.\mathscr{D}_j \right\} \tag{6.10}$$

The intuition behind this filter is that: first, $\mathscr{D}_i$ supports all possibilities that $\mathscr{D}_j$ supports and more; second, the implementation of $\mathscr{D}_j$ requires all components to implement $\mathscr{D}_i$ plus some other components. Therefore, $\mathscr{D}_i$ is more efficient than $\mathscr{D}_j$. As a result, the maximum *DAT* size is equal to the sum of all primitive alternatives in all evolution possibilities.

Example 6.3 In Example 6.2, the *DAT*(BP-P3) includes two annotated design alternatives: $\langle\{$C2,C7,C8$\}, 1, 0, 0, \emptyset\rangle$ and $\langle\{$C2,C6,C7,C8$\}, 1, 0, 0, \emptyset\rangle$. The former has a primitive design alternative, while the latter has not. Hence we can eliminate the latter from *DAT*(BP-P3). ■

We revise the equation (6.8a–6.8c) to calculate a *DAT* of a parent node based on its children's *DAT*s as follow:

$$DAT(x) = \begin{cases} \mathsf{filter}(\mathop{\mathsf{merge}}_{\langle z,x\rangle \in H_o} \mu(\langle z,x\rangle) \cdot DAT(z)) & \text{for observable node} \quad (6.11a) \\ \mathsf{filter}(\bigcup_{\langle z,x\rangle \in H} DAT(z)) & \text{for requirement node} \quad (6.11b) \\ \mathsf{filter}(\mathop{\mathsf{join}}_{\langle z,x\rangle \in H_c} DAT(z)) & \text{for compound node} \quad (6.11c) \end{cases}$$

The algorithm `propagateDAT()` is rewritten as `propagateDAT*()`, see Algorithm 4, to reflect the new formulation. In Algorithm 4, we change line 19, 21, 23 where we apply the filter per each *DAT*. Hereafter, we define that algorithm `generateDAT*()` and `updateDAT*()` are almost identical to `generateDAT()` and `updateDAT()` respectively. One exception in the revised algorithms is that they call `propagateDAT*()` instead of `propagateDAT()`.

When the total number of primitive design alternatives is still very big, we extend the filter to keep only *winner annotated design alternatives*, and eliminate all others (*i.e., loser annotated design alternatives*). A winner annotated design alternative has one of its metrics'

```
procedure propagateDAT*($\mathcal{H}_{RM}$: hypergraph)
precondition
  Q : contains all ready-to-process nodes.
  READY: is properly set to the readiness of every node.
postcondition
  DAT : contains proper DATs of all nodes.
begin
  markAllNodesUnvisited(); {for every node x, set VISIT[x] ←0}
  while $Q \neq \emptyset$ do
    x ←dequeue(Q);
    if (VISIT[x] = 0) {check whether node is ready, but not visited}
      VISIT[x] ←1; {mark node visited}
      if x is leaf node then
        DAT[x] ←$\{\langle\{x\}, 1, 0, 0, \emptyset\rangle\}$;
      else
        DAT[x] ←$\emptyset$ ;
        {calculate the DAT[x] based on incoming edges of x, i.e., x's children }
          if x is an observable node then
            for each $\langle z, x\rangle \in \mathcal{H}_{RM}$ do DAT[x] ←filter(DAT[x] merge $\mu(\langle z, x\rangle)\cdot$ DAT[z]);
          else if x is a requirements node then
            for each $\langle z, x\rangle \in \mathcal{H}_{RM}$ do DAT[x] ←filter(DAT[x] $\cup$ DAT[z]);
          else {x is a compound node}
            for each $\langle z, x\rangle \in \mathcal{H}_{RM}$ do DAT[x] ←filter(DAT[x] join DAT[z]);
      for each $\langle x, t\rangle \in \mathcal{H}_{RM}$ do {outgoing edges of x}
        READY[t] ←READY[t] − 1;
        if READY[t] = 0 then
          enqueue(Q, t);
end
```

Algorithm 4: Propagating DAT in a hypergraph.

values be the best, *i.e.,* highest Max Belief, lowest Residual Disbelief, or lowest Max Disbelief. Importantly, we consider the win-lose relationship with respect to metrics individually, because debating whether a Max Disbelief winner is better than a Residual Disbelief (or Max Disbelief) winner is similar to the aforementioned long-tail problem that we try to avoid (see Section 5.3).

The intuition behind such filter is that: when we calculate the *DAT* of a parent node by combining *DATs* of its children, the combination of winner annotated design alternatives in child *DATs* will produce a winner annotated design alternative in the parent *DAT* as shown in the following lemmas.

Lemma 1. *Once a DAT of a node is calculated, the DAT and the metrics' values of its annotated design alternatives are unchanged given that the hypergraph does not change.*

Proof. See Section 6.4.2 □

Lemma 2. *When merging two design alternative tables $DAT(x)$ and $DAT(y)$ by using the* merge *operator defined in equation (6.7), a winner annotated design alternative $\mathscr{D}$ in the merged design alternative table corresponds to a winner in the either $DAT(x)$, or $DAT(y)$ if:*

- *$\mathscr{D}$ is a Max Belief winner, or*
- *$\mathscr{D}$ is a Residual Disbelief winner and annotated design alternatives between DAT(x) and DAT(y) are disjoint, or*
- *$\mathscr{D}$ is a Max Disbelief winner.*

Proof. See Section 6.4.3 □

Lemma 3. *When combining two design alternative tables $DAT(x)$ and $DAT(y)$ by using the* join *operator defined in equation (6.6), a winner annotated design alternative $\mathscr{D}$ in the joined design alternative table is always the combination of two winners in $DAT(x)$ and $DAT(y)$ if:*

- *$\mathscr{D}$ is a Max Belief winner, or*
- *$\mathscr{D}$ is a Residual Disbelief winner and annotated design alternatives between DAT(x) and DAT(y) are disjoint, or*
- *$\mathscr{D}$ is a Max Disbelief winner.*

Proof. See Section 6.4.4 □

Therefore, when we filter out all loser alternatives from a *DAT* and keep only winner ones, we can assure that all annotated design alternatives in the *DAT* of the root node are ones that have the best metric values according to Lemma 2, Lemma 3.

Lemma 2 and 3 hold for Max Belief and Max Disbelief winner, however they might not hold for Residual Disbelief winner in the case that annotated design alternatives in child *DATs* are not disjoint. This explains why the lattice structure generates an exponential complexity. In order to obtain a correct result, we need to proceed as follows:

- Before generating *DAT*s, we firstly compute a set of “candidate" solutions (or candidates, for short) for every node in the hypergraph, which are the union of candidates of all child nodes below it. The candidate of a leaf node is the node itself.
- Then, for every observable node, we compute a set of “multiple-run" candidates for this node, which are intersection of candidates of pairwise child nodes.
- For every child node of an observable node, we propagate down the multiple-run candidates of the observable node intersecting with the candidates of the child node. When the child node is an observable node, what is propagated further down is the union of the “local" multiple-run candidates and the multiple-run candidates from the parent.
- Hence, while generating *DATs*, for every node we have a set of multiple-run candidates. In the revised algorithm, each time a filter operation is applied at a node, we keep the winner alternatives and other alternatives that are entirely in the multiple-run candidates of the node. At a result, the *DAT* of every node x has at most $3+2^{|MR(x)|}$ alternatives, which are winners for each metric plus all possible multiple-run candidates.

Theorem 1. *The algorithm* `generateDAT*()` *is polynomial in the number of nodes* $O(n^2)$ *and exponential in* $O(\max|MR(n)|)$*, where* $|MR(n)|$ *is the maximum number of multiple-run candidates.*

Proof. See Section 6.4.5 □

If this is still unsatisfactory, we can use only Max Belief as a criterion to select the winner. We refer to such filter as *1-Max Belief winner* filter operator, as shown as below:

$$\mathsf{filter}_{mb}\, DAT(x) = \{\mathscr{D}_i \in DAT(x) \mid \nexists \mathscr{D}_j \in DAT(x) : \pi_{mb}.\mathscr{D}_j \geq \pi_{mb}.\mathscr{D}_i\} \tag{6.12}$$

Theorem 2. *The algorithm* `generateDAT*()` *using the filter operator as of (6.12) terminates in polynomial time in the number of nodes* $O(n^2)$.

Proof. See Section 6.4.6 □

Theorem 3. *When the algorithm* `generateDAT*()` *terminates, each entry in the DAT of the root node is either a Max Belief winner, or a Residual Disbelief winner, or a Max Disbelief winner.*

Proof. See Section 6.4.7 □

6.4 Proofs of Algorithm Complexity

6.4.1 Proof of Proposition 1

Proposition 1. *The complexity of the algorithm* `generateDAT()` *and the algorithm* `updateDAT()` *is* $O(n^2 \cdot \max|DAT|^4)$, *where* $|DAT|$ *is the size of a DAT of a node in the hypergraph.*

Proof. In `generateDAT()` (Algorithm 1), the complexity of the `makeQempty()` at line 6 is $O(1)$. The `for` loop at line 7 repeats at most n, where n is the total number of nodes in a hypergraph. Then the complexity of this `for` loop is $O(n)$. As a result, the complexity of `generateDAT()` depends on that of `propagateDAT()` (*i.e.*, Algorithm 2). Likewise, we also have the complexity of `updateDAT()` depends on `propagateDAT()`.

Algorithm 2 consists of two loops: one for setting the process status of all nodes (line 8), and another one for processing node (line 9–27). The former iterates every node and sets its visited status to 0. Hence the complexity of this loop is $O(n)$. The latter `while` loop processes all nodes in queue `Q`. Since every node is processed at most once, which is enforced in line 11–12, this `while` loop repeats at most n times. In each loop, the algorithm computes the *DAT* for the processing node x (line 17–23), the propagates this *DAT* to parent nodes (line 24–27).

With reference to (6.6), (6.7), the complexity of the join and merge operators are $O(\max|DAT|^4)$, $O(\max|DAT|^2)$ respectively, where $|DAT|$ is the size of a *DAT*. Asides, a node has maximum $n-1$ connections. Thus, the `for` loops at line 19 (or line 21, or 23) and 24 are at most $n-1$ iterations. Thus, the upper bound of the complexity of the `while` loop is $O(n^2 \cdot \max|DAT|^4)$.

Therefore, the complexity of Algorithm 2 is $O(n^2 \cdot \max|DAT|^4)$. □

6.4.2 Proof of Lemma 1

Lemma 1. *Once a DAT of a node is calculated, the DAT and the metrics' values of its annotated alternatives are unchanged given that the hypergraph does not change.*

Proof. According to Algorithm 2, a *DAT* of a node x is ready to calculate when all its children's *DAT*s are property calculated (see line 26). One it has been calculated, its status is updated to visited (see line 12). Hence its *DAT* will not be touch later due to the check in line 11. □

6.4.3 Proof of Lemma 2

Lemma 2. *When merging two design alternative tables* $DAT(x)$ *and* $DAT(y)$ *by using the* merge *operator defined in equation (6.7), a winner annotated design alternative* $\mathcal{D}$ *in the merged de-*

sign alternative table corresponds to a winner in the either $DAT(x)$, or $DAT(y)$ if:

- *$\mathscr{D}$ is a Max Belief winner, or*
- *$\mathscr{D}$ is a Residual Disbelief winner and annotated design alternatives between DAT(x) and DAT(y) are disjoint, or*
- *$\mathscr{D}$ is a Max Disbelief winner.*

Proof. Let $DAT(z) = DAT(x)\,\mathsf{merge}\,DAT(y)$ be the merged design alternative table of $DAT(x)$ and $DAT(y)$. Suppose that $\mathscr{D}_Z$ is a winner annotated design alternative in $DAT(z)$.

Case 1 ($\mathscr{D}_Z$ is a *Max Belief winner*), we have:

$$\pi_{mb}.\mathscr{D}_Z = \max_{\mathscr{D}_{Zi} \in DAT(z)} \pi_{mb}.\mathscr{D}_{Zi}$$

From (6.3)(6.4)(6.7), we have:

$$\exists \mathscr{D} \in DAT(x) \cup DAT(y) : \pi_{mb}.\mathscr{D}_Z = \pi_{mb}.\mathscr{D} \wedge \pi_D.\mathscr{D}_Z = \pi_D.\mathscr{D}$$

Suppose that $\mathscr{D}$ is not a winner in neither $DAT(x)$ nor $DAT(y)$, it implies:

$$\exists \mathscr{D}' \in DAT(x) \cup DAT(y) : \pi_{mb}.\mathscr{D}' > \pi_{mb}.\mathscr{D} = \pi_{mb}.\mathscr{D}_Z$$
$$\Rightarrow \exists \mathscr{D}_Z{}' \mathsf{consolidate}_{DAT}(z)\mathscr{D}' \in DAT(z) : \pi_{mb}.\mathscr{D}_Z{}' > \pi_{mb}.\mathscr{D}_Z$$

This contradicts to the premise that $\mathscr{D}_Z$ is a Max Belief winner.

Case 2 ($\mathscr{D}_Z$ is a *Residual Disbelief winner*), we have:

$$\pi_{rd}.\mathscr{D}_Z = \min_{\mathscr{D}_{Zi} \in DAT(z)} \pi_{rd}.\mathscr{D}_{Zi}$$

In the case that alternatives are disjoint between $DAT(x)$ and $DAT(y)$, we have:

$$\exists \mathscr{D} \in DAT(x) \cup DAT(y) : \pi_{rd}.\mathscr{D} = \pi_{rd}.\mathscr{D}_Z$$

if $\mathscr{D}$ is not a Residual Disbelief winner, it implies that:

$$\exists \mathscr{D}' \in DAT(x) \cup DAT(y) : \pi_{rd}.\mathscr{D}' < \pi_{rd}.\mathscr{D}$$
$$\Rightarrow \exists \mathscr{D}_Z{}' = \mathsf{consolidate}_{DAT(z)}(\mathscr{D}') : \pi_{rd}.\mathscr{D}_Z{}' < \pi_{rd}.\mathscr{D}_Z$$

This contradicts to the premise that $\mathscr{D}_Z$ is a Residual Disbelief winner.

Case 3 ($\mathscr{D}_Z$ is a *Max Disbelief winner*), we have:

$$\pi_{md}.\mathscr{D}_Z = \min_{\mathscr{D}_{Zi} \in DAT(z)} \pi_{md}.\mathscr{D}_{Zi}$$

Similarly, from (6.3)(6.4)(6.7), we have one of following cases hold.

– Case 3a: $\exists \mathscr{D} \in \{\mathscr{D}_i \in DAT(x) | \pi_D.\mathscr{D}_i = \pi_D.\mathscr{D}_Z\}$ such that:

$$\begin{cases} \pi_{md}.\mathscr{D} = 0, \text{ or} \\ \pi_{md}.\mathscr{D} = \max\left\{\pi_{mb}.\mathscr{D}_j | \mathscr{D}_j \in DAT(x) \wedge \pi_D.\mathscr{D}_j \not\subseteq \pi_D.\mathscr{D}_Z\right\} = \pi_{md}.\mathscr{D}_Z \end{cases}$$

If $\pi_{md}.\mathscr{D} = 0$, obviously X is a Max Disbelief winner. Otherwise, suppose that X is not a Max Disbelief winner, it implies that:

$$\exists \mathscr{D}' \in \left\{\mathscr{D}' | \mathscr{D}' \in DAT(x) \wedge \pi_D.\mathscr{D}' \neq \pi_D.\mathscr{D}_Z\right\} \subset DAT(z) : \pi_{md}.\mathscr{D}' < \pi_{md}.\mathscr{D}$$

This contradicts to the premise that Z is a Max Disbelief winner.

– Case 3b: $\exists \mathscr{D} \in \left\{\mathscr{D}_i \in DAT(y) | \pi_D.\mathscr{D}_i = \pi_D.\mathscr{D}_Z\right\}$. Similarly we also have $\mathscr{D}$ is a Max Disbelief winner.

□

6.4.4 Proof of Lemma 3

Lemma 3. *When combining two design alternative tables $DAT(x)$ and $DAT(y)$ by using the* join *operator defined in equation (6.6), a winner annotated design alternative $\mathscr{D}$ in the joined design alternative table is always the combination of two winners in $DAT(x)$ and $DAT(y)$ if:*

- *$\mathscr{D}$ is a Max Belief winner, or*
- *$\mathscr{D}$ is a Residual Disbelief winner and annotated design alternatives between DAT(x) and DAT(y) are disjoint, or*
- *$\mathscr{D}$ is a Max Disbelief winner.*

Proof. Let $\mathscr{D}_x^w = \langle D_x^w, mb_x^w, rd_x^w, md_x^w, \phi_x^w \rangle$, $\mathscr{D}_x^l = \left\langle D_x^l, mb_x^l, rd_x^l, md_x^l, \phi_x^l \right\rangle$ respectively denote winner and loser alternatives in $DAT(x)$. Similarly, $\mathscr{D}_y^w, \mathscr{D}_y^l$ respectively denote winner and loser alternatives in $DAT(y)$.

Let $DAT(z) = \left\{\mathscr{D}_x \,\mathsf{join}\, \mathscr{D}_y | \mathscr{D}_x \in DAT(x), \mathscr{D}_y \in DAT(y)\right\}$, the application of operator consolidate on DAT(z) will be the output of combining DAT(x) and DAT(y) by using join operator.

$$\mathsf{consolidate}\, DAT(z) = DAT(x)\, \mathsf{join}\, DAT(y)$$

An annotated design alternative in $DAT(z)$ is a combination of either two loser annotated design alternatives (denoted as $\mathscr{D}_{z0}$), or a loser and a winner[1] (denoted as $\mathscr{D}_{z1}$), or two winners (denoted as $\mathscr{D}_{z2}$). We have:

$$\mathscr{D}_{z0} = \left\langle D_x^l \cup D_y^l, mb_x^l \cdot mb_y^l, 1 - \overline{rd_x^l} \cdot \overline{rd_y^l}, 0, \phi_x^l \cup \phi_y^l \right\rangle$$
$$\mathscr{D}_{z1} = \left\langle D_x^l \cup D_y^w, mb_x^l \cdot mb_y^w, 1 - \overline{rd_x^l} \cdot \overline{rd_y^w}, 0, \phi_x^l \cup \phi_y^w \right\rangle$$
$$\mathscr{D}_{z2} = \left\langle D_x^w \cup D_y^w, mb_x^w \cdot mb_y^w, 1 - \overline{rd_x^w} \cdot \overline{rd_y^w}, 0, \phi_x^w \cup \phi_y^w \right\rangle$$

Case 1 (*Max Belief winner/loser*: $mb_x^w > mb_x^l$), we have

$$\begin{aligned} mb_x^l \cdot mb_y^l - mb_x^w \cdot mb_y^w &\le mb_x^l \cdot mb_y^w - mb_x^w \cdot mb_y^w \\ &= mb_y^w (mb_x^l - mb_x^w) < 0 \end{aligned}$$

$\Rightarrow \mathscr{D}_{z2}$ is a winner alternative in DAT(z).

Applying similar proof in Case 1 (Lemma 2) we have $\mathscr{D}_{z2}$ is also a winner in consolidate $DAT(z)$.

Case 2 (*Residual Disbelief winner/loser*: $rd_x^w < rd_x^l$), we have

$$\begin{aligned} 1 - \overline{rd_x^l} \cdot \overline{rd_y^l} - \left(1 - \overline{rd_x^w} \cdot \overline{rd_y^w}\right) &\ge 1 - \overline{rd_x^l} \cdot \overline{rd_y^w} - \left(1 - \overline{rd_x^w} \cdot \overline{rd_y^w}\right) \\ &= \overline{rd_y^w} \cdot (\overline{rd_x^w} - \overline{rd_x^l}) = \overline{rd_y^w} \cdot (rd_x^l - rd_x^w) \ge 0 \end{aligned}$$

$\Rightarrow \mathscr{D}_{z2}$ is a winner alternative in DAT(z).

Applying similar proof in Case 2 (Lemma 2) we have $\mathscr{D}_{z2}$ is also a winner in consolidate $DAT(z)$ if alternatives in $DAT(z)$ are disjoint.

Case 3 (*Max Disbelief winner/loser*: $md_x^w < md_x^l$)

Let $\widehat{\mathscr{D}_x^l} = \{\mathscr{D}_x \in DAT(x) | \pi_D.\mathscr{D}_x \not\subseteq \pi_D.\mathscr{D}_x^l\}$ be the set of annotated design alternatives that are not supported by $\mathscr{D}_x^l$. Similarly, we also have: $\widehat{\mathscr{D}_x^w}, \widehat{\mathscr{D}_{z0}}, \widehat{\mathscr{D}_{z2}}$. We have:

$$\begin{aligned} \widehat{\mathscr{D}_{z0}} &\subseteq (\widehat{\mathscr{D}_x^l} \times DAT(y)) \cup (DAT(x) \times \widehat{\mathscr{D}_y^l}) \\ &= \left\{\mathscr{D}_x \,\mathsf{join}\, \mathscr{D}_y \middle| \mathscr{D}_x \in \widehat{\mathscr{D}_x^l}, \mathscr{D}_y \in DAT(y)\right\} \cup \left\{\mathscr{D}_x \,\mathsf{join}\, \mathscr{D}_y \middle| \mathscr{D}_x \in DAT(x), \mathscr{D}_y \in \widehat{\mathscr{D}_y^l}\right\} \\ \widehat{\mathscr{D}_{z2}} &\subseteq (\widehat{\mathscr{D}_x^w} \times DAT(y)) \cup (DAT(x) \times \widehat{\mathscr{D}_y^w}) \end{aligned}$$

[1]Because join is a commutative operator, $DAT(x)$ join $DAT(y)$ is equivalent to $DAT(y)$ join $DAT(x)$. Therefore, the combination of a loser in $DAT(x)$ with a winner in $DAT(y)$ is similar to the combination of a winner in $DAT(x)$ with a loser in $DAT(y)$.

Hence, when we apply consolidate on *DAT*(z), the Max Disbelief of $\mathscr{D}_{z0}$, and $\mathscr{D}_{z2}$ is as follows:

$$MaxD(\mathscr{D}_{z0}) = \max\Big(\overbrace{\max\{\pi_{md}.\mathscr{D}_x|\mathscr{D}_x \in \widehat{\mathscr{D}^l_x}\}\cdot\max\{\pi_{md}.\mathscr{D}_y|\mathscr{D}_y \in DAT(y)\}}^{a},\\ \underbrace{\max\{\pi_{md}.\mathscr{D}_x|\mathscr{D}_x \in DAT(x)\}\cdot\max\{\pi_{md}.\mathscr{D}_y|\mathscr{D}_y \in \widehat{\mathscr{D}^l_y}\}}_{b}\Big)$$

$$MaxD(\mathscr{D}_{z2}) = \max\Big(\overbrace{\max\{\pi_{md}.\mathscr{D}_x|\mathscr{D}_x \in \widehat{\mathscr{D}^w_x}\}\cdot\max\{\pi_{md}.\mathscr{D}_y|\mathscr{D}_y \in DAT(y)\}}^{c},\\ \underbrace{\max\{\pi_{md}.\mathscr{D}_x|\mathscr{D}_x \in DAT(x)\}\cdot\max\{\pi_{md}.\mathscr{D}_y|\mathscr{D}_y \in \widehat{\mathscr{D}^w_y}\}}_{d}\Big)$$

Since $\mathscr{D}^l_x, \mathscr{D}^l_y$ are Max Disbelief losers, $\mathscr{D}^w_x, \mathscr{D}^w_y$ are winners, we have:

$$\max\left\{\pi_{md}.\mathscr{D}_x \middle| \mathscr{D}_x \in \widehat{\mathscr{D}^l_x}\right\} > \max\left\{\pi_{md}.\mathscr{D}_x \middle| \mathscr{D}_x \in \widehat{\mathscr{D}^w_x}\right\}$$
$$\max\left\{\pi_{md}.\mathscr{D}_y \middle| \mathscr{D}_y \in \widehat{\mathscr{D}^l_y}\right\} > \max\left\{\pi_{md}.\mathscr{D}_x \middle| \mathscr{D}_y \in \widehat{\mathscr{D}^w_y}\right\}$$

It implies: $a > c, b > d \Rightarrow MaxD(\mathscr{D}_{z0}) > MaxD(\mathscr{D}_{z2})$

Similarly, we have $MaxD(\mathscr{D}_{z1}) > MaxD(\mathscr{D}_{z2})$. Therefore, we have $\mathscr{D}_{z2}$ is a Max Disbelief winner.

□

6.4.5 Proof of Theorem 1

Theorem 1. *The algorithm* `generateDAT*()` *is polynomial in the number of nodes* $O(n^2)$ *and exponential in* $O(\max|MR(n)|)$, *where* $|MR(n)|$ *is the maximum number of multiple-run candidates.*

Proof. Similar to the proof of Proposition 1, we have the complexity of `generateDAT*()` depends on the complexity of `propagateDAT*()` (*i.e.*, Algorithm 4).

Algorithm 4 is almost identical to Algorithm 2, except calling to filter() each time child *DATs*are merged or combined. Basically, filter() could be a loop on all entries in a *DAT*. Thus its complexity is $O(\max|DAT|)$ where $\max|DAT|$ is the maximum *DAT* size. Similar to proof of Proposition 1, the complexity of Algorithm 4 is $O(n^2 \cdot \max|DAT|^5)$

As discussed, $\max|DAT| = 3+2^{\max|MR(n)|}$. Hence the complexity of the algorithm is $O(n^2 + n^2 \cdot 2^{5\cdot\max|MR(n)|})$. □

6.4.6 Proof of Theorem 2

Theorem 2. *The algorithm* `generateDAT*()` *using the filter operator as of (6.12) terminates in polynomial time in the number of nodes $O(n^2)$.*

Proof. Similar to the proof of Proposition 1, we have the complexity of the algorithm is: $O(n^2 \cdot \max|DAT|^5)$. Here we select only one Max Belief winner, $\max|DAT| = 1$. Therefore, the complexity of the algorithm is $O(n^2)$. □

6.4.7 Proof of Theorem 3

Theorem 3. *When the algorithm* `generateDAT*()` *terminates, each entry in the DAT of the root node is either a Max Belief winner, or a Residual Disbelief winner, or a Max Disbelief winner.*

Proof. **Base case**: for node x is a leaf, the $DAT(x)$ has one single entry which consists of x itself. This entry is obvious a winner.

Induction case: for node y is a non-leaf node; and y have children nodes x_i such that $DAT(x_i)$ contains all winners. If y is a compound node, the $DAT(y)$ is computed by combining all $DAT(x_i)$ by using join operator. According to Lemma 3, all winners in $DAT(y)$ are composed from winners in $DAT(x_i)$. It means all winners in $DAT(y)$ are actually winners even when all losers are removed from $DAT(x_i)$.

Similarly, if y is a requirement node or an observable node, the $DAT(y)$ is computed by combining all $DAT(x_i)$ by using merge operator. According to Lemma 2, all winners in $DAT(y)$ are composed from winners in $DAT(x_i)$. It means all winners in $DAT(y)$ are actually winners even when all losers are removed from $DAT(x_i)$. □

6.5 Chapter Summary

This chapter presented a series of algorithms that incrementally calculate quantitative metrics in the proposed framework described in chapter 5. The algorithms were incremental as they recalculated the metrics according to any changes made to the requirements model based on the previous calculation. Thus it could minimize the calculation effort for such changes. The algorithms relied on hypergraph as a structure to capture requirements models. The computational complexity of the algorithms were also analyzed and formally proved.

In the next chapter, we are going to present UNICORN, a proof-of-concept prototype of the CASE tool that provides a GUI editor for hypergraph requirements model, and implements the algorithms described in this chapter.

CHAPTER 7

UNICORN: TOOLING AND THE FIRST (SELF) EVALUATION

This chapter presents UNICORN*, a CASE tool for modeling and reasoning about the uncertainty of requirements evolution. The tool provides graphical constructs as well as different views of requirements evolution to assist users to model requirements evolution. The tool also supports the evolution analysis in which facilitate the selection of design alternative. We additionally conduct a performance simulation for the implementation of the algorithms, and a self-evaluation study on a large example.*

THE chapter is organized as follows. Section 7.1 presents an overview of major features. Section 7.2 describes the high level architecture of the tool. Section 7.3 illustrates how the tool apply for the example previous discussed in Section 5.1. Section 7.4 presents a simulation to the performance of the tool. Section 7.5 illustrates the application of the tool in a large example. Section 7.6 summarizes this chapter.

7.1 Features Overview

UNICORN is an Eclipse-based prototype that aims to demonstrate the proposed framework described in Chapter 5 in terms of modeling and reasoning. The tool is provided as a set of EMF-based Eclipse plug-ins written in Java, relying on standard EMF technologies such as

GMF, Xtext. The features of the tool can be categorized into two major categories: *Modeling support* and *Reasoning support*.

The *modeling support* includes features necessary to model requirements evolution. Important features in this category are as follows:

- *Support requirements evolution modeling.* The GUI editor of UNICORN provides several constructs to draw a hypergraph for a requirements model with evolution rules. The structure of hypergraph supported by UNICORN allows users to model the evolution from any requirements from the very top most requirements (*i.e.,* root requirements) to the leaf ones. Figure 7.1 illustrates a simple example consisting of three requirements, where the top requirement evolves to other one.

- *Support different views.* Several views are supported to assist designers. In particular, *Normal View* shows the complete requirements with evolution rules; *Evolution View* presents only evolving parts of the model; and *Original View* displays the requirements model without any evolution.

- *Support large model.* A large requirements model can be partitioned into several sub models. Sub models are edited in separated windows. Each model can reference to other models. Changes in a model will be automatically reflected to other models.

- *Support customization and extension.* The graphical constructs of UNICORN are highly customizable. Adding a new constructs with custom figures and attributes, or modifying existing constructs can be done via configuration files and the plug-in architecture. This enables UNICORN to support the modeling of requirements in other RE languages without changing the UNICORN source code.

Figure 7.1 presents the basic constructs by which we draw requirements models in UNICORN. A requirement entity represents a requirement. A refines relation connects a requirement to other requirement. It means that the parent requirement can be fulfilled if its child is fulfilled. If more than one children are required, these children connect to an extra compound node, which in turn connects to the parent requirement. By allowing several refines relations to connect to a requirement, we can model the different design alternatives of a controllable rule. An observable entity represents an observable rule where the original requirement is connected by an evolves relation. Evolution possibilities are connected by evolution possibility relations. Elements in other diagram could be referenced by special construct off-diagram reference.

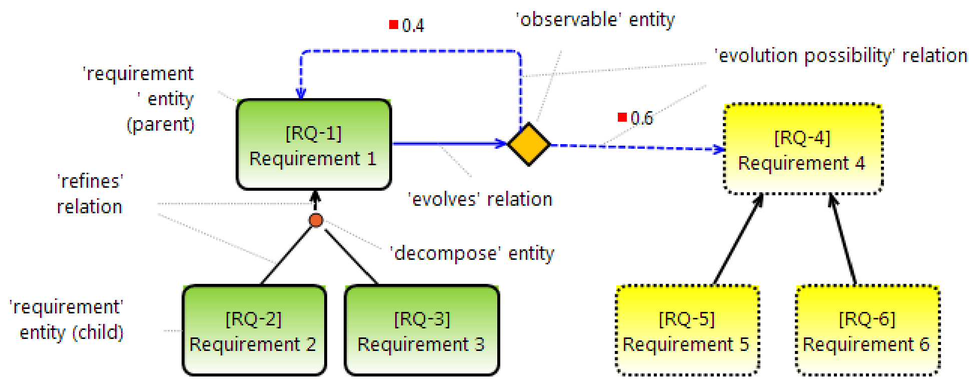


Figure 7.1: The constructs to modeling requirements evolution in UNICORN.

In Figure 7.1, the original requirements model *Before* has three requirements: 1, 2, and 3. 1 is refined to 2 and 3. Therefore, *Before* has one design alternative, which is $\{2,3\}$. *Before* might evolve to a possibility $After_i$ in which 4 is refined to either 5 or 6. The evolution probability for this evolution is 0.6. Besides, *Before* might remain unchanged with the probability of 0.4. The observable and controllable rules captured by this figure are as follows.

$$\begin{aligned} r_o(Before) &= \left\{Before \xrightarrow{0.6} After_i, Before \xrightarrow{0.4} Before\right\} \\ r_c(Before)) &= \left\{Before \rightarrow \{2,3\}\right\} \\ r_c(After_i) &= \left\{After_i \rightarrow \{5\}, After_i \rightarrow \{6\}\right\} \end{aligned}$$

The *reasoning support* provides an environment for developing automated analyses on requirements models. For example, the graphical models could be transformed into a data structure that facilitates the analysis. The traceability between the modeling constructs and transformed data structure is also maintained. Currently, we have implemented following analysis:

- *Evolution analysis*: This analysis walks through the entire requirements models and calculate quantitative metrics (*Max Belief, Residual Disbelief,* and *Max Disbelief*) for each design alternative. The analysis can incrementally update the metric values with respect to changes in the model as soon as the user changes the models.

7.2 Architectural Overview

The tool architecture is specially designed to support a high level of customization and extension. Figure 7.2 presents the overall architecture of the UNICORN tool. In this figure, components are depicted by rectangles. The headed-arrow connections denote the interaction between components where the source components invoke (or use) the target ones. These components are briefly described as follows:

- The *Universal Data Model* is a common storage for the constructs of all models. Since graphical constructs could be defined by users (*e.g.,* add new construct with custom attributes, or add new attributes to existing constructs), the Universal Data Model is a meta-meta model (see Figure 7.3).

- The *Language Registry* maintains definitions of graphical constructs in requirements models, as well as conversion rules to transform the data model to other data structures used by analysis. The construct definitions and conversion rules are defined in configuration files, which are fully customizable.

- The *Data Service* uses the construct definitions in the Language Registry to allow other components to manipulate the data stored in the data model.

- The *GUI Service* is in charge of manipulating graphical objects and data model. It employs the *Custom GUI* components to create several GUI objects (*e.g.,* construct figures, themes, and so on) consumed by the *GUI Editor*, which is a front-end graphical editor.

- The *Model Conversion Engine* uses the conversion rules stored in the Language Registry to convert the requirements model to the underlying data structures used by the custom analysis.

- The *Custom Analysis* is a set of analyses run on the editing requirements model. Each custom analysis has a *Visualizer* to show the analysis result.

Figure 7.3 describes the class diagram of the *Universal Data Model*. The *Element* is an abstract class representing any element in the model, which could be either an entity, or a relation. An *Entity* could be a requirement, or an observable node. A relation captures the relationship between entities. An *Attribute* is a special kind of element, holding the attribute value of an element.

Figure 7.4 shows the syntax of the construct definition file in the Extended Backus-Naur format. To keep the syntax tidy and clear we do not provide complete definition of some non-terminal production rule such as `expression`, as well as common terminal symbols such as identifier (`ID`). The definition file begins with a keyword `language` followed by an `ID`, which

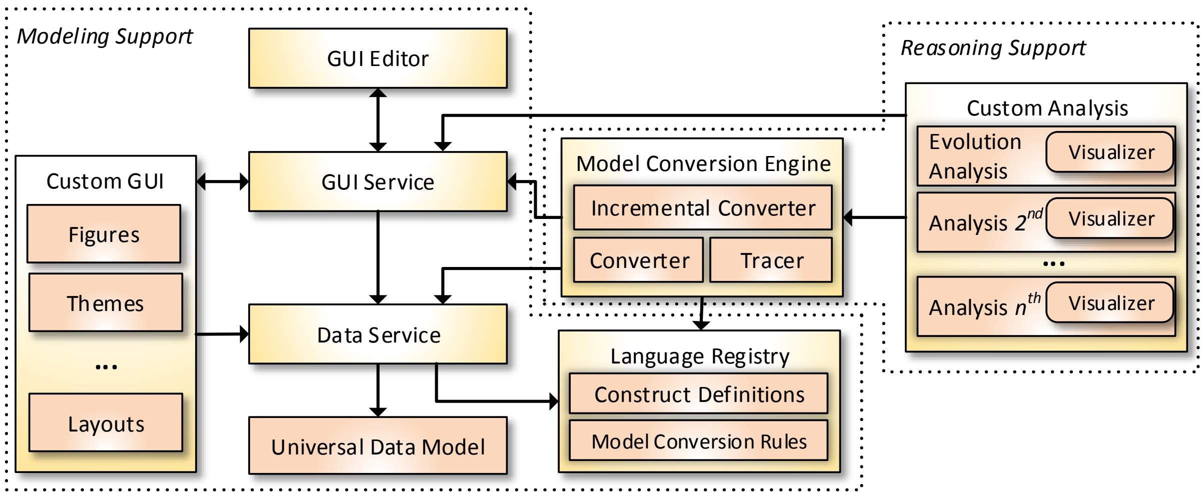


Figure 7.2: The overall architecture of the UNICORN tool.

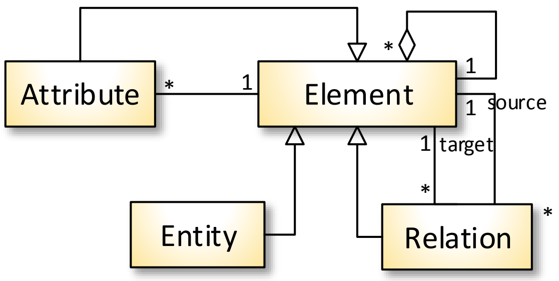


Figure 7.3: The class diagram of the *Universal Data Model*

.

is the language name and a set of elements. An element is either an entity or a relation. Each element has a set of attributes, which has name, data type (optional), and initial value.

Figure 7.5 exhibits an example where the requirement construct is defined with respect to the grammar denoted in Figure 7.4. The requirement construct is an entity whose graphical representation is a round rectangle with a label inside. The label is to show and edit the name and the description of this requirement. There is one text field Actor in requirement. The initial value of this field is a blank.

```
language ::= "language" ID (element)⋆ ";"
element ::= ("entity"|"relation") ID "{" attribute ⋆ "}"
attribute ::= ("field")? ID (":" type)? "=" expression ";"
tag ::= ID ":" expression
type :: = "string"|"float"|"date"|"boolean"|ID
```

Figure 7.4: The compact syntax of the construct definition file.

```
entity requirement {
    figure = new RoundRectFigure() { Size=(50,60),
                addChildFigure(new CenterLabelFigure("req_name"), DOCK_FILL)};
    req_name_parser = {pattern="[{0}]\n{1}", fields=(Name, Description)};
    field Actor: string = "";}
```

Figure 7.5: A fragment of a construct definition file.

7.3 Screen Shots

We demonstrate the features supported by the tool in a scenario previously described in Section 4.1. The scenario concerns the evolution in the requirements models of the ISS-ENT and BP [Adm09, section 5.6]. In this scenario we focus on the authentication and the implementation of boundary protection (BP) services.

Modeling requirements evolution. Figure 7.6 illustrates the requirements model with evolution rules of the scenario. The model says that the requirement RQ-0 is refined to both RQ-1 and RQ-2. RQ-1 is later refined to RQ-3, and so on. RQ-1 has an evolution rule where RQ-1 might remain unchanged with probability 0.4, or might evolve such that RQ-1 will be refined, in a new way, into RQ-3, RQ-4, and RQ-5. The rest of the diagram can be read in the similar manner. Due to space limit, some screen shots (*e.g.*, different views) are not provided. Interested readers are referred to the web site of the tool [1].

Reasoning about requirements evolution. Figure 7.7 shows the evolution analysis on the requirements model of the scenario, in which the evolution metrics for each design alternative are calculated. The analysis result is shown in two tabs. The first tab reports possible alternatives derived from the model and their corresponding evolution metrics. The second tab displays the DAT, which is an internal structure stored at every node in the model to cal-

[1] http://disi.unitn.it/~tran/pmwiki/pmwiki.php/Main/Unicorn

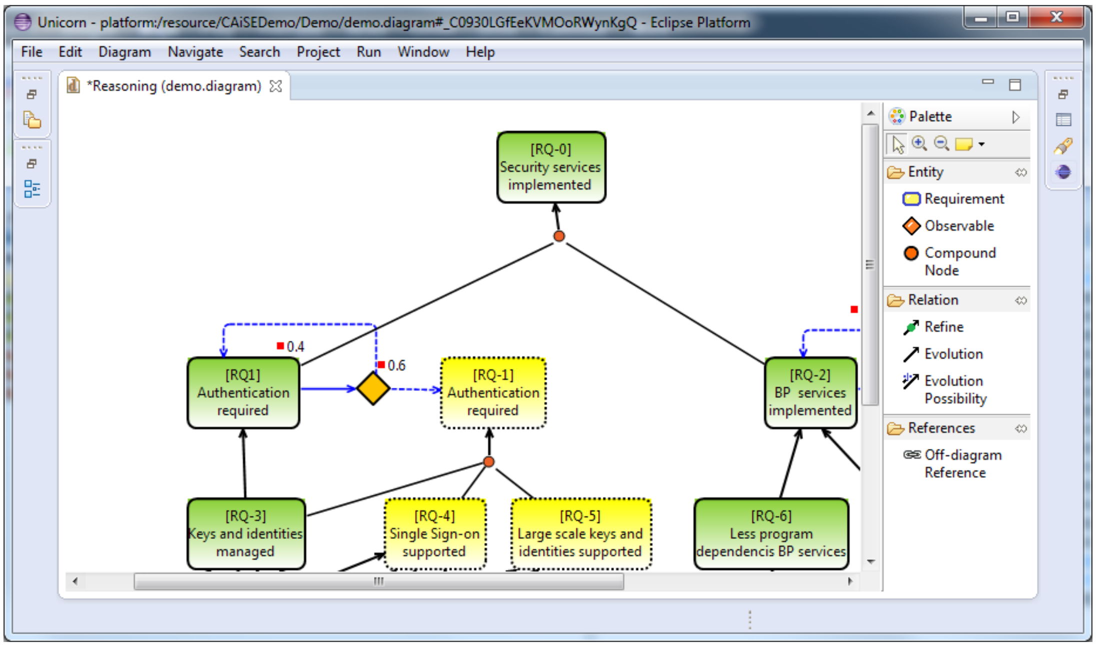
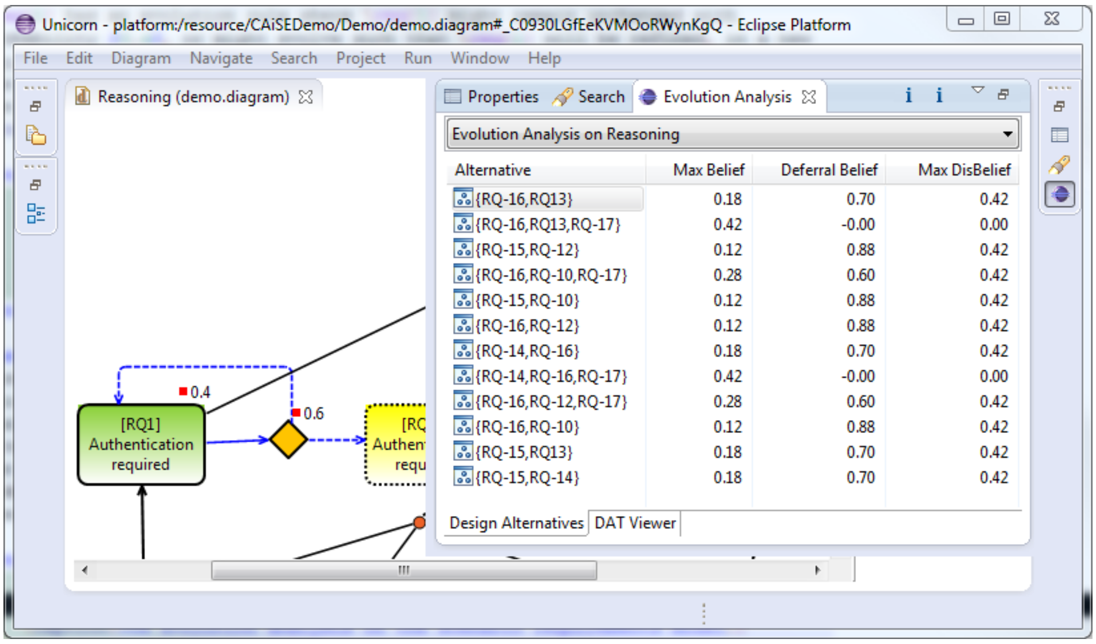

Unicorn - platform:/resource/CAiSEDemo/Demo/demo.diagram#_C0930LGfEeKVMOoRWynKgQ - Eclipse Platform
File Edit Diagram Navigate Search Project Run Window Help
*Reasoning (demo.diagram)
[RQ-0] Security services implemented
[RQ1] Authentication required
0.4
0.6
[RQ-1] Authentication required
[RQ-2] BP services implemented
[RQ-3] Keys and identities managed
[RQ-4] Single Sign-on supported
[RQ-5] Large scale keys and identities supported
[RQ-6] Less program dependencis BP services
Palette
Entity
Requirement
Observable
Compound Node
Relation
Refine
Evolution
Evolution Possibility
References
Off-diagram Reference

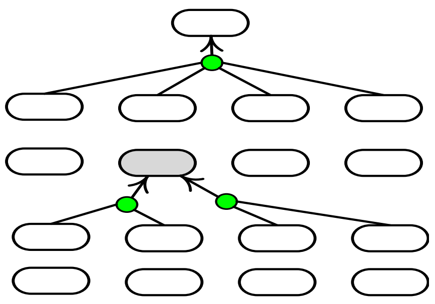
Figure 7.6: The requirements model of the scenario with evolution rules.


Unicorn - platform:/resource/CAiSEDemo/Demo/demo.diagram#_C0930LGfEeKVMOoRWynKgQ - Eclipse Platform
File Edit Diagram Navigate Search Project Run Window Help
Reasoning (demo.diagram)
Properties
Search
Evolution Analysis
Evolution Analysis on Reasoning
[RQ1] Authentication required
0.4
0.6
Design Alternatives
DAT Viewer

Alternative	Max Belief	Deferral Belief	Max DisBelief
{RQ-16,RQ13}	0.18	0.70	0.42
{RQ-16,RQ13,RQ-17}	0.42	-0.00	0.00
{RQ-15,RQ-12}	0.12	0.88	0.42
{RQ-16,RQ-10,RQ-17}	0.28	0.60	0.42
{RQ-15,RQ-10}	0.12	0.88	0.42
{RQ-16,RQ-12}	0.12	0.88	0.42
{RQ-14,RQ-16}	0.18	0.70	0.42
{RQ-14,RQ-16,RQ-17}	0.42	-0.00	0.00
{RQ-16,RQ-12,RQ-17}	0.28	0.60	0.42
{RQ-16,RQ-10}	0.12	0.88	0.42
{RQ-15,RQ13}	0.18	0.70	0.42
{RQ-15,RQ-14}	0.18	0.70	0.42

Figure 7.7: The evolution analysis on the requirements model of the scenario.

culate the evolution metrics. Additionally, users can specify their own alternative, and have its evolution metrics calculated.



To improve the readability, only incoming connections for the root and a selected (gray) node are shown.

Figure 7.8: An example of generated hypergraph for simulation.

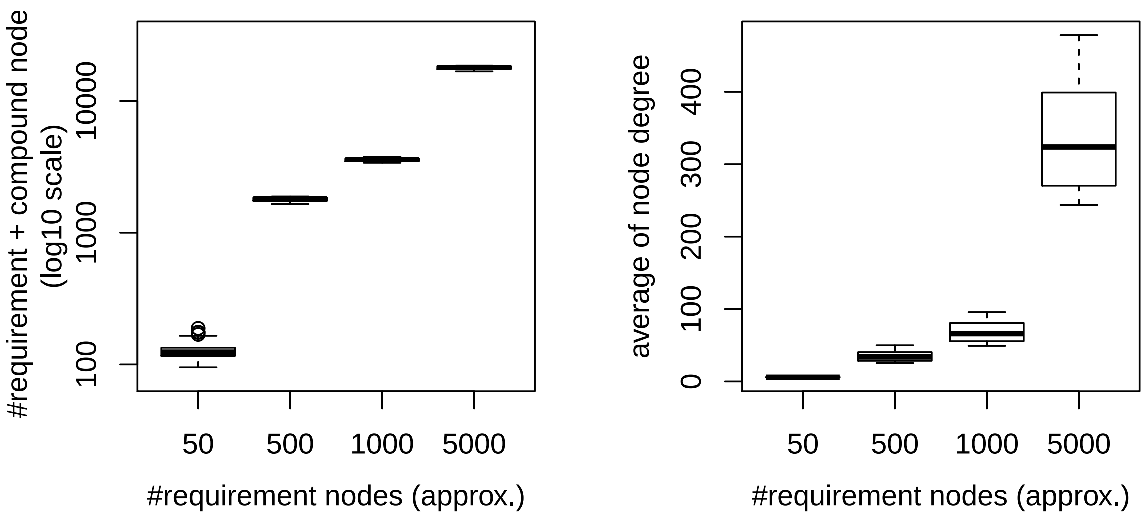
Any changes in the diagram will be automatically reflected in the analysis result. Since the analysis on requirements evolution is incremental, only changed nodes in the model are recalculated. This improves the overall performance of the tool.

7.4 Performance Simulation of the Algorithms

In this section, we describe a simulation to assess the performance of the above algorithms. Particularly, we randomly generate several hypergraphs, then we run the algorithms on these hypergraphs and measure the execution time. The performance of the algorithms can be evaluated by the execution time of the algorithms on hypergraphs with different complexities.

The complexity of a generated hypergraph is represented by the number of requirements n, the maximum level of refinement (or depth) h, and average degree of nodes d. To the simulation purpose, we generate hypergraphs in different scales: small (50 requirements), medium (500 requirements), large ($1,000$ requirements), and very large ($5,000$ requirements). For each scale, we generate 40 hypergraphs with the level of refinement h ranged from 3 to 10.

A generated hypergraph at scale of n requirement nodes and h levels of refinement is generated as follows. Firstly, we create a matrix $(h-1) \times l$ of requirement nodes, where l is the number of leaf nodes: $l \approx \frac{(n-1)}{(h-1)}$. Secondly, we create a root node that refines to all nodes in the first row of the matrix. Thirdly, for every node x at row i^{th} $(i = 1..h-2)$, we create r *refines* relations (r is a random number from 1 to l) that connect to l nodes at row $i+1$.



Left: the distribution of total nodes (requirement + compound nodes) in hypergraphs. Right: the distribution of the average degree per node in hypergraphs.

Figure 7.9: The complexity of the simulation hypergraphs.

Example 7.1 Figure 7.8 exemplifies a generated hypergraph at scale $n = 40$, and refinement level $h = 6$. The hypergraph is the matrix 4×4 of requirement nodes. The figure also demonstrates a node with 2 *refines* relations ($r = 2$). ■

Figure 7.9 shows the complexity of generated hypergraphs in terms of total nodes (include requirement nodes and compound nodes) per each scale, and the average degree (*i.e.*, the number of incoming and outgoing connections) per node in hypergraphs.

Figure 7.10 reports the simulation performance of the algorithms on generated hypergraphs in terms of the execution times. The simulation is performed on a Windows 7 machine with duo-core 2.7GHz CPU, and 6GB of RAM. For small and medium models (the number of requirements is around 500) the execution time is almost immediately (less than 1 seconds). For large models (the number of requirements is around 1,000), it is a bit longer, but still quite fast (less than 3 seconds). The execution time significantly increase for very large models (the number of requirements is around 5,000) where the total number of nodes (requirements and compounds) is more than 10,000 and the average degree per node is more than 300. However, the absolute execution time is still relative fast (approximately 10 section for 50% of the cases). Clearly, it is an evidence that the algorithm has very good performance with respect to the complexity of hypergraphs.

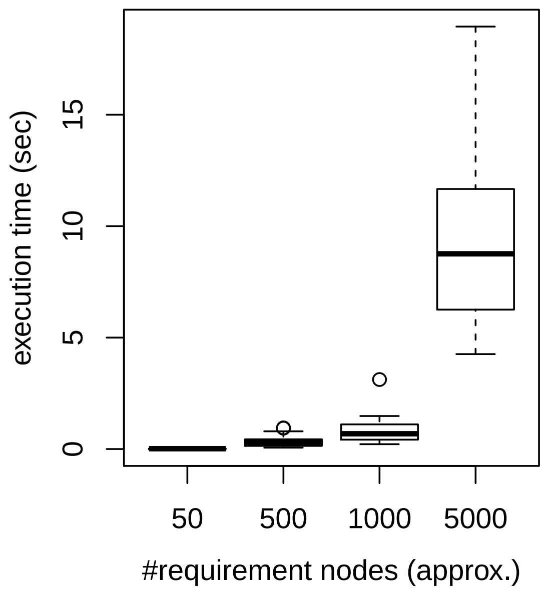


Figure 7.10: The execution time of the algorithms on simulation hypergraphs.

7.5 A Self-Evaluation Case Study

In this section we conduct a case study for the tool in both modeling and reasoning about evolution in requirements model. Notably, the term *case study* here is as same as the one used by other RE researchers, but is not the one defined in the field of empirical evaluation.

In this case study, we take a big requirements model in Si* language (approximately 150 goals), transform this model into hypergraph , and run the algorithms to enumerate design alternatives and calculate their evolution metrics. We then report the execution time of the algorithms. The details of this self-evaluation are described as follows.

Scenario. The scenario is from the ATM domain, which concerns the management and resolution of conflicts in the trajectories of aircrafts. The scenario is briefed as follows. Each flight has its own trajectory in the airspace, called Reference Business Trajectory (RBT). Occasionally, there is conflict among these RBTs. Upon conflict detection, the Air Traffic Service Unit (ATSU) notifies the conflict to all downstream ATSUs and negotiates for a new RBT alternative provisional. Finally, ATSU notifies the involved aircrafts the new RBT alterantive.

In this evaluation we mostly focus on the evolution of security requirements in the above scenario.

Table 7.1: Descriptive statistics of the hypergraph.

	Number	Min	Max	Average
Number of Nodes	178	–	–	–
Depth	26	–	–	–
Number of observable rules	5	–	–	–
Number of controllable rules	16	3	4	3.25
Number of OR branches	27	2	5	2.45
Number of design alternatives	864	–	–	–

Modeling. The modeling includes two phases. At first we model the requirements from this scenario using Si*. Here, instead of working on a very big model with unreadable text, we divide the model into sub models of requirements assigned to particular actors in the system. The Si* diagrams for the requirements model of the scenario are illustrated in Figure 7.11–7.18.

In the next phase, we analyze potential changes of the requirements model of the scenario to identify evolution rules. Then we apply the proposed framework to model evolution. We manually transform the requirements model plus evolution rules into a hypergraph requirements model. We use UNICORN to model this hypergraph. Also, we divided the hypergraph requirements model into several sub diagrams. Cross-diagram references are done via *Off-diagram reference* constructs. Figure 7.19–7.26 present these diagrams.

Reasoning. We run the evolution analysis implemented in UNICORN on the hypergraph. Table 7.1 presents some descriptive statistics of the hypergraph. The analysis take approximately 4 seconds on a 2x2.2 GHz Windows 7 machine with 6MB of RAM. A variant implementation of the algorithms, which keeps only winner alternatives (see Section 6.3), takes about 0.2 second on the same machine. Both implementations suggest 11 design alternatives with most optimal evolution metrics values (*i.e.*, highest Max Belief, lowest Residual Disbelief, and lowest Residual Disbelief).

7.6 Chapter Summary

We have presented UNICORN, a tool for modeling and reasoning about requirements evolution. By modeling support, UNICORN provided several customizable graphical constructs to model the requirements evolution. By reasoning support, UNICORN provided an environ-

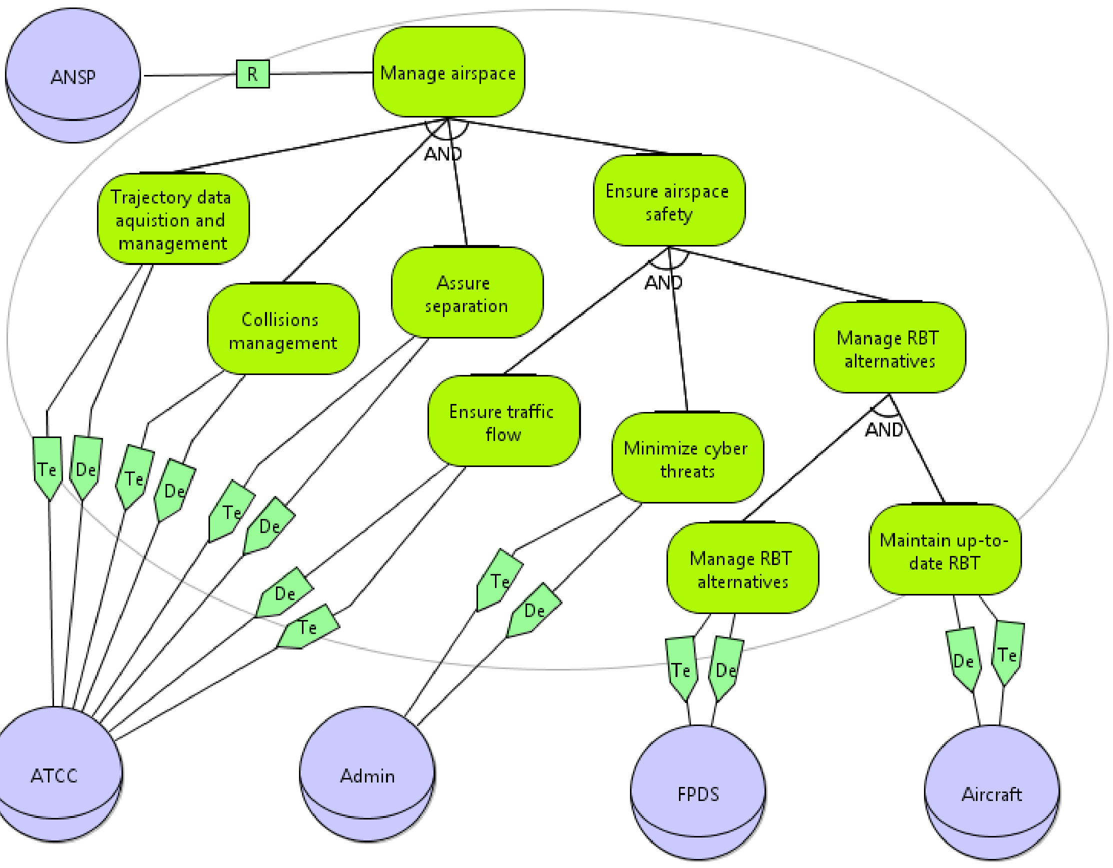


Figure 7.11: Requirements diagram for Airspace Navigation Service Provider (ANSP) actor.

ment where the graphical notation could be transformed to a data structure facilitating the analysis. UNICORN demonstrated this by implementing an analysis for requirements evolution.

To continue evaluating the proposed framework, in the next chapter we study the effectiveness of the proposed framework, particularly the modeling approach by conducting a series of empirical studies with many kinds of participants who are different levels of expertise knowledge in both ATM domain and the proposed framework.

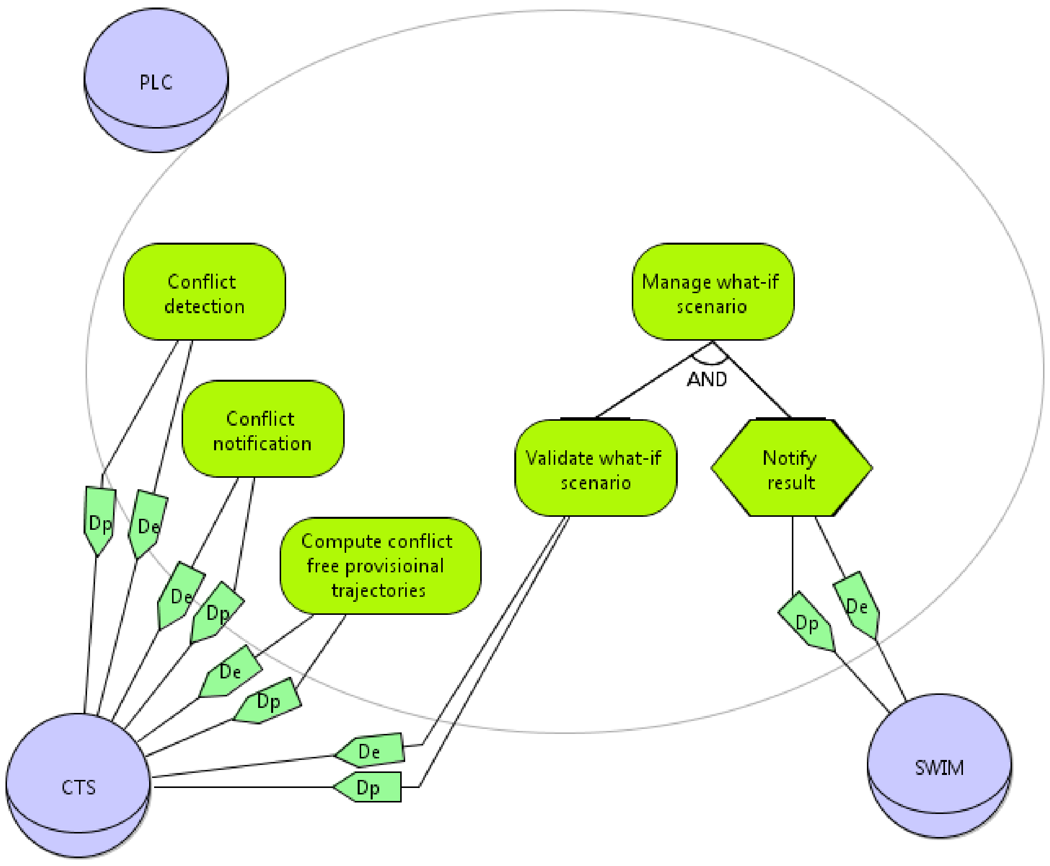


Figure 7.12: Requirements diagram for Planning Controller (PLC) actor.

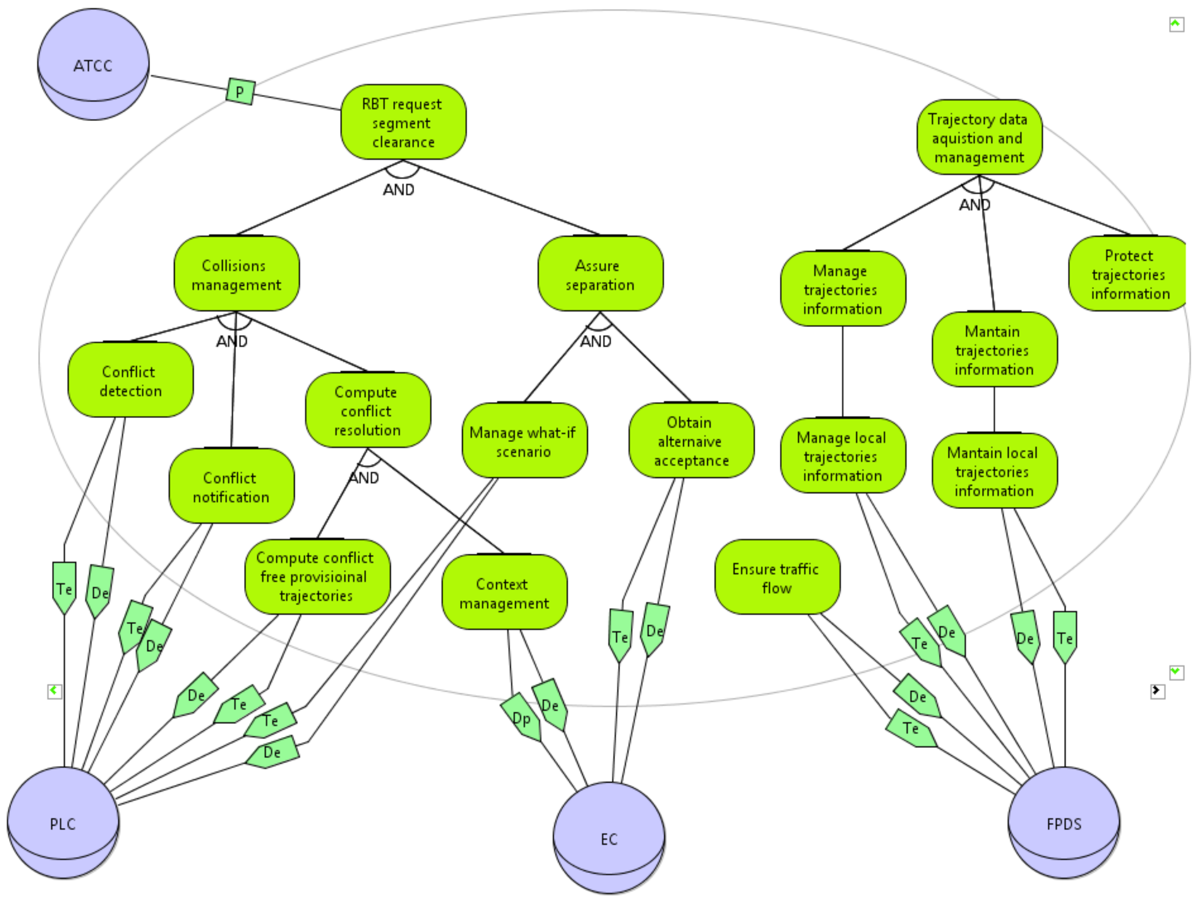

ATCC
P
RBT request segment clearance
AND
Collisions management
Assure separation
Trajectory data aquistion and management
AND
Manage trajectories information
Protect trajectories information
Mantain trajectories information
AND
Conflict detection
Compute conflict resolution
AND
Manage what-if scenario
Obtain alternaive acceptance
Manage local trajectories information
Mantain local trajectories information
Conflict notification
AND
Compute conflict free provisioinal trajectories
Context management
Ensure traffic flow
Te
De
Dp
PLC
EC
FPDS

Figure 7.13: Requirements diagram for Air Traffic Control Center (ATCC) actor.

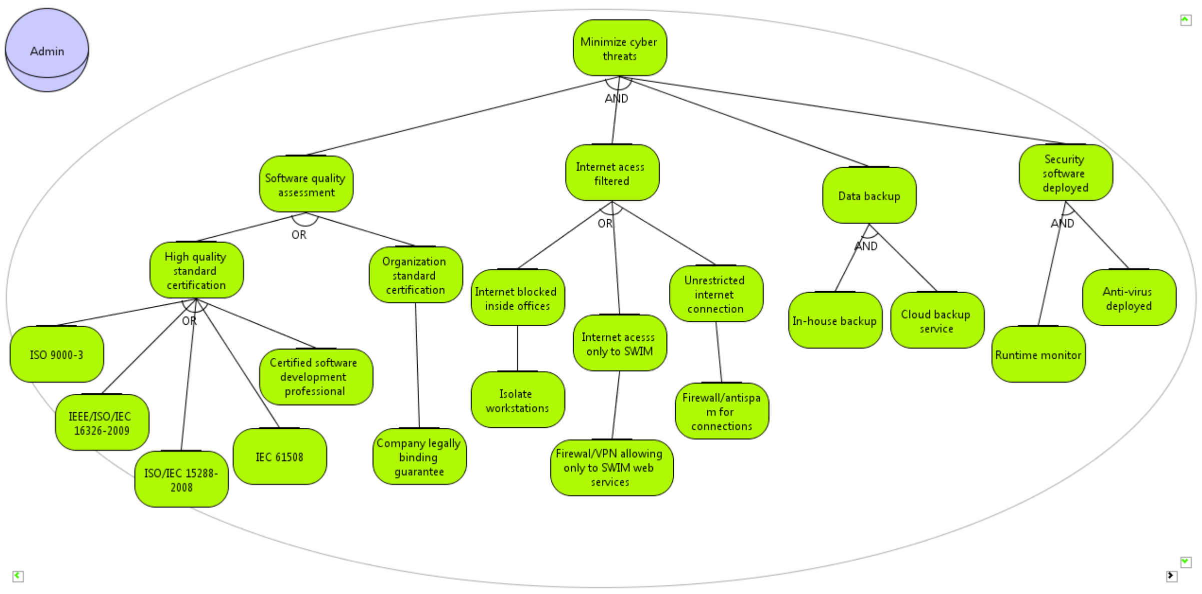

Admin
Minimize cyber threats
AND
Software quality assessment
Internet acess filtered
Data backup
Security software deployed
OR
High quality standard certification
Organization standard certification
OR
Internet blocked inside offices
Unrestricted internet connection
Internet acesss only to SWIM
AND
In-house backup
Cloud backup service
AND
Anti-virus deployed
Runtime monitor
OR
ISO 9000-3
Certified software development professional
IEEE/ISO/IEC 16326-2009
ISO/IEC 15288-2008
IEC 61508
Company legally binding guarantee
Isolate workstations
Firewall/antispam for connections
Firewall/VPN allowing only to SWIM web services

Figure 7.14: Requirements diagram for Admin actor.

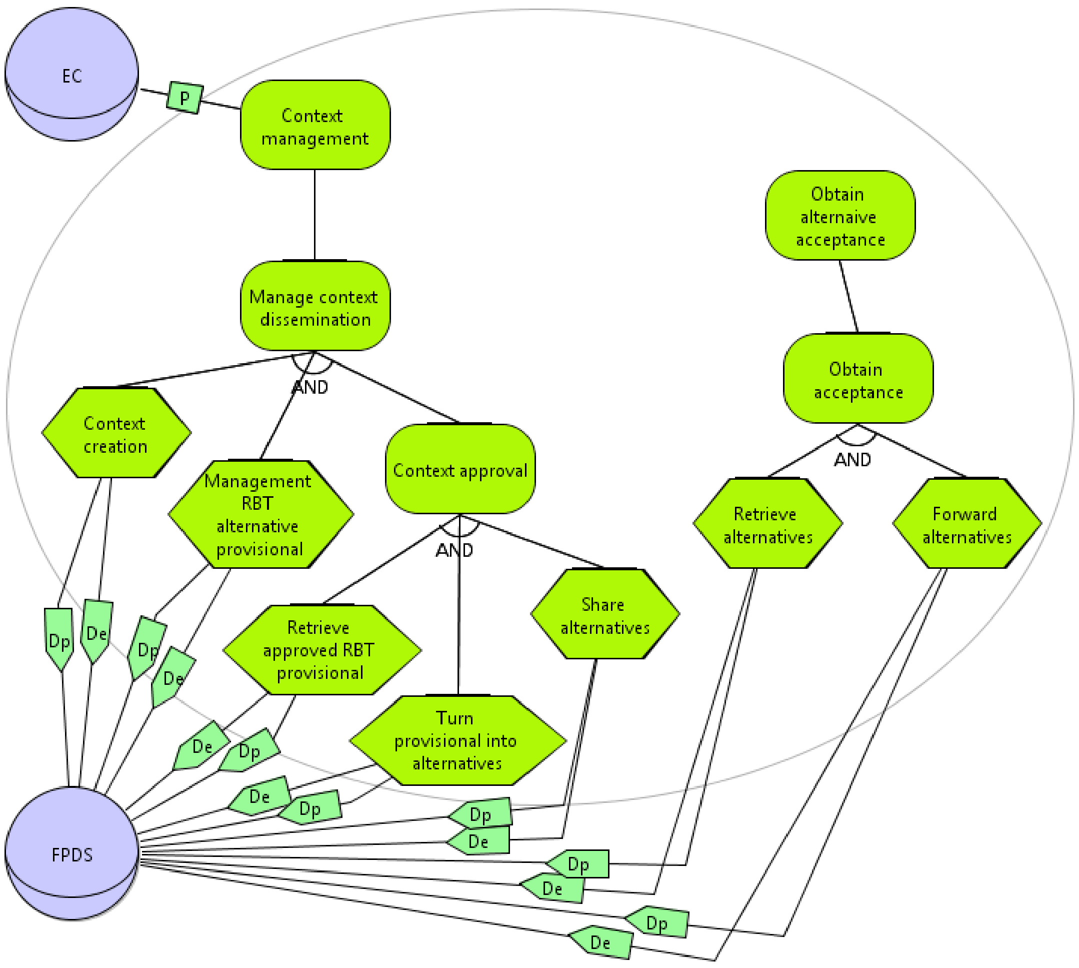

EC
P
Context management
Manage context dissemination
AND
Context creation
Management RBT alternative provisional
Context approval
AND
Retrieve approved RBT provisional
Turn provisional into alternatives
Share alternatives
Obtain alternaive acceptance
Obtain acceptance
AND
Retrieve alternatives
Forward alternatives
Dp
De
FPDS

Figure 7.15: Requirements diagram for Executive Controller (EC) actor.

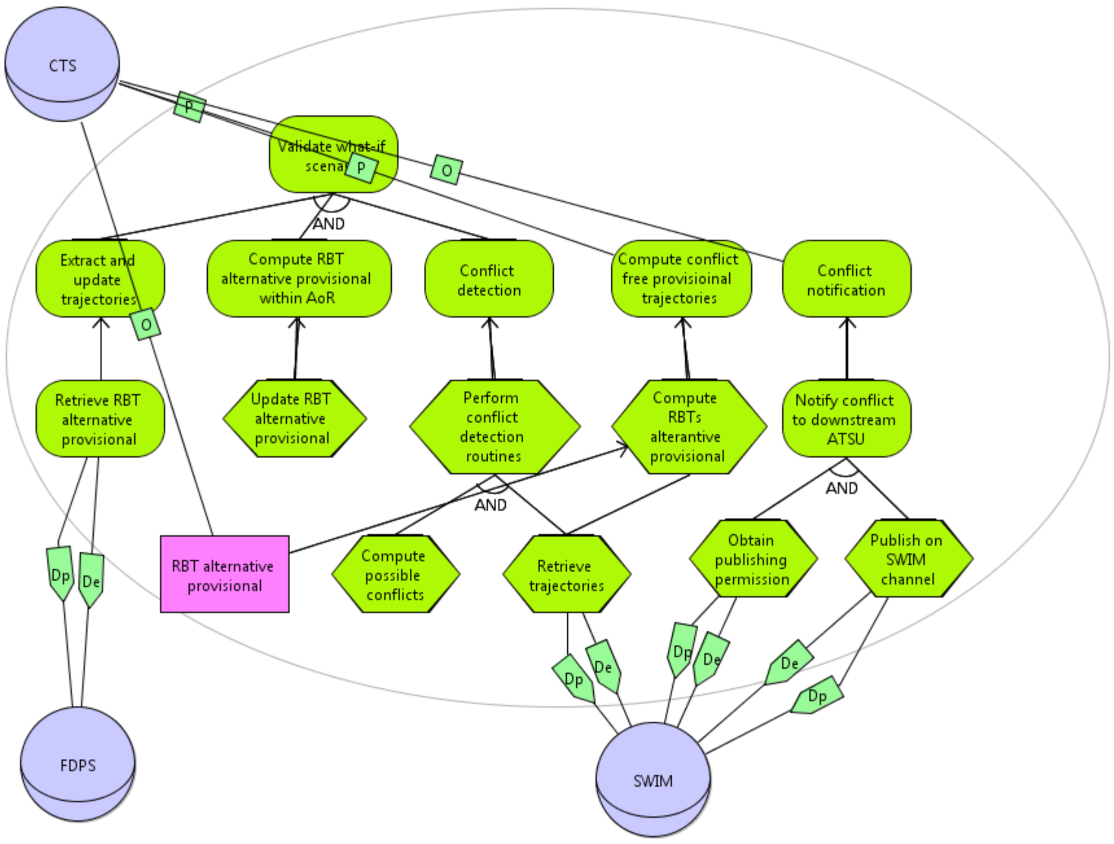

CTS
P
Validate what-if scena
P
O
AND
Extract and update trajectories
O
Compute RBT alternative provisional within AoR
Conflict detection
Compute conflict free provisioinal trajectories
Conflict notification
Retrieve RBT alternative provisional
Update RBT alternative provisional
Perform conflict detection routines
Compute RBTs alterantive provisional
Notify conflict to downstream ATSU
AND
AND
RBT alternative provisional
Compute possible conflicts
Retrieve trajectories
Obtain publishing permission
Publish on SWIM channel
Dp
De
Dp
De
Dp
De
De
Dp
FDPS
SWIM

Figure 7.16: Requirements diagram for Conflict Tools System (CTS) actor.

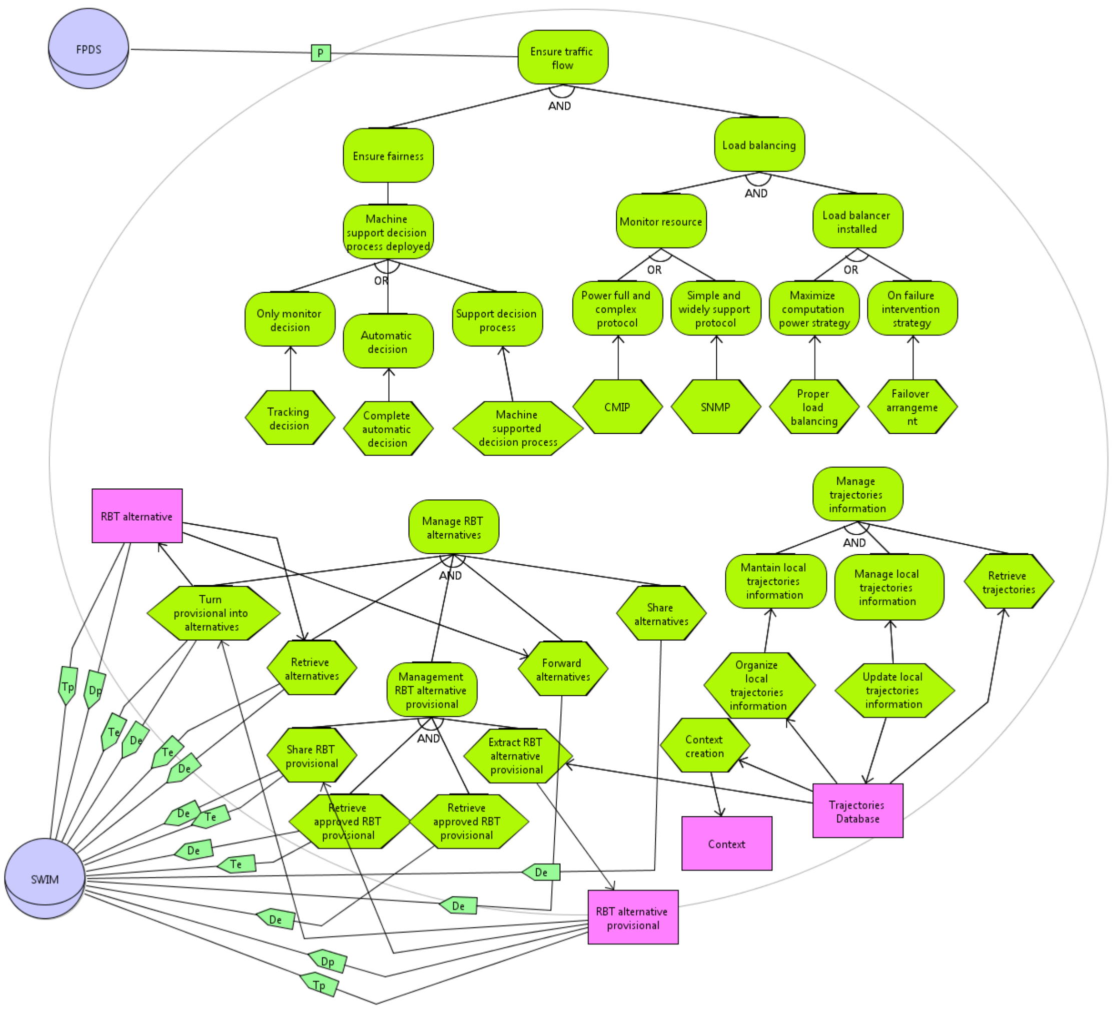

FPDS
P
Ensure traffic flow
AND
Ensure fairness
Load balancing
AND
Machine support decision process deployed
OR
Monitor resource
Load balancer installed
OR
OR
Only monitor decision
Automatic decision
Support decision process
Power full and complex protocol
Simple and widely support protocol
Maximize computation power strategy
On failure intervention strategy
Tracking decision
Complete automatic decision
Machine supported decision process
CMIP
SNMP
Proper load balancing
Failover arrangement
RBT alternative
Manage RBT alternatives
AND
Manage trajectories information
AND
Turn provisional into alternatives
Share alternatives
Mantain local trajectories information
Manage local trajectories information
Retrieve trajectories
Retrieve alternatives
Forward alternatives
Management RBT alternative provisional
AND
Organize local trajectories information
Update local trajectories information
Share RBT provisional
Extract RBT alternative provisional
Context creation
Retrieve approved RBT provisional
Retrieve approved RBT provisional
Context
Trajectories Database
SWIM
RBT alternative provisional
Tp
Dp
Te
De
Te
De
De
Te
De
Te
De
De
De
Dp
Tp

Figure 7.17: Requirements diagram for Flight Data Processing System (FDPS) actor.

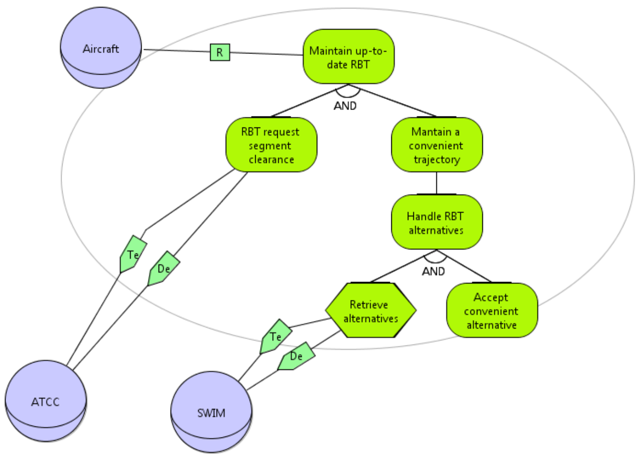


Figure 7.18: Requirements diagram for Aircraft actor.

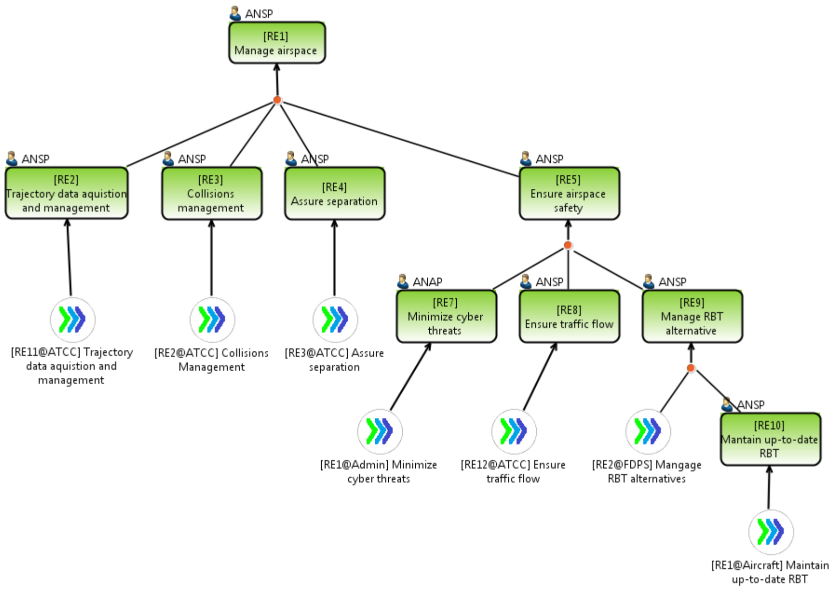

ANSP
[RE1] Manage airspace
ANSP
[RE2] Trajectory data aquistion and management
ANSP
[RE3] Collisions management
ANSP
[RE4] Assure separation
ANSP
[RE5] Ensure airspace safety
[RE11@ATCC] Trajectory data aquistion and management
[RE2@ATCC] Collisions Management
[RE3@ATCC] Assure separation
ANAP
[RE7] Minimize cyber threats
ANSP
[RE8] Ensure traffic flow
ANSP
[RE9] Manage RBT alternative
ANSP
[RE10] Mantain up-to-date RBT
[RE1@Admin] Minimize cyber threats
[RE12@ATCC] Ensure traffic flow
[RE2@FDPS] Mangage RBT alternatives
[RE1@Aircraft] Maintain up-to-date RBT

Figure 7.19: Hypergraph requirements diagram for Airspace Navigation Service Provider (ANSP) actor.

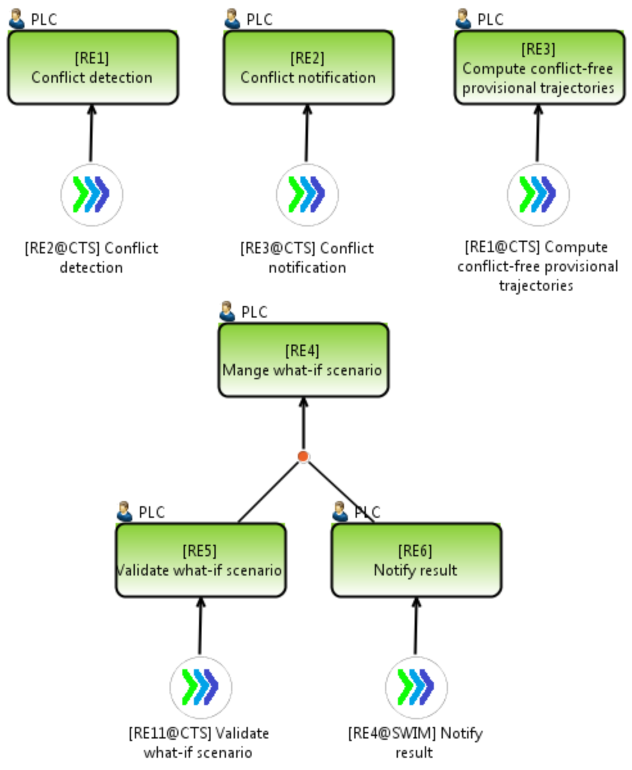

PLC
[RE1] Conflict detection
PLC
[RE2] Conflict notification
PLC
[RE3] Compute conflict-free provisional trajectories
[RE2@CTS] Conflict detection
[RE3@CTS] Conflict notification
[RE1@CTS] Compute conflict-free provisional trajectories
PLC
[RE4] Mange what-if scenario
PLC
[RE5] Validate what-if scenario
PLC
[RE6] Notify result
[RE11@CTS] Validate what-if scenario
[RE4@SWIM] Notify result

Figure 7.20: Hypergraph requirements diagram for Planning Controller (PLC) actor.

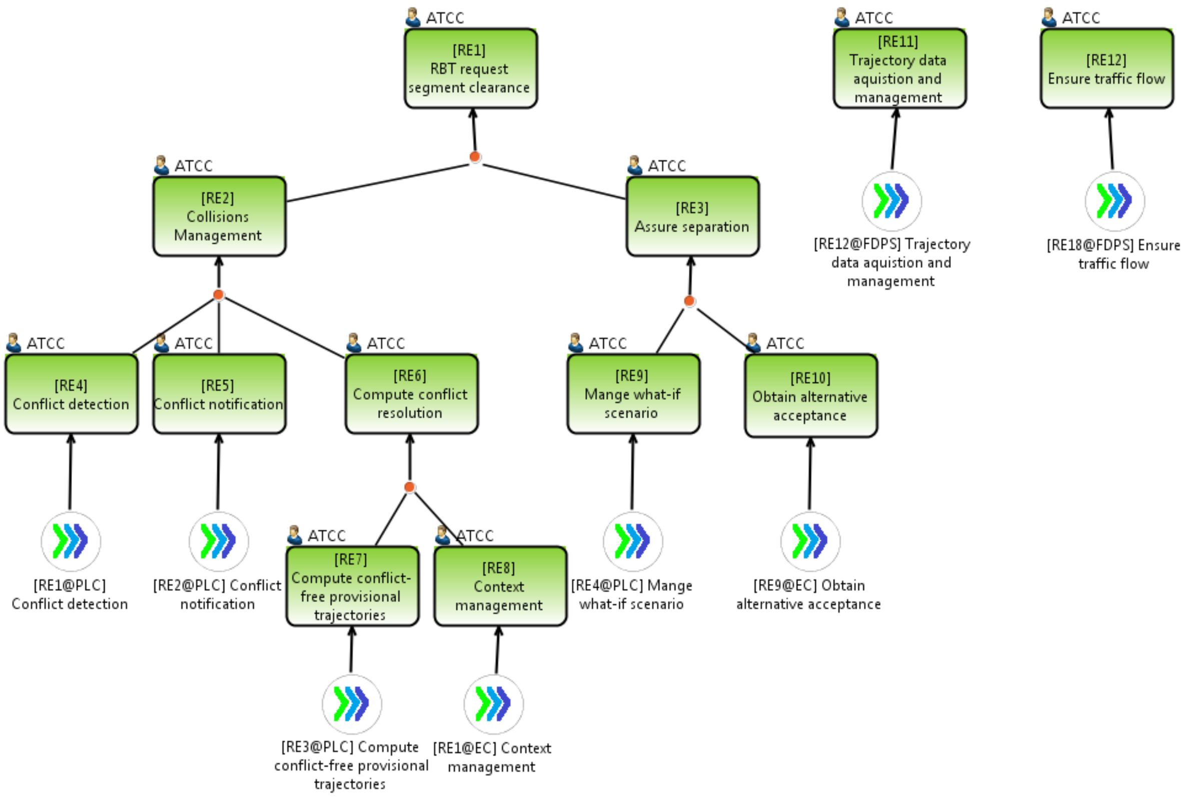

ATCC
[RE1] RBT request segment clearance
ATCC
[RE11] Trajectory data aquistion and management
ATCC
[RE12] Ensure traffic flow
ATCC
[RE2] Collisions Management
ATCC
[RE3] Assure separation
[RE12@FDPS] Trajectory data aquistion and management
[RE18@FDPS] Ensure traffic flow
ATCC
[RE4] Conflict detection
ATCC
[RE5] Conflict notification
ATCC
[RE6] Compute conflict resolution
ATCC
[RE9] Mange what-if scenario
ATCC
[RE10] Obtain alternative acceptance
[RE1@PLC] Conflict detection
[RE2@PLC] Conflict notification
ATCC
[RE7] Compute conflict-free provisional trajectories
ATCC
[RE8] Context management
[RE4@PLC] Mange what-if scenario
[RE9@EC] Obtain alternative acceptance
[RE3@PLC] Compute conflict-free provisional trajectories
[RE1@EC] Context management

Figure 7.21: Hypergraph requirements diagram for Air Traffic Control Center (ATCC) actor.

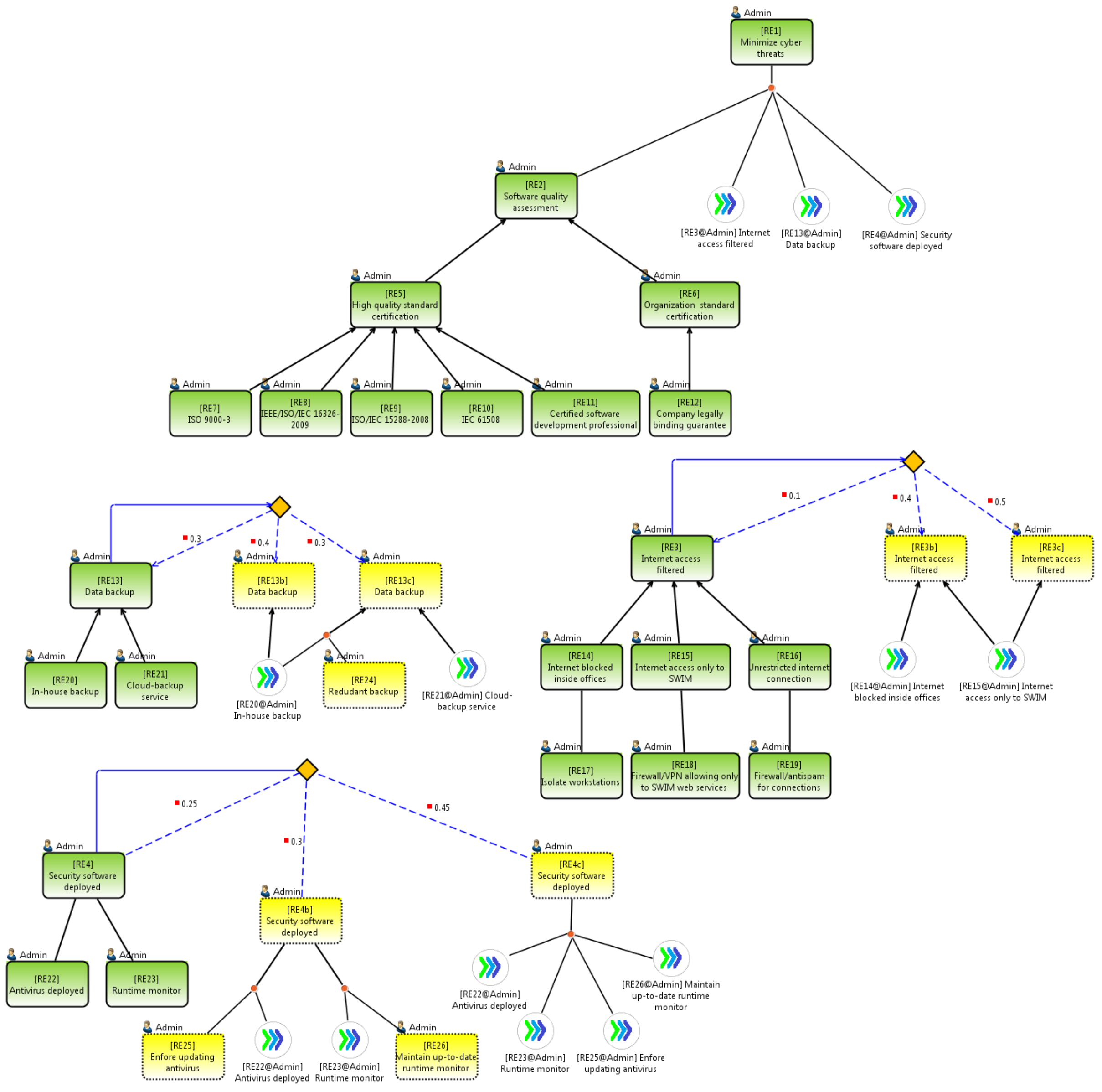

Admin
[RE1] Minimize cyber threats
Admin
[RE2] Software quality assessment
[RE3@Admin] Internet access filtered
[RE13@Admin] Data backup
[RE4@Admin] Security software deployed
Admin
[RE5] High quality standard certification
Admin
[RE6] Organization standard certification
Admin
[RE7] ISO 9000-3
Admin
[RE8] IEEE/ISO/IEC 16326-2009
Admin
[RE9] ISO/IEC 15288-2008
Admin
[RE10] IEC 61508
Admin
[RE11] Certified software development professional
Admin
[RE12] Company legally binding guarantee
0.3
0.4
0.3
Admin
[RE13] Data backup
Admin
[RE13b] Data backup
Admin
[RE13c] Data backup
Admin
[RE20] In-house backup
Admin
[RE21] Cloud-backup service
[RE20@Admin] In-house backup
Admin
[RE24] Redudant backup
[RE21@Admin] Cloud-backup service
0.1
0.4
0.5
Admin
[RE3] Internet access filtered
Admin
[RE3b] Internet access filtered
Admin
[RE3c] Internet access filtered
Admin
[RE14] Internet blocked inside offices
Admin
[RE15] Internet access only to SWIM
Admin
[RE16] Unrestricted internet connection
[RE14@Admin] Internet blocked inside offices
[RE15@Admin] Internet access only to SWIM
Admin
[RE17] Isolate workstations
Admin
[RE18] Firewall/VPN allowing only to SWIM web services
Admin
[RE19] Firewall/antispam for connections
0.25
0.3
0.45
Admin
[RE4] Security software deployed
Admin
[RE4b] Security software deployed
Admin
[RE4c] Security software deployed
Admin
[RE22] Antivirus deployed
Admin
[RE23] Runtime monitor
[RE22@Admin] Antivirus deployed
[RE26@Admin] Maintain up-to-date runtime monitor
Admin
[RE25] Enfore updating antivirus
[RE22@Admin] Antivirus deployed
[RE23@Admin] Runtime monitor
Admin
[RE26] Maintain up-to-date runtime monitor
[RE23@Admin] Runtime monitor
[RE25@Admin] Enforce updating antivirus

Figure 7.22: Hypergraph requirements diagram for Admin actor.

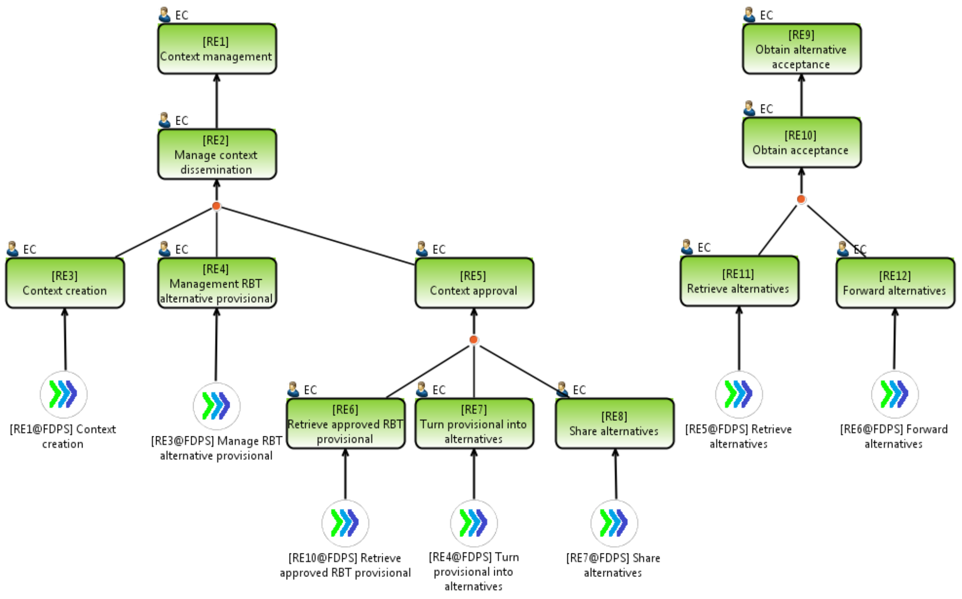

EC
[RE1]
Context management
EC
[RE2]
Manage context dissemination
EC
[RE3]
Context creation
EC
[RE4]
Management RBT alternative provisional
EC
[RE5]
Context approval
EC
[RE6]
Retrieve approved RBT provisional
EC
[RE7]
Turn provisional into alternatives
EC
[RE8]
Share alternatives
EC
[RE9]
Obtain alternative acceptance
EC
[RE10]
Obtain acceptance
EC
[RE11]
Retrieve alternatives
EC
[RE12]
Forward alternatives
[RE1@FDPS] Context creation
[RE3@FDPS] Manage RBT alternative provisional
[RE10@FDPS] Retrieve approved RBT provisional
[RE4@FDPS] Turn provisional into alternatives
[RE7@FDPS] Share alternatives
[RE5@FDPS] Retrieve alternatives
[RE6@FDPS] Forward alternatives

Figure 7.23: Hypergraph requirements diagram for Executive Controller (EC) actor.

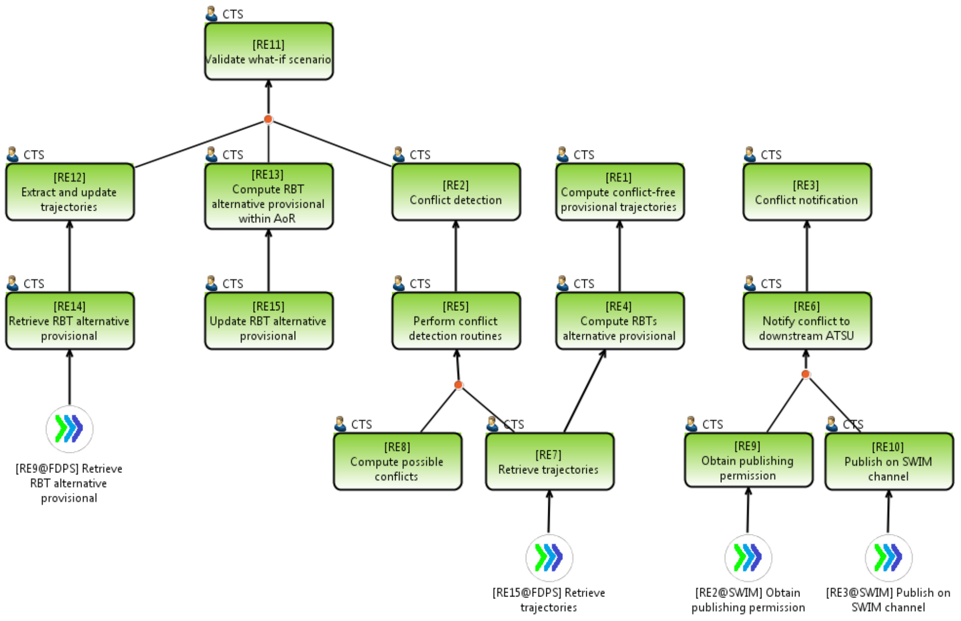

CTS
[RE11]
Validate what-if scenario
CTS
[RE12]
Extract and update trajectories
CTS
[RE13]
Compute RBT alternative provisional within AoR
CTS
[RE2]
Conflict detection
CTS
[RE1]
Compute conflict-free provisional trajectories
CTS
[RE3]
Conflict notification
CTS
[RE14]
Retrieve RBT alternative provisional
CTS
[RE15]
Update RBT alternative provisional
CTS
[RE5]
Perform conflict detection routines
CTS
[RE4]
Compute RBTs alternative provisional
CTS
[RE6]
Notify conflict to downstream ATSU
CTS
[RE8]
Compute possible conflicts
CTS
[RE7]
Retrieve trajectories
CTS
[RE9]
Obtain publishing permission
CTS
[RE10]
Publish on SWIM channel
[RE9@FDPS] Retrieve RBT alternative provisional
[RE15@FDPS] Retrieve trajectories
[RE2@SWIM] Obtain publishing permission
[RE3@SWIM] Publish on SWIM channel

Figure 7.24: Hypergraph requirements diagram for Conflict Tools System (CTS) actor.

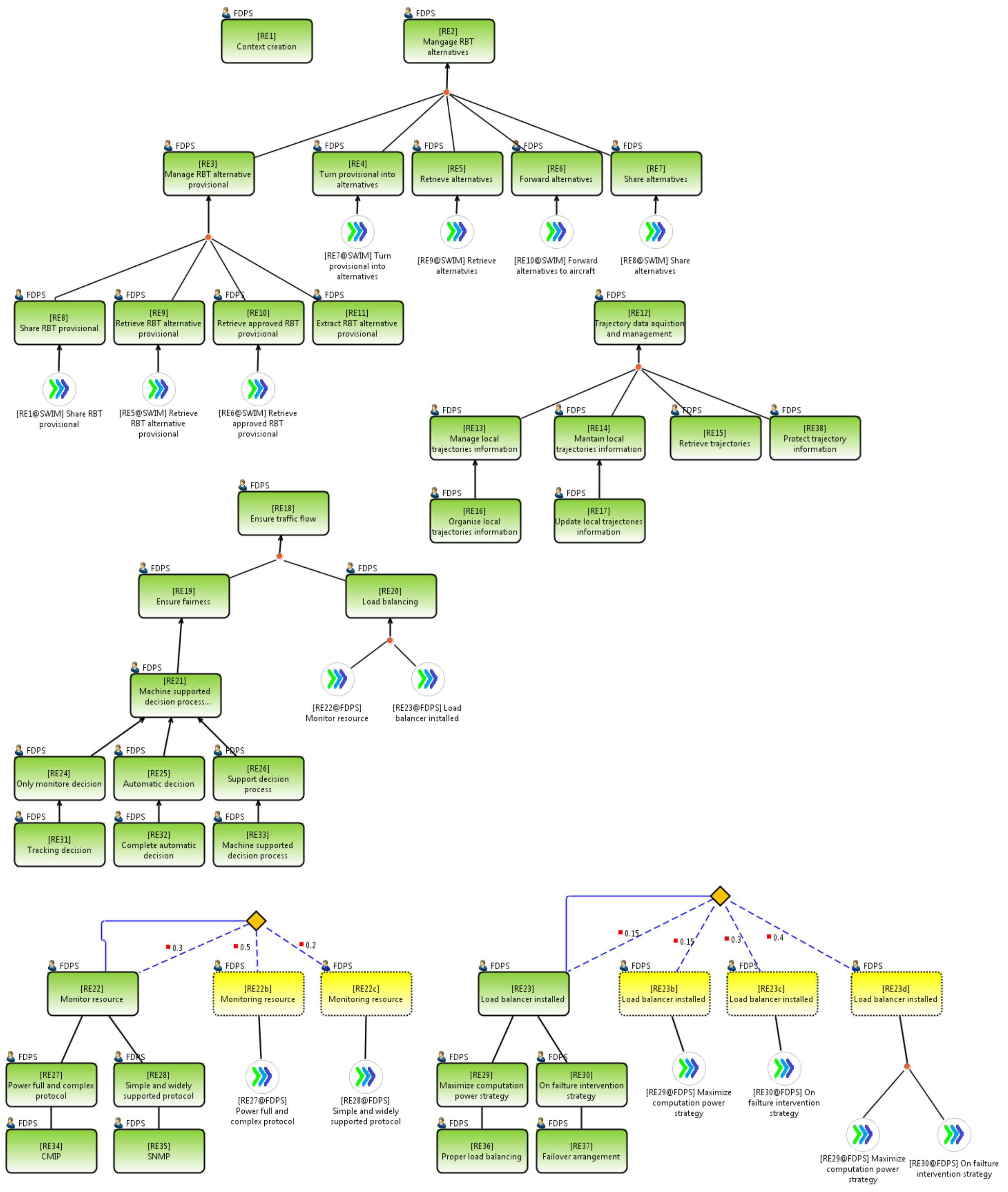

FDPS
[RE1] Context creation
[RE2] Mangage RBT alternatives
[RE3] Manage RBT alternative provisional
[RE4] Turn provisional into alternatives
[RE5] Retrieve alternatives
[RE6] Forward alternatives
[RE7] Share alternatives
[RE7@SWIM] Turn provisional into alternatives
[RE9@SWIM] Retrieve alternatvies
[RE10@SWIM] Forward alternatives to aircraft
[RE8@SWIM] Share alternatives
[RE8] Share RBT provisional
[RE9] Retrieve RBT alternative provisional
[RE10] Retrieve approved RBT provisional
[RE11] Extract RBT alternative provisional
[RE1@SWIM] Share RBT provisional
[RE5@SWIM] Retrieve RBT alternative provisional
[RE6@SWIM] Retrieve approved RBT provisional
[RE12] Trajectory data aquistion and management
[RE13] Manage local trajectories information
[RE14] Mantain local trajectories information
[RE15] Retrieve trajectories
[RE38] Protect trajectory information
[RE16] Organise local trajectories information
[RE17] Update local trajectories information
[RE18] Ensure traffic flow
[RE19] Ensure fairness
[RE20] Load balancing
[RE22@FDPS] Monitor resource
[RE23@FDPS] Load balancer installed
[RE21] Machine supported decision process...
[RE24] Only monitore decision
[RE25] Automatic decision
[RE26] Support decision process
[RE31] Tracking decision
[RE32] Complete automatic decision
[RE33] Machine supported decision process
0.3
0.5
0.2
[RE22] Monitor resource
[RE22b] Monitoring resource
[RE22c] Monitoring resource
[RE27] Power full and complex protocol
[RE28] Simple and widely supported protocol
[RE27@FDPS] Power full and complex protocol
[RE28@FDPS] Simple and widely supported protocol
[RE34] CMIP
[RE35] SNMP
0.15
0.15
0.3
0.4
[RE23] Load balancer installed
[RE23b] Load balancer installed
[RE23c] Load balancer installed
[RE23d] Load balancer installed
[RE29] Maximize computation power strategy
[RE30] On failture intervention strategy
[RE36] Proper load balancing
[RE37] Failover arrangement
[RE29@FDPS] Maximize computation power strategy
[RE30@FDPS] On failture intervention strategy
[RE29@FDPS] Maximize computation power strategy
[RE30@FDPS] On failture intervention strategy

Figure 7.25: Hypergraph requirements diagram for Flight Data Processing System (FDPS) actor.

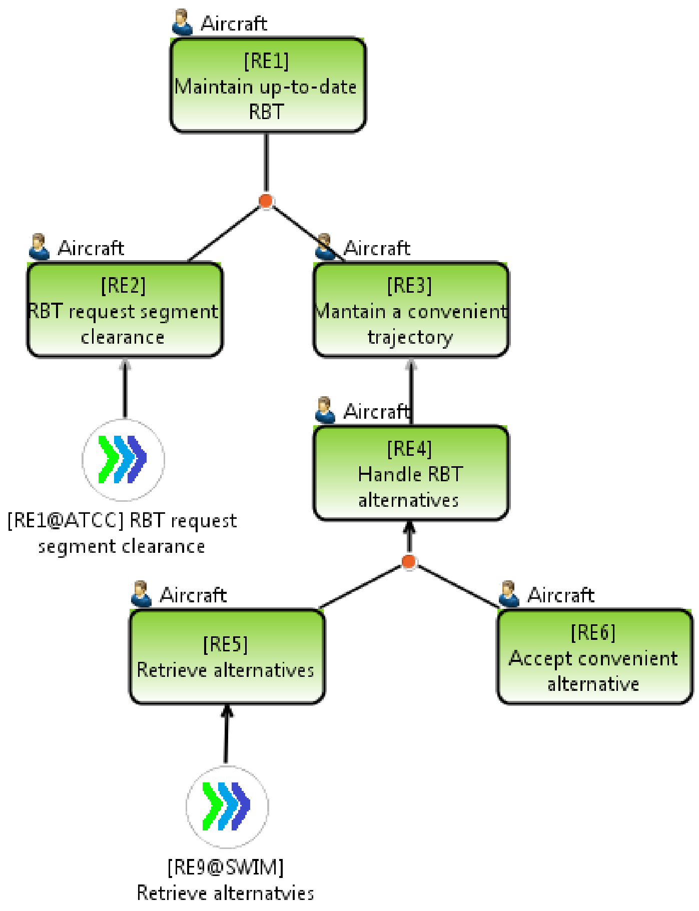


Figure 7.26: Hypergraph requirements diagram for Aircraft actor.

CHAPTER 8

EMPIRICAL EVALUATION OF THE FRAMEWORK WITH THIRD-PARTY

In this chapter, we report the results of the empirical evaluation on the modeling approach of the proposed framework. The studies involve participants who have different level of knowledge of the framework and of the ATM domain. The results from the studies show that the modeling approach is effective in capturing evolution of complex systems. In addition, domain knowledge and method knowledge do not have an observable effect on the effectiveness of the framework.

THIS chapter presents the results of an empirical evaluation conducted on the evolution modeling approach of the proposed framework (previously proposed in chapter 5). The evaluation aimed to assess the *effectiveness* of the approach in modeling requirements evolution and whether the effectiveness depends on the analyst's level of knowledge of the approach and of the application domain. To this end, three empirical studies had been conducted with different types of participants, namely domain experts, researchers, and students, with different level of knowledge of the modeling approach and of the application domain.

As context for the evaluation, the ATM domain was chosen for three main reasons. First, ATM systems are complex and critical systems that are going through significant architectural, organizational, and operational changes as planned by the SESAR [EUR03]. Second, change management is a critical issue in the ATM domain. The need of system engineering techniques to support change management is well recognized [Gra+09]. Last but not least there is a significant body of research about empirical evaluations of requirements engineering approaches in the ATM domain [MR05; Mai+04; Ncu+07]. For example, in [Mai+04], Departure Manager (DMAN), a system for managing departure of aircrafts, is used as context of evaluation. This makes it easier to benchmark the evaluation studies. In the empirical evaluation, we have focused on changes associated with the introduction of a new decision supporting tool, AMAN, and SWIM in the ATM domain.

Figure 8.1 summarizes how the empirical evaluation studies in this chapter developed along a two-year horizon. First, a study had been conducted within the researchers who have proposed the approach to model evolving requirements. Then, the envelope has been pushed further by carrying out a series of workshops with domain experts and industry practitioners as in [Ncu+07]. Last, a study with MSc students was conducted.

The researchers (or also called method experts, interchangeably) have a good knowledge of method (*i.e.,* the modeling approach proposed in chapter 5), but their knowledge of domain is limited. In contrast, domain experts (or also called practitioners, interchangeably) have a good knowledge of the domain, but their knowledge of the method is limited. The students are novices as their knowledge of domain and method are both limited.

The results from the studies show that the modeling approach is effective in capturing evolution of complex systems. In fact, the studies showed that it is reasonably possible for people different than the method's own inventor (such as students or domain experts) to build significantly large models, and identify possible ways for these models to evolve. Moreover, the studies have shown that for domain experts, method experts, and novices, if they are supplied with appropriate knowledge (*i.e.,* knowledge of method for domain experts, knowledge of domain for method experts, and knowledge of both domain and method for novices), they can model the evolution modeling with no significant difference.

This chapter is structured as follows. Section 8.1 briefly discusses how we apply the proposed modeling approach in Si* modeling language. We describe the research methodology in Section 8.2. Section 8.3 presents the analysis of the data collected during the studies. Section 8.4 summarizes the main findings. Section 8.5 discusses the threats to validity. Section 8.6 presents lessons learnt from the studies. Section 8.7 summarizes the chapter.

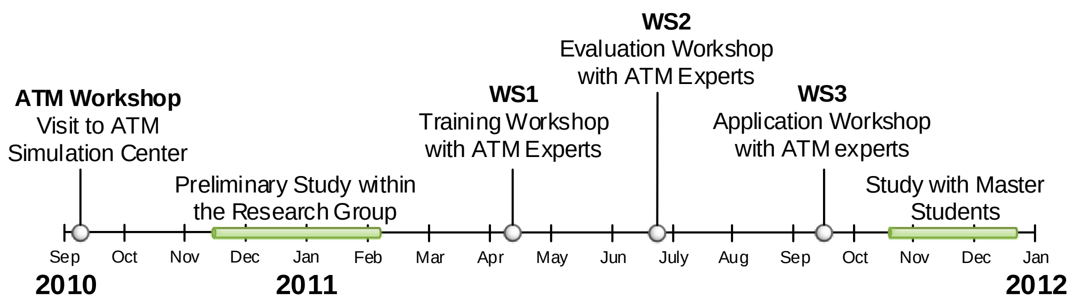


Figure 8.1: Chronology of the family of empirical studies

8.1 Requirements Evolution in Si* Modeling Language

The evolution modeling approach discussed in chapter 5 is independent from any particular RE modeling language. In the evaluation studies, we use the Si* language [Mas+10] to represent requirements models. Si* is founded on the concepts of *actor*, *goal*, *task*, *resource* and the relations *AND/OR decomposition*, *means-end*, and *delegation*. An *actor* is an active entity that models humans as well as software agents and organizations. A *goal* captures a strategic interest that actor wants to be achieved. A *task* represents a particular course of actions that produces a desired effect. It can be executed to satisfy a goal. A *resource* is an artifact produced/consumed by a goal or a task. *AND/OR decomposition* is used to refine a goal into sub-goals. The *AND-decomposition* means that the parent goal will be achieved if all its sub-goals are achieved or satisfied. The *OR-decomposition*, instead, means the parent goal will be achieved if at least one of its subgoals are achieved. The branches of a goal decomposition represent different design alternatives to fulfill the top goal. A *delegation* relation between two actors marks a formal passage of responsibility (*delegation execution*) or authority (*delegation permission*) from an actor (*delegator*) to the actor receiving the responsibility/authority (*delegatee*) to achieve a goal or to provide a resource.

Actors, goals, tasks and resources are graphically represented as circles, ovals, hexagons and rectangles, respectively. Delegation of execution and delegation of permission relations are graphically modeled as edges labeled with **De** and **Dp**, respectively.

We will now illustrate the concept of evolution rules using a simplified version of the application scenarios introduced in section 4.2 of chapter 4.

Example 8.1 (*Before* Model) Recall to the evaluation scenarios described in Section 4.2, Figure 8.2 presents an excerpt of the goal model for the Sector Team before the introduction of AMAN. In the Sector Team's goal model we do not decompose the top goals into

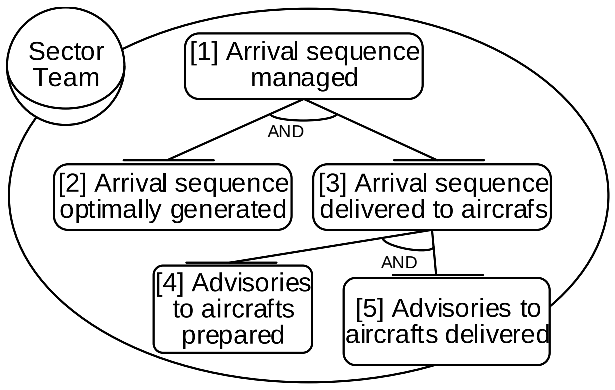

Sector Team
[1] Arrival sequence managed
AND
[2] Arrival sequence optimally generated
[3] Arrival sequence delivered to aircrafs
AND
[4] Advisories to aircrafts prepared
[5] Advisories to aircrafts delivered

Figure 8.2: An excerpt of the goal model for the Sector Team.

operational tasks to keep the model simple and easy to read. The top goal is g_1:*"Arrival sequence managed"*, which is and-decomposed into g_2:*"Arrival sequence optimally generated"* and g_3:*"Arrival sequence delivered to aircrafts"*. The latter goal g_3 is further and-decomposed into g_4:*"Advisories to aircrafts prepared"* and g_5:*"Advisories to aircrafts delivered"*. ■

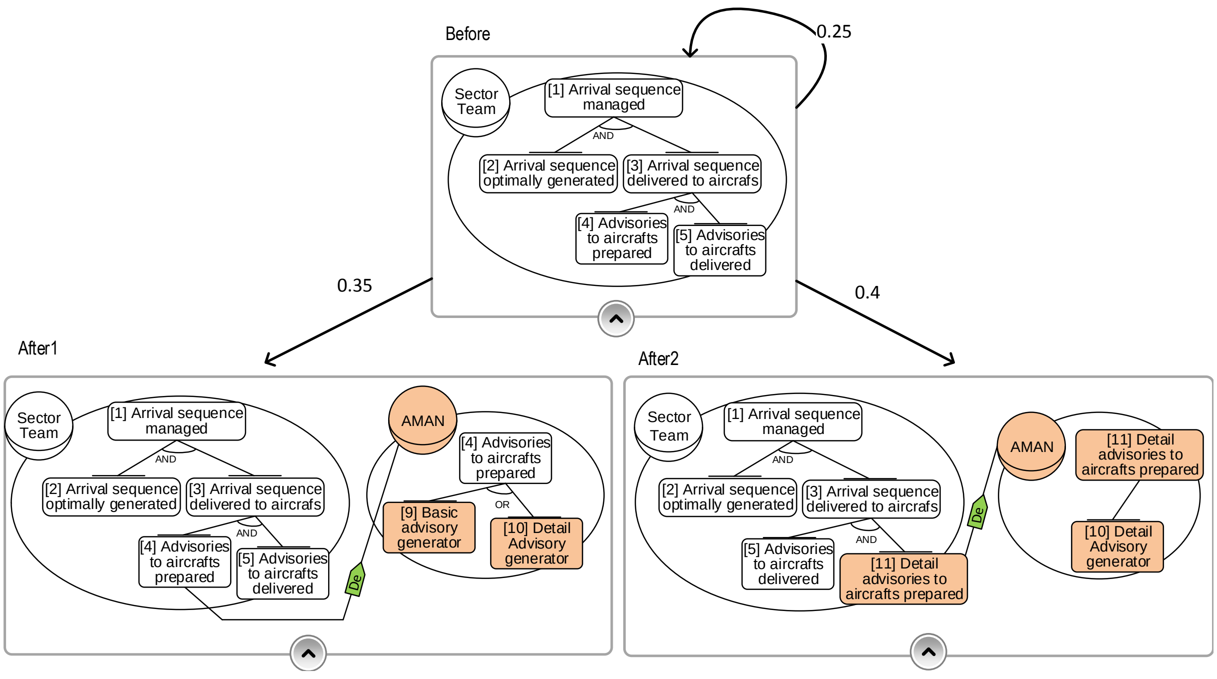
To represent an observable rule in a Si* model, we introduce a new graphical construct, which is the container of the *before* and *after* models. When a model is large and complex, wrapping the entire model inside a container is sometimes hard to follow. Therefore, we recommend to analyze the evolution in sub parts of the models. However, if a big model is unavoidable, it requires the modeling tool to have a large-model support mechanism. The prototype for modeling requirements evolution (see Chapter 7) addresses this problem by dividing a large model into several diagrams. A diagram then can link to others by using a special modeling construct – *Off-diagram Reference*, see Section 7.1.

Controllable rules are implicitly represented by *OR* decomposition, a native Si* graphical construct that allows designers to express alternative sub-goals to implement a parent goal. Therefore, in Si*, we represent controllable rules by means of *OR* decompositions.

Example 8.2 (Evolution Rules) We now illustrate how the Sector Team's goal model in Figure 8.2 can evolve. We focus on goal g_3:*"Arrival sequence delivered to aircrafts"*. We call *Before* the sub-model rooted at g_3. The figure represents one observable rule and one controllable rule. The observable rule

$$r_o(Before) = \left\{ Before \xrightarrow{0.4} After_1, Before \xrightarrow{0.35} After_2, Before \xrightarrow{0.25} Before \right\}$$

consists of three *evolution branches*: each branch corresponds to the arrow that links the before model *Before* to one of the after models $After_1$, $After_2$ and *Before*. In the first evolution


Before
0.25
Sector Team
[1] Arrival sequence managed
AND
[2] Arrival sequence optimally generated
[3] Arrival sequence delivered to aircrafs
AND
[4] Advisories to aircrafts prepared
[5] Advisories to aircrafts delivered
0.35
0.4
After1
Sector Team
[1] Arrival sequence managed
AND
[2] Arrival sequence optimally generated
[3] Arrival sequence delivered to aircrafs
AND
[4] Advisories to aircrafts prepared
[5] Advisories to aircrafts delivered
De
AMAN
[4] Advisories to aircrafts prepared
OR
[9] Basic advisory generator
[10] Detail Advisory generator
After2
Sector Team
[1] Arrival sequence managed
AND
[2] Arrival sequence optimally generated
[3] Arrival sequence delivered to aircrafs
AND
[5] Advisories to aircrafts delivered
[11] Detail advisories to aircrafts prepared
De
AMAN
[11] Detail advisories to aircrafts prepared
[10] Detail Advisory generator

The rectangles with label on top are the containers of *Before* and *After* model. The label is the name of the contained model. Each container has a chevon at the bottom to collapse/expand its content. The arrows labeled with probability connecting two containers determines that the source model evolves to the target model.

Figure 8.3: The graphical representation of the observable for goal g_3.

possibility, g_4 is delegated to AMAN. This dependency is presented by the line labeled with **De** connecting goal g_4 to the actor AMAN. The actor AMAN satisfies g_4 by either g_9:*"Basic advisory generator"*, or by g_{10}:*"Detail advisory generator"*. The probability that this possibility becomes true is 0.4. In the second evolution possibility, *Before* might evolve to *After*$_2$ where a new goal g_{11}:*"Detail advisories to aircrafts prepared"* replaces g_4. The g_{11} is also delegated to AMAN, and it is fulfilled by g_{10}:*"Detail advisory generator"*. The probability that this possibility occurs is 0.35. The third evolution possibility is that the model *Before* does not change with probability 0.25.

The controllable rule is represented by the OR-decomposition of g_4 into goals g_9 and g_{10} in *After*$_1$. This rule has only two branches corresponding to the branches of the OR-decomposition. ■

8.2 Evaluation Method

In this section we present the research questions and hypotheses (Section 8.2.1), and the protocol followed to conduct the evaluation studies (Section 8.2.2).

8.2.1 Research Objectives

Following the Goal-Question-Metric template [BR88], the goal of the studies is to assess if the method proposed in Chapter 5 is *effective* in capturing potential evolution of complex system requirements and whether effectiveness is influenced by knowledge of the domain or the method itself. Given the goal of the evaluation (see also **RQ1** – Section 2.1, and E2 – Section 2.2), the main questions to be answered are:

RQ3 *Is the approach effective in modeling requirements evolution of complex systems?*

RQ4 *How does effectiveness of the approach is impacted by knowledge of domain and knowledge of method?*

The definition of effectiveness is borrowed from the Method Evaluation Model proposed by Moody [Moo03] where the effectiveness of a method is defined as how well it achieves its objectives. Effectiveness can be measured by evaluating the quantity and/or quality of the results (output measures). Thus, to measure the effectiveness of the method to answer **RQ3** we use the following variables that correspond to the main characteristics of evolution rules, the main outcome of the method's application:

- *size of baseline.* It is the number of unique model elements and interconnections in the before model of an observable rule.
- *size of change.* It is number of unique model elements and interconnections across all after models that are not in the before model or disappeared from the before model of an observable rule.
- *number of evolution rules.*
- *number of branches for evolution rules.*

The calculation of these variables is illustrated using a simplified version of the scenarios introduced in section 8.1.

Example 8.3 (Counting Dependent Variables) The dependent variables for the evolution rule described in Example 8.2 can be computed as follows:

- *size of baseline* = *8*, which includes 1 actor (Sector Team), 5 unique goals (g_1–g_5), 2 AND-decompositions (g_1 decomposes to g_2, g_3; and g_3 decomposes to g_4, g_5).
- *size of change* =*11*, which includes 1 new actor (AMAN) + 1 deleted goal (g_4) + 3 new unique goals (g_9, g_{10}, g_{11}) + 2 De-dependency + 1 new OR-decomposition + 2 new AND-decompositions (g_3 decomposes to g_5, g_{11}; g_{11} decompose to g_{10}) + 1 deleted AND-decomposition (g_3 decomposes to g_4, g_5).
- *number of evolution rules* = *2*, which includes 1 observable rule, and 1 controllable rule (g_4 OR-decomposes to g_9, g_{10}.)
- *number of branches for evolution rules*: the observable rule has 3 branches, and the controllable rule has 2 branches.

■

To investigate the second research question **RQ4**, we use as control variables the method knowledge and the domain knowledge of subjects participating in the studies. We also defined the following set of null hypotheses $Hn.m_0$: n denotes the research question to which the hypothesis is related, m denotes the progressive hypothesis number, and 0 denotes that it is a null hypothesis.

H2.1$_0$ *There is no difference in the size of baseline identified by researchers, practitioners and master students.*

H2.2$_0$ *There is no difference in the size of changes identified by researchers, practitioners and master students.*

H2.3$_0$ *There is no difference in the number of evolution rules identified by researchers, practitioners and master students.*

H2.4$_0$ *There is no difference in the number of branches for evolution rules identified by researchers, practitioners and master students.*

8.2.2 Experimental Design

The protocol consists of three main phases:

- *Training.*
  - Participants are administered a questionnaire to collect information about their level of expertise in requirement engineering, security and on other methods they may know.
  - Participants have to attend lectures about the modeling approach and on the ATM evolution scenarios depending on their expertise.
  - Participants are given a training material consisting of the slides used for introducing the modeling approach and documents describing the evolution scenarios and their requirements.
- *Application.*
  - Participants work alone or in groups and apply the modeling approach to the ATM evolution scenarios.
  - At the end of the application phase, participants have to deliver a report documenting the application of the method.
- *Evaluation.*
  - Participants are requested to evaluate the modeling approach through focus group interviews.
  - An ATM domain expert evaluates the report delivered by the participants to assess the quality of the models and the evolution rules drafted by them.

8.2.3 Experimental Procedure

We have conducted three studies with different kinds of participants. Following the terminology in [NB12], first, we run a preliminary study where the participants were the same researchers who have proposed the approach: the researchers have a good knowledge of the approach but are domain limited. Second, we have conducted a study with domain experts (a.k.a practitioners) who are novice to the approach, but have a very good knowledge of the ATM domain. Third, we have conducted a study with master students who are method and domain limited (*i.e.,* they have little prior knowledge of the approach and of the ATM domain).

Table 8.1 summarizes the participants and their knowledge for each study. By 'domain knowledge' (or knowledge of domain), we mean the participants' level of expertise to the

Table 8.1: Participants' knowledge in the empirical studies.

Study	**Participants**	**Method Knowledge**	**Domain Knowledge**
Study 1	Researchers	Good	Limited
Study 2	Domain Experts	Limited	Good
Study 3	Master Students	Limited	Limited

scenarios during the course of the evaluation studies. For the participants who do not have any prior domain knowledge, we provide the scenario documents (see Table 4.1) in advance. Similarly, by 'method knowledge' (or knowledge of method), we also mean the level of expertise of participants to the modeling approach (in Section 5.1). We provide training workshop or lectures to participants in advance. To conduct the studies, we have followed a mixed research approach that combines hypothesis testing with focus group interviews.

In what follows, for each study, we describe the participants, and the setting of the study.

8.2.3.1 Study 1: Preliminary Study within the Research Group

Participants Three researchers have participated in the experiment. All of them had a background in requirement engineering and security, and were involved in the design of the approach to model and reason on requirements evolution.

Setting The researchers have first gained knowledge about the domain by attending half a day workshop about ATM procedures and tools, and safety and security issues in ATM organized by Deep Blue. Deep Blue also provided to the research team documentation about ATM process, AMAN and SWIM architecture and their functional and non functional requirements. After the training on the ATM domain, the researchers were engaged in three modeling sessions that took place at the University of Trento, each of the duration of half a day. During these sessions, the researchers worked independently and modeled several evolutionary scenarios following the approach to model requirements evolution. The scenarios considered include the introduction of the AMAN; the introduction of the ADS-B, a new surveillance tool used to determine aircrafts' positions, the introduction of the SWIM, and the introduction of the AMAN and SWIM to connect AMAN with queue management tools in other airports. For each of the evolutionary scenarios, the researchers have drawn an original model *Before* and identified an evolution possibility $After_i$. The *Before* model and the *After* models have been modeled in the Si* language [Mas+10].

8.2.3.2 Study 2: Workshops with ATM experts

The study was organized into three separated workshops held in April 2011 (WS1), June 2011 (WS2), and September 2011 (WS3). The workshops involved both researchers and ATM experts. The role of researchers was to facilitate the workshop and make observations. The role of ATM experts was to apply the modeling approach and provide feedback about its effectiveness to model requirements changes.

- *Training workshop* (WS1) trained the participants on the modeling approach.
- *Evaluation workshop* (WS2) focused on the evaluation of the quality of the models and the evolution rules drawn by the researchers.
- *Application workshop* (WS3) asked ATM experts to apply the approach.

8.2.3.3 WS1: The Training Workshop

Participants Seven ATM experts have participated in WS1: four of them are Deep Blue consultants with various background (*e.g.*, Computer Science, Human Factors, Safety and Security) who have worked in several projects related to the ATM domain. The other three ATM experts have been working for an European Air Navigation Service Provider with different roles and responsibilities: one is a system administrator, while the other two are air traffic controllers. The ATM experts have also extensive experience with the evaluation of new operational concepts [EUR10] and are currently involved in various SESAR evaluations.

Setting The workshop started with a training session to introduce the experts to the requirements engineering domain and the modeling approach for evolving requirements. Then, ATM experts assessed the representation of changes (in terms of goals), the likelihood of particular change scenarios and the representation of such changes. Then, the research team held a focus group with the participants to identify possible evolution rules.

8.2.3.4 WS2: The Evaluation Workshop

Participants Eight ATM experts participated in the second ATM workshop: seven participants were the same from the first workshop plus one additional participant who works as ATM manager.

Figure 8.4: Third ATM workshop.

Setting During the workshop, the researchers have shown the original model and the possibility of evolution model *After* they have drawn in Study 1. The quality of the requirements models and of the evolution rules has been discussed and the models have been revised with the domain experts. At the end of the workshop, the researchers conducted a semi-structured interview to collect preliminary feedbacks on the approach.

8.2.3.5 WS3: The Application Workshop

Participants The third workshop had eleven participants: a security engineer from industry and ten ATM experts. The ATM experts were the same as the other workshop plus two other Deep Blue consultants who have expertise in Security and Safety for ATM systems.

Setting The workshop started with a brief presentation of the scenario to which the experts had to apply the modeling approach to requirements evolution and a summary of the steps they had to follow. The participants were divided into four heterogeneous groups (in terms of expertise). Each group had to draw an original model and one possibility of evolution model *After* using the Si* tool. At the end of the workshop, the research team engaged the participants into a focus group where the participants have provided additional feedback about the modeling approach. The application phase and the focus groups session have been audio-video recorded (see Figure 8.4). Due to the limited time availability of the participants, the application phase did not terminate with the third workshop but continued remotely over a three months period going from September to November 2011.

8.2.3.6 Study 3: Study with Master Students

Participants Eleven students enrolled in the Master in Computer Science at the University of Trento participated in the study. They had a background in Security Engineering and Information Systems.

Setting Students were trained about the approach for evolving requirements during the Security Engineering course and they were introduced to the ATM domain. As additional materials, they received three documents describing AMAN and SWIM users requirements, SWIM content and information services, and AMAN and SWIM core architecture. Then, the participants were divided in four groups. Each group chose a possible scenario associated with the introduction of AMAN and SWIM network, and had to apply the approach for evolving requirements. After examining the scenarios, they drafted Si* models representing the requirements of the chosen scenario, and identified controllable and observable evolution rules. The participants were not observed during the application phase. Thus, to collect data about the application phase, students were asked to deliver a report describing in details the application of the approach and the generated models.

8.3 Quantitative Data Analysis

We collected the artifacts produced by researchers, domain experts and students as summarized in Table 8.2. The table reports for researchers, domain experts and students the mean and standard deviations of size of baseline, size of changes, and number of branches for controllable and observable rules.

To take into account the quality of the evolution rules and requirements models generated by students and researchers, we asked to a Deep Blue consultant who was expert in the ATM domain to assess the quality of the requirements and evolution rules. The level of quality was evaluated on a four-item scale as specified in Table 8.3.

Based on this scale, the groups who have got an assessment *Valuable* or *Specific* were classified as good groups because they have produced evolution rules and requirements models of good quality. On the contrary, the groups who were assessed *Generic* or *Unclear* were considered as not so good (bad) groups. Figure 8.5 reports the final assessments in the

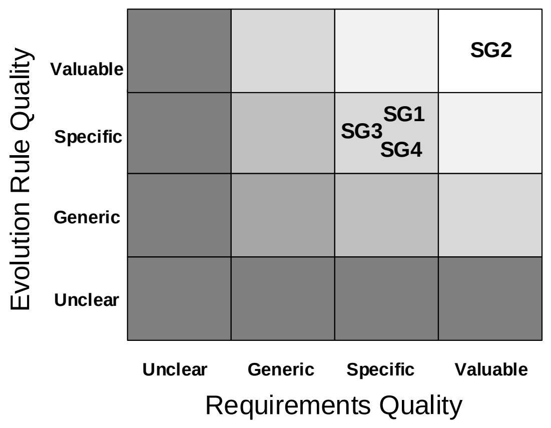
Table 8.2: Data about the Type of Participants and the Artifacts Generated.

Some standard deviation values for practitioners are not available because we have only one group of practitioners in the evaluation studies in this chapter.

	practitioner		**researcher**		**student**	
Effect	mean	std.dev	mean	std.dev	mean	std.dev
Size of Baseline	188.00	0.00	28.67	13.49	156.88	69.62
Size of Change	14.67	4.73	16.67	9.42	9.33	5.47
Number of Observable Rules	3.00	–	2.00	1.00	6.00	2.94
Number of Branches per Observable Rule	2.00	0.00	2.17	0.41	3.42	0.83
Number of Controllable Rules	3.00	–	2.00	1.00	13.00	7.16
Number of Branches per Controllable Rules	2.00	0.00	2.00	0.00	2.27	0.49
Total Number of Rules	6.00	–	4.00	2.00	19.00	10.10

Table 8.3: Scale for expert assessment on the quality of requirements and evolution rules.

Scale	**Requirements Quality**	**Evolution Rules Quality**
Unclear	Not clear which are the requirements for the scenario	Not clear which are the evolution rules for the scenario
Generic	Requirements are present but they are not specific for the scenario	Evolution rules are present but they are not specific for the scenario
Specific	Requirements are present and they are related to the scenario	Evolution rules are present and they are related to the scenario
Valuable	Requirements are present and propose real solutions for the scenario	Evolution rules are present and propose real evolution possibilities for the scenario



SG is the acronym for Student Group.

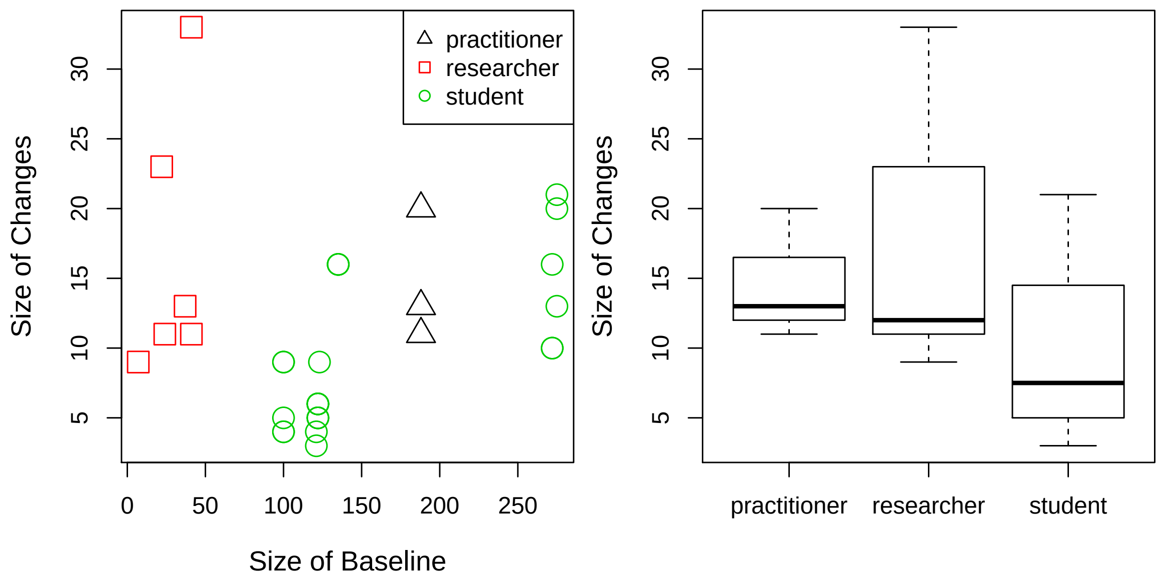
Figure 8.5: The quality of requirements models and evolution rules produced by students.

matrix where the columns are the different quality levels of identified requirements models, and the rows are those of identified evolution rules. Among four groups of students, three have identified most of specific and important requirements and requirements evolution of the ATM scenarios. The last group was even better. They recognized valuable requirements and evolution rules from the scenarios. This implies that the artifacts produced by students are good enough for the purpose of this work. The quality of the models produced by the researchers was assessed that have good quality on requirements and evolution by the ATM experts during the second workshop.

8.3.1 Preparation for an Analysis of Variance

We wanted to determine the differences between the size of baseline, size of the change for evolution rules, the number of branches for evolution rule, and the number of evolution rules produced by researchers, practitioners, and students by means of the analysis of variance (ANOVA). In order to apply ANOVA, we first checked that its assumptions are satisfied. All the p-values in the following results are given under the assumption that the significance level $\alpha = 0.05$.

Dependent Variables are Normally Distributed To check whether the dependent variables are normally distributed we used the Shapiro-Wilk test [RW11] for normality. For all dependent variables the p-value returned by Shapiro-Wilk test is lower than 0.05, and thus the



(a) Size of Change vs. Size of Baseline

(b) Size of Change for Participants Type

Figure 8.6: Size of Baseline and Size of Changes for Type of Participants

variables are not normally distributed.

Homogeneity of Variances We test the homogeneity of the variances with Flinger-Killen test. The test results are not significant with $p \geq 0.05$, except for the total number of branches. The assumption on homogeneity of variances is thus met for all the dependent variables except for the total number of branches.

Observations Independence By design, the observations about the different types of participants are totally independent of each other.

8.3.2 Results

Since the assumptions on normal distribution are not satisfied, we cannot use ANOVA. We need to apply Kruskal-Wallis test, which is the non-parametric alternative to ANOVA when ANOVA assumptions are not met.

Table 8.4: Kruskal Wallis Summary

Effect	**Degree of Freedom**	**Kruskal Wallis χ^2**	**p-value**
Size of Baseline	2	15.842	**0.000**
Size of Change	2	6.342	**0.042**
Number of Observable Rules	2	4.652	0.098
Number of Controllable Rules	2	5.622	0.060
Total Number of Rules	2	5.622	0.060
Number of Branches for Observable Rule	2	12.786	**0.002**
Number of Branches for Controllable Rule	2	2.801	0.246

Size of baseline and Size of change First, we compared the size of baseline and size of changes identified by researchers, practitioners and students. Figure 8.6(a) shows that researchers have sketched requirement models of lower size but have considered changes of small, medium and big size. Similarly, practitioners have produced a single big requirement model and changes of similar complexity of researchers. Students have produced two different kind of artefacts: some group of students produced small models and small changes; other groups identified one big model and changes of increasing complexity like practitioners. Obviously, the figure shows a difference between domain experts, students, and researchers on the baseline of the models. However, looking at Figure 8.6(b), which reports the distribution of size of changes, we could observe that the size of change among participants might not as much different as the size of baseline.

The results of Kruskal-Wallis test reported in Table 8.4 confirmed the observation. The results are statistically significant both for the size of baseline (p-value = 0.000) and the size of changes (p-value =0.042). However, in Table 8.5, a pairwise comparison conducted using Wilcoxon rank-sum test shows that the difference between the size of baseline is statistically significant only for the pair researcher – student (p-value = 0.000). Instead, the difference between the size of change is not statistically significant for any pairs of participant types.

Number of evolution rules We then compared the difference in the number of evolution rules across the different types of participants. Figure 8.7(a) shows the median of the number of evolution rules in total and for type of rules (observable and controllable) produced by the researchers, practitioners and students. Students produced obviously more evolution rules than researchers and practitioners; while researchers and practitioners produced around the same number of rules. The same holds if we consider the number of controllable rules, but

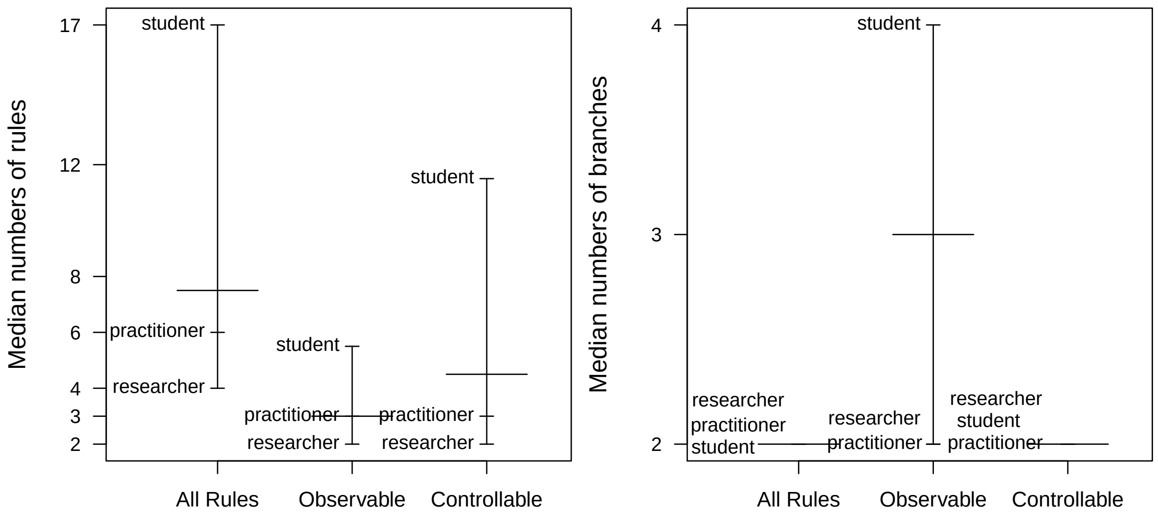
Table 8.5: Wilcoxon Rank-Sum Test Summary -Pairwise Comparison among Type of Participants.

Since we perform three comparisons per each effect, the Bonferroni-corrected significant level α is $^{0.05}/_{3} = 0.017$.

Effect	**Pair of Participant Types**		**p-value**
Size of Baseline	practitioner	researcher	0.025
	practitioner	student	0.171
	researcher	student	**0.000**
Size of Change	practitioner	researcher	1.000
	practitioner	student	0.111
	researcher	student	0.035
Number of Observable Rules	practitioner	researcher	0.637
	practitioner	student	0.468
	researcher	student	0.074
Number of Controllable Rules	practitioner	researcher	0.637
	practitioner	student	0.400
	researcher	student	0.057
Total Number of Rules	practitioner	researcher	0.637
	practitioner	student	0.400
	researcher	student	0.057
Number of Branches for Observable Rule	practitioner	researcher	0.637
	practitioner	student	**0.016**
	researcher	student	**0.003**
Number of Branches for Controllable Rule	practitioner	researcher	**0.000**
	practitioner	student	0.340
	researcher	student	0.175
Total Number of Branches	practitioner	researcher	0.556
	practitioner	student	0.051
	researcher	student	0.023

not the number of observable rules. In fact, researchers, practitioners and students have identified around the same number of observable rules.

We checked with Kruskal-Wallis test (Table 8.4) if these results are statistically significant.



(a) Number of Rules for Participants Type (b) Number of Branches for Participants Type

The short horizonal lines show the median of the total number of evolution rules (a), and median of the total number of branches per rules (b) for type of participants and type of evolution rule.

Figure 8.7: Number of Rules and Branches for Participants Type.

The differences in the number of observable rules (p-value = 0.098), number of controllable rules (p-value = 0.060), and total number of rules (p-value = 0.060) identified by researchers, practitioners and students are not statistically significant. These results are confirmed by the pairwise comparison that we have run with Wilcoxon rank-sum test as shown in Table 8.5.

Number of branches for evolution rules Last, we compared the difference in the number of branches for evolution rules across the different type of participants. Figure 8.7(b) reports the median of the number of branches. Obviously, students perform better than researchers and practitioners with respect to the number of branches for observable rules. They produced more branches of observable rules than those of controllable rules (although in Figure 8.7(a) we see that they produced less observable rules than controllable ones). With respect to the total number of branches for all rules and the total number of branches for controllable rules, there is no observable difference between students, practitioners and researchers.

In Table 8.4, the results of the Kruskal-Wallis test show that the difference in the number of branches per observable rules (p-value = 0.002) is statistically significant. This does not hold for the number of branches for controllable rules (p-value = 0.246). In Table 8.5, the

Table 8.6: Hypothesis testing results

No	Hypothesis	Result
H2.1$_0$	No difference in the size of baseline sketched by researchers, practitioners and master students.	Rejected
H2.2$_0$	No difference in the size of changes identified by researchers, practitioners and master students	Non conclusive
H2.3$_0$	No difference in the number of evolution rules identified by researchers, practitioners and master students.	Accepted
H2.4$_0$	No difference in the number of branches for evolution rules identified by researchers, practitioners and master students	Rejected (for the number of branches for observable rules)

pairwise comparison with Wilcoxon rank-sum test shows that the total number of branches for observable rules is significant difference for the pair practitioner – student (p-value = 0.016) and researcher – student (p-value = 0.003). The difference in the number of branches for controllable rules is statistically significant only for the pair practitioner – researcher (p-value = 0.000). For the total number of branches, there is no statistically significant difference in any pairs of participant types.

8.4 Discussion

This section summarizes the main findings from the studies we conducted (see Table 8.6).

8.4.1 Method's Effectiveness

As shown in Table 8.2, researchers, practitioners and students were able to produce requirements models of medium size and identify new requirements associated with the introduction of the SWIM and the AMAN. In addition, the evaluation of the quality of the models carried by the ATM expert shows that participants were able to recognize and identify evolution rules specific to the introduction of the SWIM and the AMAN. Since the number and the quality of the requirements model and evolution rules identified by the participants was reasonably good, we can conclude that the proposed framework is effective in modeling requirements evolution.

8.4.2 Impact of Knowledge of Domain and Knowledge of Method

Impact on Size of Baseline The results of Kruskal-Wallis test (see Table 8.4) and of the Wilcoxon rank-sum test (see Table 8.5) show that the size of initial requirement models produced by students is higher than the one of researchers. Thus, null hypothesis $H_{2.1.0}$ can be rejected. We could also conclude that domain knowledge and method knowledge do not have an observable effect on the size of baseline variable for two main reasons: a) students who have limited knowledge of the method have produced bigger models than researchers who are method aware; b) students have produced initial requirements models with similar size to the one of the models produced by practitioners who are domain aware.

Impact on Size of Change The Kruskal-Wallis test (see Table 8.4) shows that there is a statistically significant difference in the size of changes identified by researchers, practitioners and students. However, this result is not supported by the Wilcoxon rank-sum test (see Table 8.5), which shows there is no statistically significant difference between any of the pairs of type of participants. Thus, we have not enough evidence to accept or reject hypothesis $H_{2.2.0}$. Based on the results of the Wilcoxon rank-sum test we may conclude that domain knowledge and method knowledge do not determine the size of change.

Impact on Number of Evolution Rules With respect to the number of evolution rules, both the Kruskal-Wallis test (see Table 8.4) and the Wilcoxon rank-sum test (see Table 8.5) show that there is no statistically significant difference among researchers, practitioners and students. Thus, $H_{2.3.0}$ can be accepted and we can conclude that domain knowledge and method knowledge do not have an effect on the number of evolution rules.

Impact on Number of Branches for Evolution Rules Both the Kruskal-Wallis test (see Table 8.4) and the Wilcoxon rank-sum test (see Table 8.5) show that there is a statistically significant difference in the number of branches for observable rules. Students produced observable rules with significantly more branches than the one of researchers and practitioners. As a result, $H_{2.4.0}$ is rejected for the number of branches of observable rule. The students' domain knowledge and method knowledge are limited, it is expected that they would identify less branches than other participant types. However, as mentioned, we have here an evidence that the students performed better than researchers and practitioners with respect to the total number of branches for rule. Thus, we could conclude that domain knowledge and method knowledge do not have an observable effect on the identification of a higher number of evolution scenarios.

8.4.3 Implications for the Method

During the focus group interviews, the ATM experts reported important aspects of the approach that require further investigation. They all pointed out that it is not possible to predict all the possible changes in advance especially for complex systems such as ATM systems:

"*Sometimes, when you apply you discover a third change that is better than the one you have predicted*", ATM Manager
"*The model may be good but when you switch from theory to practice you realize that there are many situations that you did not consider*", ATM Manager.
"*We are talking about very complex systems. You don't know from the beginning all the actors involved in the process. There are always certain changes that you cannot predict due to the complexity of the system*", Senior Deep Blue consultant.

The ATM experts went further and suggested that an iterative approach should be applied to identify all the possible evolution alternatives:

"*It should be an iterative process* ", and "*you need to have more iterations if you want to reach 100%. You cannot foreseen everything at the beginning*", ATM expert.

We have addressed this suggestion in the proposed framework by proposing an incremental reasoning support (described in Chapter 6). This will efficiently enables a multi-round modeling and reasoning of evolution where potential changes are iteratively identified.

The ATM experts also suggested that the graphical representation should be simplified because it does not scale very well for complex systems such as ATM systems. They remarked that an incremental approach should be used to draw the rules. We noted this issue concerning the graphical representation of observable rules (see Figure 8.3). It is difficult to graphically represent observable rules if we treat requirements at a high level of abstraction. Instead, we have proposed alternative representations depending on RE languages. For example, in goal based languages like i*, KAOS, we suggested the use of additional construct such as *observable node* to determine the evolution of a particular goal in a goal model (see Figure 7.1); in risk graph, we suggested the use of tags associated to risk graph elements (see Figure 10.13(b)).

8.5 Threats to Validity

We discuss the four main types of threats to validity [Woh+12] in what follows.

Conclusion validity Conclusion validity is concerned with issues that affect the ability to draw the correct conclusion about the relations between the treatment and the outcome of the experiment. The main threat to conclusion validity relevant for the studies is *low statistical power*. The sample size must be big enough to come to correct conclusions. We performed a post-hoc power analysis for the Kruskal-Wallis test and Wilcoxon rank-sum test for the three users' cohorts. The size of the sample is too small to have a power of 0.80. Therefore, it will be necessary to run the experiment again with more subjects for each user's cohorts - researchers, practitioners and students.

Internal validity Internal validity is concerned with issues that may indicate a causal relationship between the treatment and the outcome, although there is none. A threat to internal validity can be the use of different application scenarios across the three study groups. Different scenarios may generate a bias in this experiment as effects might be due to (or canceled by) the varying difficulties of the scenarios. To mitigate this risk we have asked practitioners to identify scenarios that were close in complexity according to their opinion. An important reason behind the use of different scenarios was to avoid a problem that emerged in previous pilot studies: when a scenario is repeatedly used (and therefore progressively refined) domain experts tend to focus on the adequacy of the requirements models at object level rather than at meta-level. For example, by reconsidering the same scenario twice or assessing the work of the students in comparison with their own, practitioners might have spent several minutes discussing whether the right sentence for the goal in Figure 8.2 was "detailed advisory", and evaluate a model as inadequate because "specific advisory" should have been used in its place. By using different scenarios the evaluation focused on the forest rather than the trees. We believe that the advantages far outweigh the risks.

Another threat to internal validity is related to the use of Si* as requirement modeling language. The feedback provided by the ATM experts on the possible adoption of the graphical representation to model requirements evolution in the ATM domain can be biased by the fact that the requirements model were drawn in the Si* requirements language. Si* graphical notation tends to get very complex even for simple models and this aspect may have influenced the feedback of the ATM experts. We should organize another study using a different requirements language to evaluate whether the feedbacks depend on the use of Si*.

Construct validity Construct validity concerns generalizing the result of the experiment to the concept and theory behind the experiment. The main threat to construct validity in the experiment of this chapter was represented by a communication gap between the research team and the domain experts. Research team and domain experts might use same terms with different meanings and this can lead to misunderstandings; therefore, wrong or unrelated feedback might be provided. For example, the distinction between *goal* and *resource* was difficult to understand for the experts. A resource in the requirements engineering domain is an artifact produced/consumed by a goal, which captures a strategic interest of a stakeholder that is intended to be fulfilled. In the ATM domain, a goal has the same meaning that resource has in the requirements engineering domain and this lead to confusion. To mitigate this threat we have included a "*mediator*" who occasionally reformulated questions of the research team for the domain experts and reformulated domain experts' feedback for the researchers. The mediator role in this experiment was played by a member of Deep Blue who has a solid Information and Communication Technology (ICT) background (PhD in Compute Science), and is very strong experience in the ATM domain (20+ year experience as senior consultant). Before running the studies, we have several discussion with this member to ensure that he can fully understand the proposed framework.

External validity External validity concerns the ability to generalize experiment results outside the experiment settings. External validity is thus affected by the objects and the subjects chosen to conduct the experiment. We reduced the threats to external validity by making the experimental environment as realistic as possible. In fact, as object of the experiment we have chosen a real evolutionary application scenario proposed by Deep Blue, a consulting company, which is actively involved in SESAR Initiative.

However, a threat to external validity of the experiment results is represented by the use of Si* as requirements modeling language. To generalize these results beyond this empirical evaluation, we should run other controlled experiments where different requirements modeling languages – problem frames, natural language, tables – are used by subjects to draw the evolution rules.

8.6 Lessons Learnt

In this section we highlight a number of aspects that we should take into account to continue our research.

- *Subjects' Selection.* The selection of domain experts strongly influences the relevance of feedback collected and the satisfaction of the success criteria chosen for the case studies. In the case studies, the selected domain experts had a different background and so we were able to collect feedback about the approach to requirements evolution from different perspectives. However, an issue of the domain is the separation between ATM organizations and IT suppliers. They have different and often competing stakes. In future studies, we think that one should evaluate the approach separately with two groups of ATM organizations and IT supplier, and identify methods to *firewall feedback* by different groups. This might highlight competitive advantages that one group might gain over the other by adopting the method.

- *Language Gaps.* Another interesting lesson concerns the foreign language gap. The level of engagement of the domain experts depends on two main factors: the means to provide feedback, and the language in which such feedback needs to be provided. Our workshop sessions included Hungarians, Indians, Italians, Norwegians, and Vietnameses; juggling between languages made our meetings lively. Albeit obvious in hindsight, this was not mentioned in the previous work by N. Maiden and others [MR05; Mai+04; Ncu+07] because their studies were clearly English-to-English. A possible solution is that the domain experts can discuss in their mother tongue and then provide summary feedback in English, but this hampers the immediacy of the feedback, and "minority opinions" might not be reported (we noticed this phenomenon during the workshops). The mediator was a useful tool to mitigate the internal validity threats also in this setting.

- *Determinants of Users' Technology Acceptance.* A major factor in the level of engagement of domain experts is the *perceived compliance* with the practice in industry. In the ATM domain, the discussion was facilitated by the existence of a model representation (influence diagrams), which was very close to goal models. When we proposed the same approach to another company, a show-stopper in the discussion with a practitioner was simply "We use DOORS" (and therefore cannot use and should not waste time evaluating requirements models in format different than DOORS). This was purely a syntactical limitation, not a semantical or methodological one: we could have perfectly used DOORS to link requirements expressed by goal models, but our tool simply did not do it, as we thought this was just "Engineering". This is indeed true if we considered limiting our evaluation to an experiment (as noted in [CF+09] this is what the vast majority of RE papers report). Being able to syntactically interface with these tools (even for just gathering requirements IDs to label goals), is essential to obtain better perceived compliance and thus a better engagement and case-study based evaluation. As future work, we would like to organize a controlled experiment to evaluate if perceived compliance is a motivating factor that leads to an individual's intention to use a methodology.

- *Evolution Probability Setting.* An aspect of the approach that deserves further investigation concerns the definition of a systematic process to obtain evolution probabilities. Typically, decision-makers are not able to provide probability that an event occurs but just the frequency with which the event happens [Lun+11a, Chap. 10.2.1]. Even when they are able to provide probabilities, they are subjective and contain a high degree of personal bias. In fact, the probability that an event occurs differs from person to person. We could explore the use of game theory *e.g.,* Clarke-Tax mechanism to assess evolution probabilities based on the probabilities of an event specified by different decision makers.

8.7 Chapter Summary

In this chapter we reported the results of three studies that we have conducted in the ATM domain to evaluate the effectiveness and the impact that domain knowledge and method knowledge on effectiveness of the proposed framework.

The main findings from the studies were that the approach is effective in modeling requirements evolution. In fact, the studies showed that since researchers, practitioners and students were able to produce significantly big requirements models, and identify possible ways for these models to evolve. In addition, domain knowledge and method knowledge did not have an observable effect on the effectiveness of the approach, since there was no statistically significant difference between the artifacts produced by students who have limited knowledge about both domain and method, and researchers who were method aware and practitioners who were domain aware. Determining which are aspects that can have an effect on the effectiveness the proposed framework will be the subject of a separate case study.

In the subsequent chapters we are going to present how to apply the proposed framework in an other field – risk assessment. This will be an evidence for the applicability of the proposed framework.

Part III

Applying the Proposed Framework to Evolving Risks

CHAPTER 9

EARLY DEALING WITH EVOLVING RISKS IN SOFTWARE SYSTEMS

Existing risk assessment methods often rely on a context of a target software system at a particular point of time. Such contexts of long-lived software systems tend to evolve over time. Consequently, risks might also evolve. Therefore, in order to deal with evolving risks, decision makers need to select an appropriate risk countermeasure alternative that is more resilient to evolution than others. To facilitate such decision, we propose a pioneer method taking the uncertainty of evolutions and outputs of a risk assessment to produce additional information about the evolution resilience of countermeasure alternatives.

SECURITY risk analysis concludes with a set of recommended options for mitigating unacceptable risks [ISO09]. In order to treat risks, decision makers (or managers) have to make decisions on proper countermeasures to implement. However, such investment decisions may be complicated. An organization needs the best possible information on risks and countermeasures to decide what is the best investment. This involves deciding which countermeasures offer a good trade-off between benefit and spending. The expenditure required to implement the countermeasures, together with their ability to mitigate risks, are factors that affect the selection. Inappropriate and over-expensive countermeasures are money lost. Therefore, a systematic method that helps to reduce business exposure while balancing countermeasure investment against risks is needed. Such a method should help answering questions like *"(1): How much is it appropriate to spend on countermeasures?"* and *"(2): Where should spending be directed?"* as highlighted by Birch and McEvoy [BM92].

Unfortunately, there exists little support for the prescriptive and specific information that managers require to select cost-effective risk countermeasures. Several cost estimation models have been proposed, but most are only loosely coupled to risk analysis. For example, the Security Attribute Evaluation Method (SAEM) [But02] is well-suited to evaluate risk reduction, but is very vague on the issue of cost effectiveness. Likewise, [WHO09] suggests several methods to assess cost of risks (*e.g.,* Cost-Of-Illness, Willingness-To-Pay), but none of these methods provide specific support to evaluate countermeasure expenditure. Chapman and Leng [CL04] propose a framework that justifies mitigation strategies based on cost-difference, but does not take the benefit-difference (*i.e.,* level of risk reduction) between strategies into consideration.

Effective decision-making requires a correct risk model incorporating multi-aspect information on countermeasures and a method to select between cost-effective countermeasure alternatives. The multi-aspect information should contain the knowledge about the countermeasures themselves, their associated expenditures and suitability to mitigate risks, as well as the impacts they may have on each other.

Such effective decisions are much more important in the context of long-lived software systems, which keep evolving to continuously satisfy changing business needs, new regulations, or the introduction of new technologies. Such evolutions might expose the software systems to new risks, and might make the output of the current risk analysis on the software systems become partially obsoleted. Consequently, the software systems might be no longer secure.

The results of a software system risk assessment are typically valid under a given context, which is a particular system configuration, and under certain requirements and assumptions about the target system at a particular point of time. Once a particular risk assessment is completed, countermeasures are proposed and decision makers (or managers) face the question of selecting an appropriate countermeasure alternative (*i.e.,* a set of countermeasures) to be implemented in order to mitigate unacceptable risks. However, when the context evolves, risks might also evolve. Previously acceptable risks might become unacceptable or vice versa, or new risks emerge [Lun+11a, Chap. 15]. For example, any risk mitigated by SHA-0 based countermeasure was acceptable before 2004, but might be unacceptable later since

SHA-0 was efficiently attacked[1]. Thus, a current countermeasure alternative may no longer be appropriate and it is necessary to develop new ones to address the evolving or newly emerging risks. This might include adding additional security requirements as a protection to ensure the system security, or the creation of completely new security controls. Obviously, implementing new ones to replace for obsoleted ones may be more expensive than having one that still may be appropriate for evolving risks. The decision makers then face an alike question of selecting appropriate countermeasure alternative, yet in the extent of evolution of the context and evolving risks.

While there exist several established risk assessment methods *e.g.*, [Lun+11a; ML05; Nor10; Sch99; Sto+02; Tra+13b], few provide support for a systematic selection on risk countermeasure alternatives [Tra+13b; Sto+02; Nor10], and even less for dealing with evolving risks [Lun+11b]. Traditional risk assessment methods typically perform on a context at a particular point of time and hence cannot guarantee the continuous validity of the risk assessment results in an evolving context. Concerning evolutions, in [Lun+11b; SS13], the authors proposed a general technique and guideline for managing risk in changing systems. However, they did not mention the uncertainty of evolution (*i.e.*, likelihood of occurrence), and how to use this information to support the decision making process.

As an effort to fill some of that void, this chapter adapts the framework described in Chapter 5 to introduce a risk-evolution approach. The focus of this chapter is not on how to obtain the uncertainty information, but rather on how to make use of them to produce additional factors to support the decision making process. Instead, the risk-evolution approach adapts the evolution rules (modeling) and evolution metrics (reasoning) to the evolving risks. The purpose of this risk-evolution approach is to quantify countermeasure alternatives in terms of evolution metrics, and hence assist the selection of an evolution-resilient countermeasure alternative.

This chapter is organized as follows. Section 9.1 presents common terms in this chapter. Section 9.2 describes the risk-evolution approach. Section 9.3 exemplifies the approach in a case study. Section 9.4 summarizes the chapter.

9.1 Terminology

- *Context*: includes all required information to do risk assessment such as assumptions about the working environment, requirements model, targets to be protected and so

[1]http://en.wikipedia.org/wiki/SHA-0#SHA-0, site visited on March, 2013

on. It is the premises for and the background of the risk analysis, as well as the purposes of the analysis and to whom the risk analysis is addressed [Lun+11a, Chap. 5]. According to ISO 31000:2009, a context includes all external factors (*e.g.,* regulatory, environment) and internal factors (*e.g.,* business process, policies, standards, system functions, reference models).

- *Before context*: is the current context at the current time.
- *After context*: is the future context with potential changes.

9.2 The Risk-Evolution Approach

The proposed risk-evolution approach relies on a key concept: *context*. The context here is for risk assessment. We employ the definition of *context* from [Lun+11a, Chapter 5] where a context is "the premises for and the background of the risk analysis. This includes the purposes of the analysis and to whom the risk analysis is addressed". A context can be implied to include all required information to do a risk assessment for a software system, for instance, requirements model of the software system, domain assumptions, the targets needed to protect and so on. The elements in a context, however, may change and evolve over time due to numerous reasons *e.g.,* introduction of new requirements, threats; changes in security standards, regulation. This makes the context changed.

Figure 9.1 presents the conceptual models, expressed as a UML class diagram, on which the risk-evolution approach builds. A *Context Evolution Model* is a collection of evolution rules, which captures the evolutions of context. An *Evolution Rule* is either *Observable Rule*, or *Controllable Rule*. The former captures the evolutions of a context. The latter captures all possible alternatives addressing risks within a context. Further discussion on evolution rules is provided in Section 9.2.3. An *Evolution Rule* has one before context and many after contexts. A *Context* is one mentioned before. A context can be enriched with the output of the risk assessment. A *Risk Countermeasure Alternative* includes a collection of countermeasures, and a list of risks with residual risk levels after applying the countermeasures. Each risk countermeasure alternative is quantified with *Evolution Metric*(s), which are detailed in Section 9.2.4.

Table 9.1 briefs the steps of the proposed risk-evolution approach. The details of these steps are elaborated next.

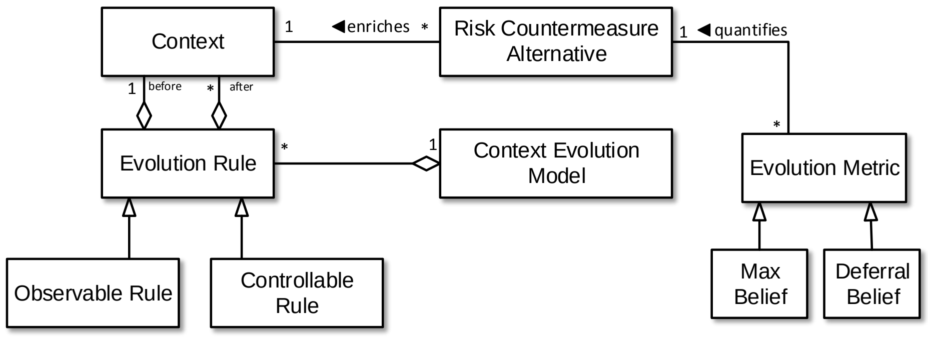


Figure 9.1: The conceptual model of the proposed risk-evolution approach.

9.2.1 Step 1 – Identify Evolving Contexts

This step takes all documents about the planned and potential changes of the system as inputs. We consider four different evolution perspectives: *maintenance, before-after*, and *continuous evolution*, which are discussed in [Lun+11a, Chap. 15]. The *maintenance* perspective relates to the outdate of a risk document of an existing system. Hence it is not the focus of this work. The *before-after* perspective predicts future contexts by anticipating planned and unplanned changes in the current context. The *continuous* evolution perspective predicts the evolution of the current context over time based on planned gradual changes.

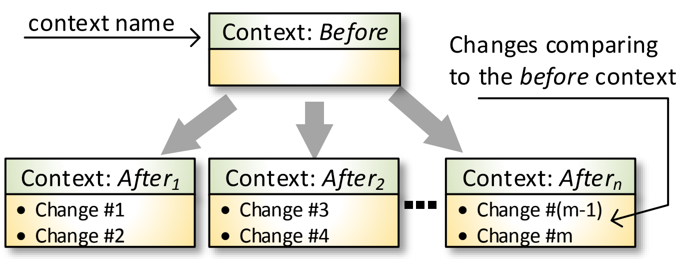
We abuse the notation of *before* and *after* contexts to represent these evolution perspectives (except the *maintenance* one). Figure 9.2(a) demonstrates the *before-after* evolution perspective. A context is depicted as a rectangle with child compartments. The first compartment shows the context name, and the second compartments exhibits the changes comparing to the *before* context. In this perspective, a *before* context might have many possibilities to evolve to other *after* contexts, denoted as *evolution possibility*. At the end of the day, exact one possibility materializes. Each evolution possibility associates with an *evolution probability*, which is the likelihood that a possibility materializes. Figure 9.2(b) illustrates the *continuous* evolution perspective where changes happen continuously. The *before* context at current time t_0 might evolve an *after* context at time t_1, which might continuously evolve at time t_2, and so forth.

After contexts can be identified by using any input document that describes potential changes (either planned or unplanned) in the current context. Unplanned changes could be anticipated by domain experts by using several techniques such as brainstorming with chalk and blackboard, or techniques for requirements changes anticipation. Readers are referred

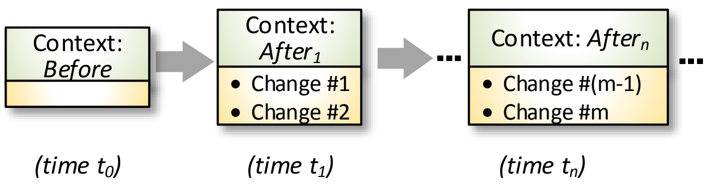
Table 9.1: The steps of the proposed risk-evolution approach.

Step 1 *Identify evolving contexts:*	
DESCRIPTION	Identify all possible changes that would change the risk picture of the system. Changes could be planed or not.
INPUT	any document of changes in context.
OUTPUT	set of contexts, including the current and evolved ones. Evolved contexts are associated with evolution possibilities.
Step 2 *Perform risk assessment:*	
DESCRIPTION	Apply an existing risk assessment method on each identified context. Also, the risk countermeasure alternatives are expected to as a part of output of the risk assessment method.
INPUT	contexts identified in Step 1
OUTPUT	risk countermeasure alternatives
Step 3 *Model context evolution:*	
DESCRIPTION	Establish the context evolution model from the identified contexts and their correponsing risk countermeasure alternatives by using evolution rules.
INPUT	contexts with evolution probabilities (Step 1), risk countermeasure alternatives (Step 2)
OUTPUT	context evolution model
Step 4 *Perform evolution analysis:*	
DESCRIPTION	Run the evolution analysis on the established context evolution model to calculate the evolution metrics for each risk countermeasure alternatives to support the decision making process.
INPUT	context evolution model
OUTPUT	context evolution model quantified by evolution metrics

to [Lam09b, Chap. 6] for a more detailed discussion of these techniques. The evolution probabilities are the experts' belief that evolution possibilities might happen. The probability semantics is accounted by using the game-theoretic approach described in Section 5.2.



(a) Before-after evolution



(b) Continuous evolution

Figure 9.2: The evolution perspectives of contexts

9.2.2 Step 2 – Perform Risk Assessment

In this step, we employ a state-of-the-art risk assessment method (*e.g.*, Attack Trees [Sch99], Cause-Consequence Diagrams [ML05], and CORAS [Lun+11a]) to perform risk assessment for identified contexts. The outcome of this step is list of risk countermeasure alternatives, which are also the output of a risk assessment method.

A *risk countermeasure alternative* includes a list of countermeasures, and the residual risks (with residual risk level) of a system after implementing the countermeasures. A countermeasure could be a security controls (*e.g.*, technology, policy), or a high level security requirement that mitigates risks. A risk level is a pair of the likelihood by which a risk might occur, and its impact. Based on risk level, a risk is categorized, such as *acceptable* or *unacceptable*. A residual risk level is the risk level after implementing countermeasures. Figure 9.3(a) depicts an example risk matrix where the risk R1 is unacceptable and the risk R2 is acceptable. Figure 9.3(b) shows the reduction of R1 where the risk level of R1 is reduced from unacceptable to acceptable by applying the countermeasure C1. This risk reduction may also be represented in text format, for example, R1: $\langle l, I\rangle \xrightarrow{C1} \langle l', I'\rangle$, where $\langle l, I\rangle$ and $\langle l', I'\rangle$ are the risk levels without and with applying countermeasure C1 . We intentionally do not specify the scales of likelihood and impact in Figure 9.3 because some might prefer qualitative while others might prefer to have quantitative scales.

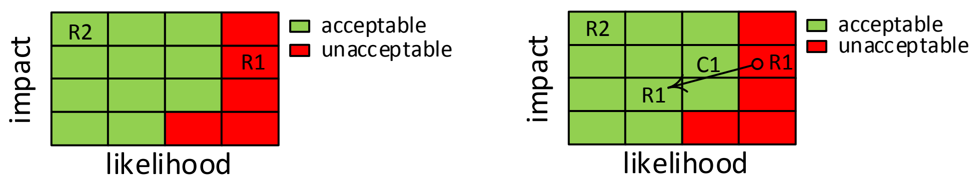
These information can be provided by many state-of-the-art risk assessment methods such as Fault Tree Analysis [IEC90], Event Tree Analysis [Iec], Attack Trees [Sch99], Cause-Consequence Diagrams [Rob+01; ML05], Bayesian networks [Cha91], and CORAS [Lun+11a]. Hence the proposed method is compatible with these methods.

When performing risk assessment on *after* contexts, we can do either a full risk assessment from scratch, or an incremental risk assessment taking advantage on the risk assessment on the *before* context. Needless to say, the former strategy does not use resources efficiently. The latter is better since it only addresses the changed parts of the *after* context comparing to the *before* context [Lun+11a, Chap.15].

9.2.3 Step 3 – Model Context Evolution

This step takes the identified contexts and their corresponding risk countermeasure alternatives to establish the context evolution model. We employ the modeling from Section 5.1 to model the context evolution in terms of evolution rules. There are two kinds of rules: *observable rule* and *controllable rule*. The former captures the way how the context evolves. The latter captures different alternatives to address risks in each context. An evolved context, as aforementioned, is foreseen with a certain evolution probability. For the sake of simplicity, we assume that the evolving contexts identified in Step 1 are complete and mutual exclusive. In other words, exact one of the *after* contexts materializes at the end.

Let $\mathscr{C}$ be a context, and $\mathscr{C}_i$ be the i^{th} *after* context of $\mathscr{C}$, and ca_j be a risk countermeasure alternative of $\mathscr{C}$. The observable rule $r_o(\mathscr{C})$ and controllable rules $r_c(\mathscr{C})$ are described as



(a) Risk matrix (b) Residual risk level

Figure 9.3: Risk level and Residual risk level.

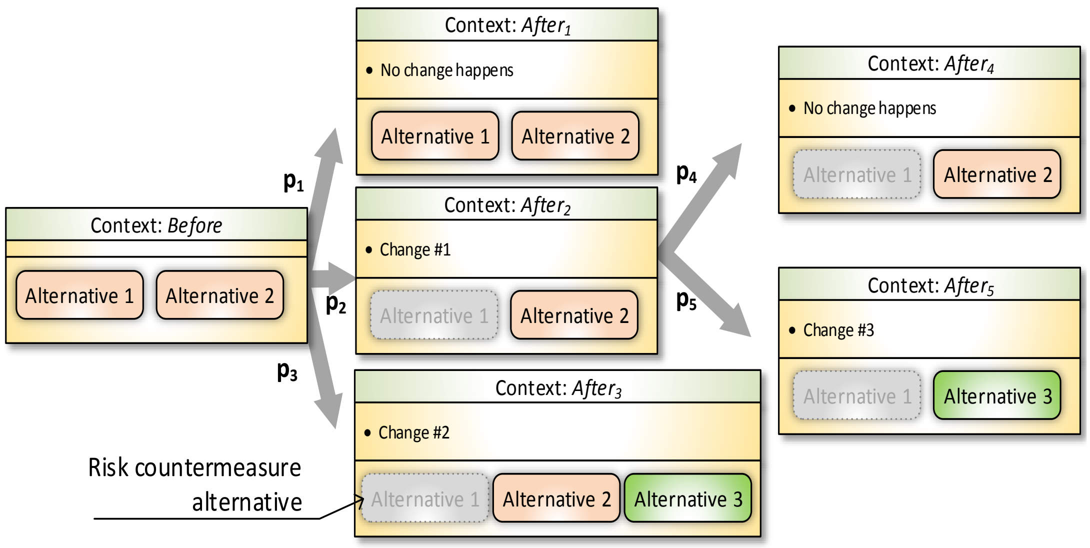


Figure 9.4: The context evolution model.

follows.

$$r_o(\mathscr{C}) = \left\{ \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \,\middle|\, \sum_{i=1}^{n} p_i = 1 \right\} \tag{9.1}$$

$$r_c(\mathscr{C}) = \{\mathscr{C} \to ca_j \,|\, j = 1..m\} \tag{9.2}$$

where n is the number of *after* contexts of $\mathscr{C}$; p_i is the evolution probability for which $\mathscr{C}$ evolves to $\mathscr{C}_i$; m is the number of risk countermeasure alternatives of $\mathscr{C}$. The sum of all p_i is 1 since the *after* contexts are complete and mutual exclusive.

The *before-after* evolution perspective is represented by an observable rule. The *continuous* evolution perspective is represented as a sequence of observable rules where the current context of an observable rule is the *after* context of another observable rule, so on and so forth.

Figure 9.4 shows a graphical visualization of the context evolution model of the *continuous* evolution perspective. The observable rule is denoted by connections from a *before* context to *after* contexts. The decorators on the connections are the evolution probabilities. To denote the controllable rule, the rectangles representing context are extended with a new compartment containing risk countermeasure alternatives, which are represented by round rectangles. The controllable rule then is understood as different risk countermeasure alternatives of a context.

9.2.4 Step 4 – Perform Evolution Analysis

This step performs an evolution analysis on the context evolution model. The analysis employs evolution metrics from Section 5.3. In particular for the field of evolving risks, the evolution metrics aim to quantify to what extent a risk countermeasure alternative can resist the evolution. This analysis relies on three quantitative metrics: *Max Belief*, *Residual Disbelief*, and *Max Disbelief*.

Max Belief (*MaxB*): is the maximum belief that a risk countermeasure alternative will be appropriate if evolution happens. By term *appropriate*, we mean the residual risks after applying the countermeasure alternative in the evolved contexts will still be acceptable. So, the system will still be safe.

Residual Disbelief (*ResD*): is the belief that a risk countermeasure alternative will be inappropriate after evolution happens. It is also the belief by which the implementation of the risk countermeasure alternative should be delayed until the context is clearly known.

Max Disbelief (*MaxD*): is the maximum belief that a risk countermeasure alternative will be inappropriate if evolution happens.

We define a binary function appropriate() that takes two inputs: a context $\mathscr{C}$, and a risk countermeasure alternative *ca*, to produce 1 if *ca* is appropriate within $\mathscr{C}$, or 0 otherwise. The *Max Belief* and *Residual Disbelief* of *ca* for the *before-after* evolution of the context $\mathscr{C}$ are as follows.

$$MaxB(ca|_{\mathscr{C}}) = \max_{\{\langle \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \rangle \in r_o(\mathscr{C}) | \text{appropriate}(\mathscr{C}_i, ca) = 1\}} p_i \tag{9.3}$$

$$ResD(ca|_{\mathscr{C}}) = 1 - \sum_{\{\langle \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \rangle \in r_o(\mathscr{C}) | \text{appropriate}(\mathscr{C}_i, ca)\}} p_i \tag{9.4}$$

$$MaxD(ca|_{\mathscr{C}}) = \max_{\{\langle \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \rangle \in r_o(\mathscr{C}) | \text{appropriate}(\mathscr{C}_i, ca) = 0\}} p_i \tag{9.5}$$

For the *continuous* evolution of the context $\mathscr{C}$, we extend concept *Max Belief*, *Residual Disbelief*, and *Max Disbelief* to *Continuous Max Belief* ($MaxB^*$), *Continuous Residual Disbelief* ($ResD^*$), and *Continuous Max Disbelief* ($MaxD^*$). The formulas of these extended metrics are as follows.

$$MaxB^*(ca|_{\mathscr{C}}) = \begin{cases} \mathsf{appropriate}(\mathscr{C}, ca) & \text{if } \mathscr{C} \text{ does not evolve,} \\ \max\limits_{\{\langle \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \rangle \in r_o(\mathscr{C}) | \mathsf{appropriate}(\mathscr{C}_i, ca) = 1\}} p_i \cdot MaxB^*(ca|_{\mathscr{C}_i}) & \text{otherwise.} \end{cases} \tag{9.6}$$

$$ResD^*(ca|_{\mathscr{C}}) = \begin{cases} 1 - \mathsf{appropriate}(\mathscr{C}, ca) & \text{if } \mathscr{C} \text{ does not evolve,} \\ 1 - \sum\limits_{\{\langle \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \rangle \in r_o(\mathscr{C}) | \mathsf{appropriate}(\mathscr{C}_i, ca) = 1\}} p_i \cdot (1 - ResD^*(ca|_{\mathscr{C}_i})) & \text{otherwise.} \end{cases} \tag{9.7}$$

$$MaxD^*(ca|_{\mathscr{C}}) = \begin{cases} \mathsf{appropriate}(\mathscr{C}, ca) & \text{if } \mathscr{C} \text{ does not evolve,} \\ \max\limits_{\{\langle \mathscr{C} \xrightarrow{p_i} \mathscr{C}_i \rangle \in r_o(\mathscr{C}) | \mathsf{appropriate}(\mathscr{C}_i, ca) = 0\}} p_i \cdot MaxD^*(ca|_{\mathscr{C}_i}) & \text{otherwise.} \end{cases} \tag{9.8}$$

In (9.6)(9.7), by saying *the context $\mathscr{C}$ does not evolve*, we mean that no evolving context of $\mathscr{C}$ is identified. This does not mean $\mathscr{C}$ stops evolving, but its evolution may be ignored in the analysis due to some reason.

After this analysis, each risk countermeasure alternative is quantified with two evolution metrics: *Max Belief*, *Residual Disbelief*, and *Max Disbelief*. Together with the benefit and cost of a risk countermeasure alternative, these evolution metrics can support designers in selecting the most evolution-resilient alternative for the system. To the perspective of evolution-resilience, a better alternative is one that has a higher *Max Belief*, a lower *Residual Disbelief*, and a lower *Max Disbelief*.

9.3 The Running Example

This section exemplifies the proposed method by a running example taken from an industrial project in the ATM domain: the SWIM project [Pro08; Adm09]. The SWIM project aims to provide consistent, efficient, transparent, and secure means for information interchange among ATM systems. This example focuses on the Messaging Service, a part of SWIM, which is responsible for a common and reliable layer to exchange messages within the SWIM architecture.

Figure 9.5 illustrates the Messaging Service (MSG) architecture. In side the MSG, The Mediator component is in charge of message transformation. The Message Routing is in

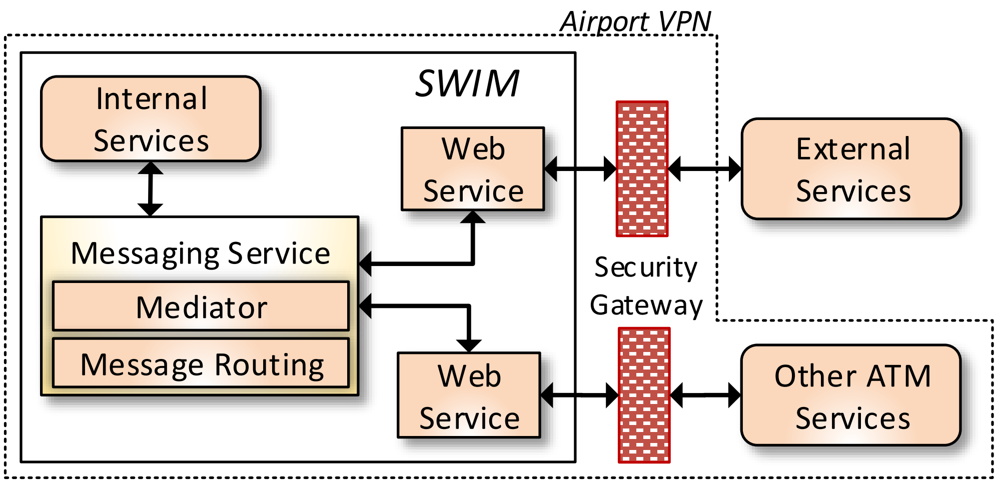


Figure 9.5: The architecture of the Messaging Service within SWIM.

charge of point-to-point message transmission. Internal Services within SWIM can directly connect to the MSG. The communications between other ATM Services, External Service and MSG are done through the Web Service interface, and are subject to the supervision of the Security Gateway. Hereafter, we apply steps in the proposed method on the running example.

Applying Step 1: For simplicity, we do not show the *before* context. Instead, we only describe anticipated changes that might happen in the *before* context:

C1 The business rules might become more complex, therefore more expressive policies are required to protect confidential resources.

C2 The ATM network might change from private network to public internet.

Since these changes are independent, either only C1, or only C2, or both might happen. Consequently, the following *after* contexts are identified with corresponding probabilities of occurrence:

- *After*$_1$ ($\mathscr{C}_1$): no change will happen – 0.10.
- *After*$_2$ ($\mathscr{C}_2$): only C1 will happen – 0.30.
- *After*$_3$ ($\mathscr{C}_3$): only C2 will happen – 0.35.

- *After*$_4$ ($\mathscr{C}_4$): both C1 and C2 will happen – 0.25.

Applying Step 2: We perform risk assessment for all identified contexts. We only consider one risk resulting from the risk assessment on the *before* context of MSG: *"unwanted access to confidential resources"* (R1). The corresponding countermeasure, in a high level of abstraction, is: *"implements authentication/authorization mechanism"* (SR-1). It is further refined into three operational countermeasure alternatives: ca_1:X.509+SAML[2], ca_2:Kerberos+LDAP[3], and ca_3:SAML+XACML[4]. Each countermeasure alternative includes two components: one for authentication (the first component), and another one for authorization (the last one). We use RLV1, RLV1', and RLV1" to respectively represent for the residual risk level of R1 after applying ca_1, or ca_2, or ca_3

We do not describe the risk assessment on *after* contexts, but discuss only the difference between the risk assessment output of these contexts and that of the *before* context, as shown below.

- $\mathscr{C}_1$: as same as the *before* context.

- $\mathscr{C}_2$: a new risk is identified: *"unauthorized access to confidential resources because the authorization mechanism cannot capture expressive policies"*[5] (R2). The countermeasure for R2 is: *"implement a high expressive authorization mechanism"* (SR-2). The refined countermeasure alternative will be either ca_2, or ca_3. Similarly, the residual risk levels of R2 by applying ca_2 and ca_3 are respectively denoted as RLV2 and RLV2'.

- $\mathscr{C}_3$: a new risk is identified: *"system collapses due to the malicious attacks on centralized key server"* (R3). The countermeasure for R3 is: *"implement a robust key management mechanism"* (SR-3). The refined countermeasure alternatives will be either ca_1, or ca_3. Similarly, the residual risk levels of R3 by applying ca_2 and ca_3 are respectively denoted as RLV2 and RLV2'.

- $\mathscr{C}_4$: both risks R2 and R3 are identified. Consequently, the only refined countermeasure alternative will be SAML+XACML. The residual levels of R2 and R3 are denoted as RLV2, and RLV3 respectively.

[2]Security Assertion Markup Language
[3]Lightweight Directory Access Protocol
[4]eXtensible Access Control Markup Language
[5]This is a limit of SAML[RR09, chapter 6]

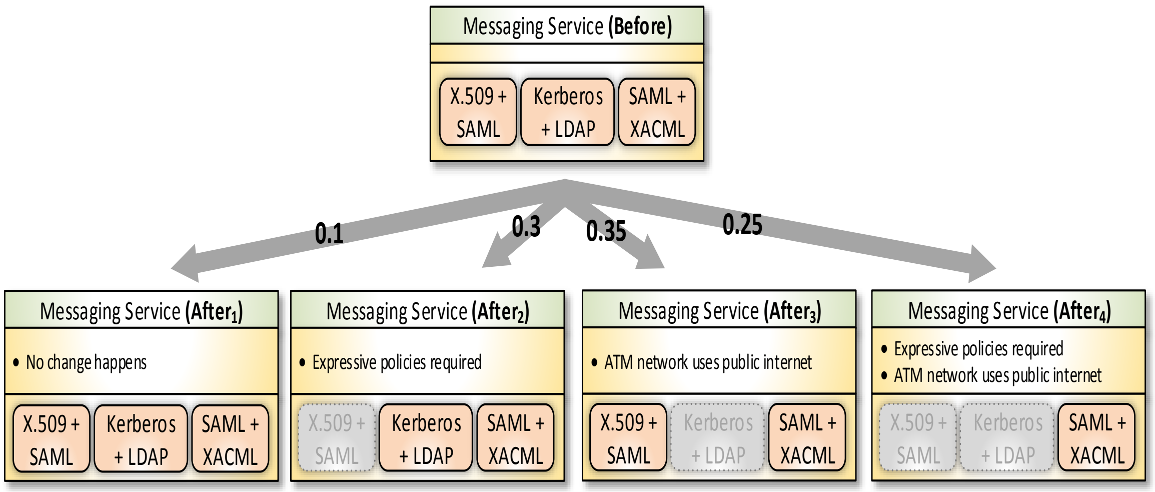

Messaging Service (Before)
X.509 + SAML
Kerberos + LDAP
SAML + XACML
0.1
0.3
0.35
0.25
Messaging Service (After₁)
No change happens
Messaging Service (After₂)
Expressive policies required
Messaging Service (After₃)
ATM network uses public internet
Messaging Service (After₄)
Expressive policies required
ATM network uses public internet

Figure 9.6: The evolution of Messaging Service.

Applying Step 3: Let $\mathscr{C}_0$ be the *before* context, the evolution rules are as follows:

$$r_o(()\mathscr{C}_0) = \left\{\mathscr{C}_0 \xrightarrow{0.1} \mathscr{C}_1, \mathscr{C}_0 \xrightarrow{0.30} \mathscr{C}_2, \mathscr{C}_0 \xrightarrow{0.35} \mathscr{C}_3, \mathscr{C}_0 \xrightarrow{0.25} \mathscr{C}_4\right\},$$

$$r_c(()\mathscr{C}_1) = \{\mathscr{C}_1 \rightarrow \langle\{\text{X.509,SAML}\}, \text{RLV1}\rangle, \mathscr{C}_1 \rightarrow \langle\{\text{Kerberos,LDAP}\}, \text{RLV1}\rangle,$$
$$\mathscr{C}_1 \rightarrow \langle\{\text{SAML,XACML}\}, \text{RLV1}\rangle\},$$

$$r_c(()\mathscr{C}_2) = \{\mathscr{C}_2 \rightarrow \langle\{\text{Kerberos,LDAP}\}, \text{RLV1, RLV2}\rangle, \mathscr{C}_2 \rightarrow \langle\{\text{SAML,XACML}\}, \text{RLV1, RLV2}\rangle\},$$

$$r_c(()\mathscr{C}_3) = \{\mathscr{C}_3 \rightarrow \langle\{\text{X.509,SAML}\}, \text{RLV1, RLV3}\rangle, \mathscr{C}_3 \rightarrow \langle\{\text{SAML,XACML}\}, \text{RLV1, RLV3}\rangle\},$$

$$r_c(()\mathscr{C}_4) = \{\mathscr{C}_4 \rightarrow \langle\{\text{SAML,XACML}\}, \text{RLV1, RLV2, RLV3}\rangle\}$$

Figure 9.6 exhibits the context evolution model of the running example. In the figure, to improve the readability, we represent the risk countermeasure alternative by its corresponding countermeasures.

Applying Step 4: From (9.3)(9.4), we calculate the *Max Belief*, *Residual Disbelief*, and *Max Disbelief* of each risk countermeasure alternative. The results are reported in Table 9.2.

From the evolution-resilient perspective, the alternative SAML + XACML is the best since it has the highest *Max Belief*, the lowest *Residual Disbelief* and the lowest *Max Disbelief* . The alternative X.509 + SAML places the second, and the last one Kerberos + LDAP is the third. These information are combined with the residual risks to support the countermeasure selection.

Table 9.2: The *Max Belief* and *Residual Disbelief.*

Risk Countermeasure Alternative				
Counermeasures	Residual Risks	**MaxB**	**ResD**	**MaxD**
ca_1:X.509 + SAML	R1:RLV1, R3:RLV3	0.35	0.55	0.30
ca_2:Kerberos + LDAP	R1:RLV1’, R2:RLV2	0.30	0.60	0.35
ca_3:SAML + XACML	R1:RLV1”, R2:RLV2’, R3:RLV3’	0.35	0	0

9.4 Chapter Summary

In this chapter, we considered a target software system in a period of time with different contexts. We captured the evolution of risks, and then reason on their uncertainties to quantify evolution-resilient countermeasure alternatives in terms of three evolution metrics.

It would be even more beneficial for decision makers if we can also provide more information to assess these countermeasure alternatives within a particular context at a particular point of time. Hence, in the next Chapter 10 , we go deeper into the risks of the target software system at a particular context. We shall evaluate every countermeasure alternative in a context in terms of cost and benefit.

CHAPTER 10

SELECTING COST-EFFECTIVE RISK COUNTERMEASURES

This chapter proposes a method to integrate the cost assessment into risk analysis to aid the selection of cost-effective risk countermeasures. The proposed method makes use of a risk graph model annotated with potential countermeasures, estimates for their cost and effect. A calculus is then employed to reason about this model in order to support decision by means of decision diagrams. We exemplify the instantiation of the method in the CORAS method for security risk analysis. We also enrich the capacity of evolution modeling and reasoning to the proposed method by applying the risk-evolution approach described in the previous chapter.

THE previous chapter has presented a risk-evolution approach that operates on any risk assessment methods. This is an advantage of it, but the representation the evolution is limited in a very generic way. This chapter presents a method that support the selection of cost-effective risk countermeasures based on risk graph. Then we apply the proposed risk-evolution approach to the method. Since we base the method on risk graph, which is generic enough to represent for many risk assessment methods, the risk-evolution approach (Chapter 9) plus the method in this chapter could deliver a more fine-grain representation of evolution in risk assessment, while still enabling the capability to deal with evolution.

The organization of this chapter is as follows. In Section 10.1 we present the proposed method, including the steps, the modeling support and the analysis techniques. Section 10.2 details the calculus for propagating and aggregating reduction effect and effect dependency. Section 10.3 exemplifies the method in CORAS. Section 10.4 discusses how to model evolution in risk graph and how to perform reasoning about evolution. Finally, Section 10.5 summarizes this chapter.

10.1 The Proposed Method

As illustrated in Figure 10.1, the proposed method takes a risk model resulting from a risk assessment and the associated risk acceptance criteria as input and delivers a set of recommended countermeasure alternatives as output. Hence, the method assumes that risk assessment has already been conducted, i.e. that risks have been identified, estimated and evaluated and that the overall risk analysis process is ready to proceed with the risk treatment phase. We moreover assume that the risk analysis process complies with the ISO 31000 risk management standard [ISO09], in which risk countermeasure is the final phase. The method consists of three main steps as follows:

STEP 1 *Annotate risk model:* Identify and document countermeasures. The results are documented by annotating the risk model taken as input with relevant information including the countermeasures, their cost, their reduction effect (i.e., effect on risk value), as well as possible effect dependencies (i.e., countervailing effects among countermeasures).

STEP 2 *Perform countermeasure analysis:* Enumerate all countermeasure alternatives and reevaluate the risk picture for each alternative. The analysis makes use of the annotated risk model and a calculus for propagating and aggregating the reduction effect and effect dependency along the risk paths.

STEP 3 *Perform synergy analysis:* Perform synergy analysis for selected risks based on decision diagrams. The outcome is recommended countermeasure alternatives, which cost-effectively mitigate the selected risks.

Figure 10.2 presents the conceptual model, expressed as a UML class diagram [Rum+04] on which the proposed method builds. A *Risk Model* is a structured way of representing an unwanted incident and its causes and consequences using graphs, trees or block diagrams [Rob+01], or tables [Lun+11a]. An unwanted incident is an event that harms or reduces the

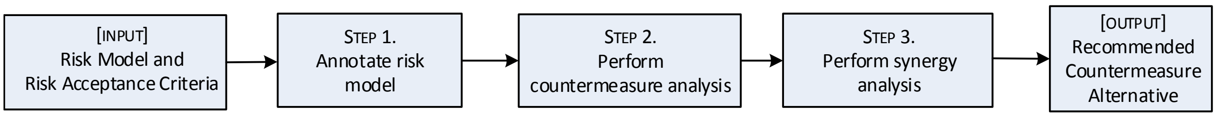


Figure 10.1: Steps of the proposed method.

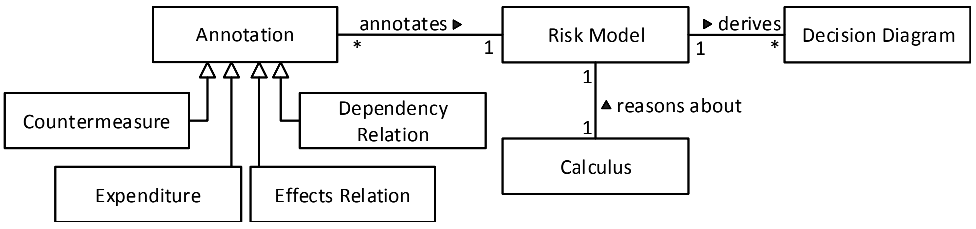


Figure 10.2: Conceptual model.

value of an asset, and a risk is the likelihood of an unwanted incident and its consequence for a specific asset [ISO09]. A *Countermeasure* mitigates risk by reducing its likelihood and/or consequence. The *Expenditure* includes the expenditure of countermeasure implementation, maintenance and so on. The *Reduction Effect* captures the extent to which a countermeasure mitigates risks. The *Reduction Effect* could be the reduction of likelihood, and/or the reduction of consequence of a risk. The *Effect Dependency* captures the countervailing effect among countermeasures that must be taken into account in order to understand the combined effect of identified countermeasures. The *Calculus* provides a mechanism to reason about the annotated risk model. Using the *Calculus*, we can perform countermeasure analysis on annotated risk models to calculate the residual risk value for each individual risk. A *Decision Diagram* facilitates the decision making process based on the countermeasure analysis.

10.1.1 Input Assumptions

The input required by the proposed method is a risk model generated by a risk assessment, and the corresponding risk acceptance criteria. To ensure that the method is compatible with several risk modeling techniques, we expect the risk model could be understood as a risk graph instantiation. A risk graph [Bra+10] is a common abstraction of several established risk modeling techniques such as Fault Tree Analysis (FTA) [IEC90], Event Tree Analysis (ETA) [Iec], Attack Trees [Sch99], Cause-Consequence diagrams [Rob+01; ML05], Bayesian networks [Cha91], and CORAS risk diagrams [Lun+11a]. Hence, the method complies with these

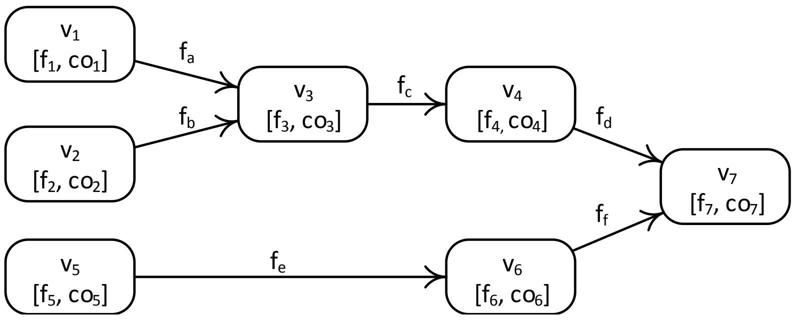
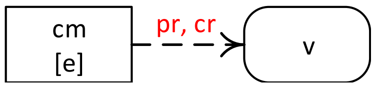


Figure 10.3: Risk graph.



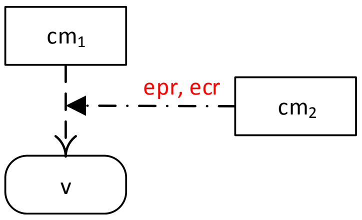


Figure 10.4: Countermeasure with *treats* relation. Figure 10.5: Effect dependency relation.

risk modeling techniques, and can be instantiated by them.

A risk graph is a finite set of vertices and relations (see Figure 10.3). Each vertex v represents a threat scenario, *i.e.*, a sequence of events that may lead to an unwanted incident, and can be assigned a probability p, and a consequence co. A *leads-to* relation from v_1 to v_2 means that the former threat scenario may lead to the latter. Probabilities on the relations are conditional probabilities indicating the likelihood of the former to lead to the latter when the former occurs.

10.1.2 Detailing of Step 1 – Annotate Risk Model

This step annotates the input risk model with required information for further analysis. There are four types of annotation as follows:

Countermeasure: In risk graphs, countermeasures are represented as rectangles. In Figure 10.4 there is one countermeasure and this is named *cm*.

Expenditure: In risk graphs, expenditure is expressed within square brackets following the countermeasure name (e in Figure 10.4). This is an estimated of the total amount of money spent to ensure the mitigation of countermeasure including expenditure of implementation, deployment, maintenance, and so on.

Reduction effect: In risk graphs, reduction effect is represented by a dashed arrow decorated by two numbers (*pr* and *cr* in Figure 10.4). It captures the mitigating effect of a countermeasure in terms of reduced likelihood (*i.e., probability reduction - pr*), reduced consequence (*i.e., consequence reduction - cr*), or both. Both pr and cr are relative percentage values, *i.e.,* $pr, cr \in [0,1]$.

Effect dependency: In risk graphs, effect dependency is represented by a dash-dot arrow with solid arrowhead decorated by two numbers (*effect on probability reduction (epr),* and *effect on consequence reduction (ecr)* in Figure 10.5). It captures the impact of a countermeasure to the reduction effect of another, *i.e.,* it can increase or decrease *pr* and/or *cr* of another countermeasure. The *epr* impacts *pr* while the *ecr* impacts *cr.* Both *epr* and *ecr* are relative percentage values, *i.e.,* $epr, ecr \in [0,1]$.

10.1.3 Detailing of Step 2 – Countermeasure Analysis

The countermeasure analysis in this step is conducted for every individual risk of the annotated risk model. The analysis enumerates all possible countermeasure combinations, called *countermeasure alternatives* (or *alternatives* for short) and evaluates the residual risk value (*i.e.,* residual consequence and probability) with resect to each alternative to determine the most efficient one. Residual risk value is obtained by propagating the reduction effect along the risk model to get the revised risk values. To this purpose, we have developed a calculus with propagation rules. An example of rule is shown as below.

Rule 2.1 (Countermeasure) If there is a *treats* relation from countermeasure cm to vertex $v(p, co)$ with probability reduction pr and consequence reduction cr, we have:

$$\frac{cm \xrightarrow{pr,cr} v \quad v(p, co)}{v(p \cdot \overline{pr}, co \cdot \overline{cr})}$$

Rule 2.1 applies to countermeasures as depicted in Figure 10.4. The probability reduction pr on the probability p of the scenario means that p is reduced by $pr \in [0,1]$. Hence, p is multiplied by $\overline{pr} = 1 - pr$. Likewise for the consequence reduction. The complete list of rules is available in [Tra+13b].

From the leftmost threat scenarios (*i.e.,* scenarios that have only outgoing *leads-to* relations), probabilities assigned to threat scenarios are propagated to the right. During the propagation, probabilities assigned to *leads-to* relations and reduction effects of countermeasures are taken into account. Finally, the propagation stops at the rightmost threat sce-

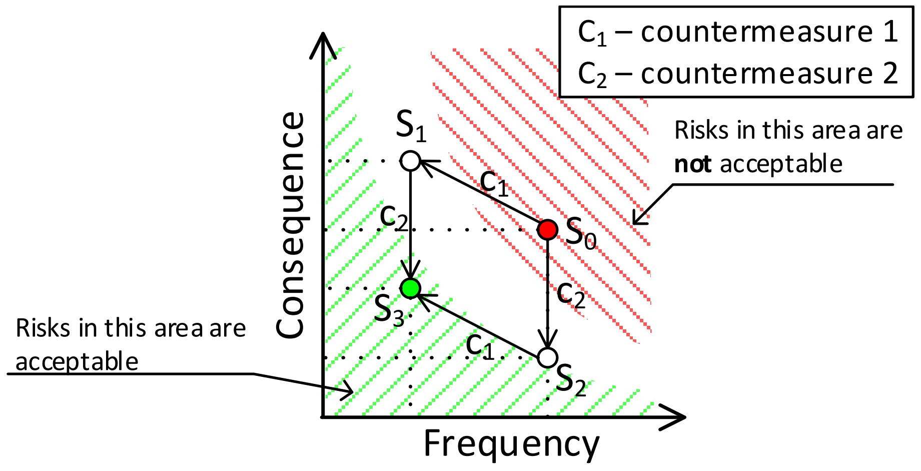


Figure 10.6: Decision diagram.

narios (*i.e.,* scenarios that have only incoming *leads-to* relations). Based on the results from the propagation, the residual risk value is computed.

Decision Diagram (Figure 10.6) is a directed graph used to visualize the outcome of a countermeasure analysis. A node in the diagram represents a *risk state*, which is a triplet of probability, consequence, and countermeasure alternatively of the risk in analyzed. The probability and consequence are respective the X and Y coordinate of the node. The countermeasure alternative is annotated on the path from the *initial state* S_0 where no countermeasure applied to the node. Notice that we ignore all states whose residual consequence and probability are both greater than those of S_0 since it is useless to implement such countermeasures.

10.1.4 Detailing of Step 3 – Synergy Analysis

The aim of the synergy analysis is to recommend a cost-effective countermeasure alternative for mitigating all risks, namely *global countermeasure alternative.* Such recommendation is based on the decision diagrams for the individual risks (generated in Step 2), and the risk acceptance criteria, and the overall costs (OC) of global countermeasure alternatives, which are calculated as follows:

$$\mathrm{OC}(ca) = \sum_{r \in R_{ca}} \mathrm{rc}(r) + \sum_{cm \in ca} \mathrm{cost}(cm) \tag{10.1}$$

where ca is a global countermeasure alternative; R_{ca} is the set of risks with respect to the global countermeasure alternative ca; rc() is a function that yields the loss (in monetary value) due to the risk taken as argument (based on its probability and consequence); cost() is a function that yields the expenditure of the countermeasure taken as argument.

The synergy analysis is decomposed into three following substeps:

STEP 3A *Identify global countermeasure alternatives* : Identify the set of global countermeasure alternatives CA for which all risks are acceptable with respect to the risk acceptance criteria. Decision diagrams of individual risks can be exploited for identifying CA.

STEP 3B *Evaluate global countermeasure alternatives* : If no such global countermeasure alternative is identified ($CA = \emptyset$), do either of the following:

- Identify new countermeasures and go to Step 1, or
- Adjust the risk acceptance criteria and go to Step 3A

If some global countermeasure alternatives are identified ($CA \neq \emptyset$), select a global countermeasure alternative $ca \in CA$ with the lowest overall cost OC(ca).

STEP 3C *Decide cost-effective global countermeasure alternative:* : If OC(ca) is acceptable (for the customer company in question) then terminate the analysis. Otherwise, identify more (cheaper and/or more effective) countermeasures and go to Step 1.

The above procedure may of course be detailed further based on various heuristics. For example, in many situations, with respect to Step 3A, if the global countermeasure alternative $ca \in CA$, then we do not have to consider other global countermeasure alternative ca' such that $ca' \subseteq ca$. However, we do not go into these issues here.

10.2 The Calculus

10.2.1 Rules for Risk Graphs

In this section, we present the formal calculus for risk graphs. The calculus extends the calculus presented in [Bra+10] with rules to deal with treatment effect and dependency. Note that for all rules there is an implicit assumption that the premises and the conclusion are type-correct.

Rule 1.1 (Relation) If there is a direct relation from v to v', we have:

$$\frac{v(p) \quad v \xrightarrow{p'} v'}{(v \sqcap v')(p \cdot p')}$$

Rule 1.2 (Mutually exclusive vertices) If the vertices v and v' are mutually exclusive, we have:

$$\frac{v(p) \quad v'(p)}{(v \sqcup v')(p)}$$

Rule 1.3 (Statistical independent vertices) If the vertices v and v' are statistically independent, we have:

$$\frac{v(p) \quad v'(p')}{(v \sqcup v')(p+p')}$$

Rule 1.4 (Countermeasure) If the countermeasure cm treats vertex v, we have:

$$\frac{cm \xrightarrow{pr,cr}_{r} v \quad v_t(p,co) \quad \neg\exists r' : cm_{r'} \in t}{v_{t\cup\{cm_r\}}(p \cdot \overline{pr}, co \cdot \overline{cr})}$$

Rule 1.5 (Effect dependency) If there is an effect dependency from a countermeasure cm' to a *treats* relation, we have:

$$\frac{cm' \xrightarrow{epr,ecr} \left(cm \xrightarrow{pr,cr} v\right) \quad cm' \notin r}{cm \xrightarrow{pr+epr,cr+ecr}_{r\cup\{cm'\}} v}$$

10.2.2 Rules for Treatment Diagrams

In this section, we present the formal calculus for treatment diagrams. The calculus contains Rule 2.1 to Rule 2.4 from CORAS calculus [Lun+11a, Chapter 13], and new introduced rules to deal with treatment effect and dependency. In the context of treatment diagrams of this work, we work with frequency instead of probabilities. Hence, we replace p with f, pr with fr, and epr with efr from Rule 2.1 to Rule 2.10.

Rule 2.1 (Initiate) For a threat t and scenario/incident v related by the initiates relation, we have:

$$\frac{t \xrightarrow{f} v}{(t \sqcap v)(f)}$$

Rule 2.2 (Leads-to) For the scenarios/incidents v_1 and v_2 related by the leads-to relation, we have:

$$\frac{v_1(f) \quad v_1 \xrightarrow{l} v_2}{(v_1 \sqcap\!\!| \, v_2)(f \cdot l)}$$

Rule 2.3 (Mutually exclusive scenarios/incidents) If the scenarios/incidents v_1 and v_2 are mutually exclusive, we have:

$$\frac{v_1(f) \quad v_2(f)}{(v_1 \sqcup v_2)(f)}$$

Rule 2.4 (Independent scenarios/incidents) If the scenarios/incidents v_1 and v_2 are separate and statistically independent, we have:

$$\frac{v_1(f_1) \quad v_2(f_2)}{(v_1 \sqcup v_2)(f_1 + f_2)}$$

Rule 2.5 (Treatment on scenario) If the treatment tms treats the scenario v, we have:

$$\frac{tms \xrightarrow{fr}_r v \quad v_t(f) \quad \neg\exists r' : tms_{r'} \in t}{v_{t \cup \{tms_r\}}(f \cdot \overline{fr})}$$

Rule 2.6 (Treatment on risk) For treatment tms that mitigates the risk v, we have:

$$\frac{tms \xrightarrow{fr,cr}_r v \quad v_t(f, co) \quad \neg\exists r' : tms_{r'} \in t}{v_{t \cup \{tms_r\}}(f \cdot \overline{fr}, co \cdot \overline{cr})}$$

Rule 2.7 (Treatment on leads-to relation) If the treatment tms treats the leads-to relation relating the scenario v and the scenario/risk v', we have:

$$\frac{tms \xrightarrow{fr}_r (v \xrightarrow{f} v')}{tms \xrightarrow{fr}_r (v \sqcap\!\!| \, v')}$$

Rule 2.8 (Treatment on initiate relation) If the treatment tms treats the initiate relation relating the threat t and the scenario v, we have:

$$\frac{tms \xrightarrow{fr}_r \left(t \xrightarrow{f} v\right)}{tms \xrightarrow{fr}_r (t \sqcap\!\!| \, v)}$$

Rule 2.9 (Treatment on threat) If the treatment tms treats the threat t and t initiates the scenario v, we have:

$$\frac{tms \xrightarrow{fr}_r t \quad t \xrightarrow{f} v}{tms \xrightarrow{fr}_r (t \sqcap\!\!| \, v)}$$

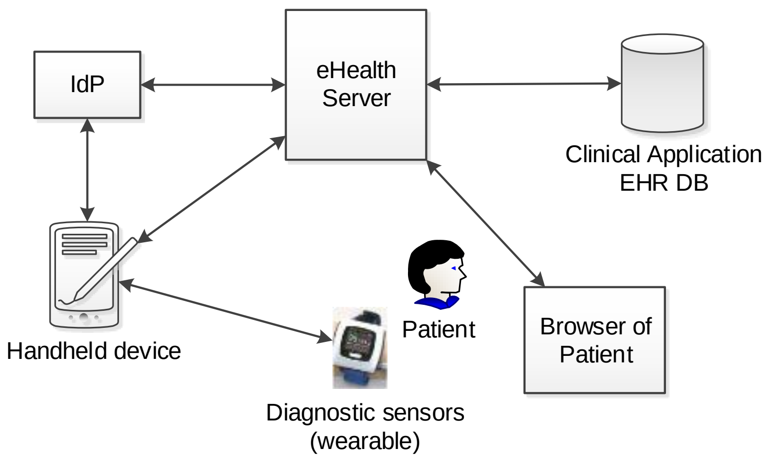
Rule 2.10 (Effect dependency) If the treatment tms' affects the *treats* relation connecting the treatment tms and a scenario/risk v, we have:

$$\frac{tms' \xrightarrow{efr,ecr} \left(tms \xrightarrow{fr,cr} v\right) \quad tms' \notin r}{tms \xrightarrow{fr+efr,cr+ecr}_{r\cup\{tms'\}} v}$$

10.3 Exemplification in CORAS

As a demonstration of applicability, this section instantiates the proposed method into the CORAS method for security risk analysis[Lun+11a] and exemplifies how the resulting extended CORAS method and language can be used to select cost-efficient risk countermeasures in an example drawn from a case study within the eHealth domain [Ome+12].

The *risk model* in the CORAS method is captured by so-called *risk diagrams*. A risk diagram is a causality graph consisting of potential causes (*i.e., threats*) that might (or might not) exploit flaws, weaknesses, or deficiencies (*i.e., vulnerabilities*) causing a series of events (*i.e., threat scenarios*) to happen, which could lead to *unwanted incidents* with certain likelihood and concrete consequence (*i.e., risks*) to a particular *asset*. Threat scenarios and risks are also called core elements in the risk diagram notation.



The patient has one or more monitoring devices in the form of wearable sensors. They provide data to an application in a handheld device, which does some processing on the data, aggregates the results and sends them to the eHealth Server. The patient and his devices are authenticated by an Identity Provider (IdP).

Figure 10.7: Architectural sketch of Patient Monitoring scenario (from [NES11, Figure 3.2])

In the risk diagram, there are two kinds of relationships with assigned likelihoods: *initiate* and *leads-to* relations. The former connects a threat to a core element, and the latter connects a core element to another core element. Likelihoods assigned to *initiate* relations can be either *probabilities* or *frequencies*, whereas, likelihoods assigned to *leads-to* relations are *conditional likelihoods*.

Any risk diagram can be understood as an instantiation of a risk graph; such conversion is formally defined in [Bra+10]. To make the instantiation more comprehensible, we also present a running example that exploits an eHealth scenario proposed by the NESSoS project[NES11] to exemplify the resulting extended CORAS method.

10.3.1 eHealth Running Example: Patient Monitoring

As illustrated in Figure 10.7, patients' behaviors and symptoms are monitored in realtime. This provides an improved basis for disease diagnoses and tailored therapy prescription regiments. Patients are equipped with sensors that continuously collect patients data and send these data to a handheld smart device (*e.g.*, smart phone). This smart device, in turn, sends the patient data to external eHealth servers where the patients' eHealth Records (EHRs) are updated.

The CORAS risk diagram in Figure 10.8 presents a partial result from a risk analysis of the

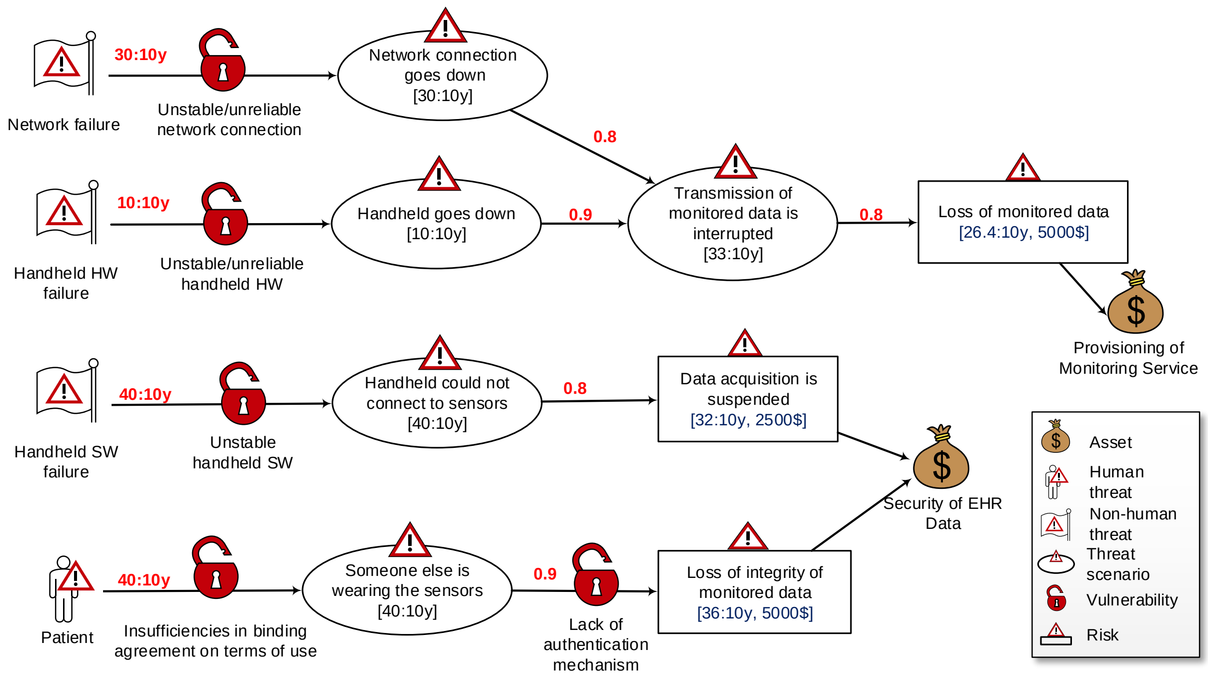

30:10y
Network failure
Unstable/unreliable network connection
Network connection goes down [30:10y]
0.8
10:10y
Handheld HW failure
Unstable/unreliable handheld HW
Handheld goes down [10:10y]
0.9
Transmission of monitored data is interrupted [33:10y]
0.8
Loss of monitored data [26.4:10y, 5000$]
Provisioning of Monitoring Service
40:10y
Handheld SW failure
Unstable handheld SW
Handheld could not connect to sensors [40:10y]
0.8
Data acquisition is suspended [32:10y, 2500$]
Security of EHR Data
40:10y
Patient
Insufficiencies in binding agreement on terms of use
Someone else is wearing the sensors [40:10y]
0.9
Lack of authentication mechanism
Loss of integrity of monitored data [36:10y, 5000$]
Asset
Human threat
Non-human threat
Threat scenario
Vulnerability
Risk

Figure 10.8: Risk diagram of the scenario.

Patient Monitoring scenario [Ome+12]. In this risk diagram, *network failure* exploits the vulnerability *unstable/unreliable network connection* to initiate *network connection goes down*. Likewise, *handheld HW failure* exploits the vulnerability *unstable/unreliable handheld HW* to initiate *handheld goes down*. Both *handheld goes down* as well as *network connection goes down* may lead to the *transmission of monitored data is interrupted*. This, consequently, may lead to *loss of monitored data*, which impacts the *provisioning of monitoring service*. The rest of the diagram is interpreted in the similar manner.

We assume in the following that this diagram is a consistent and complete documentation of risks identified during the risk assessment. We moreover use frequencies to estimate likelihoods of core elements. Reasoning about frequencies in the risk and treatment assessment is also supported by the proposed calculus.

10.3.2 Applying Step 1 – Annotate Risk Model

In this step, we annotate the CORAS risk diagrams according to Step 1 to create CORAS treatment diagrams. Note that in CORAS, countermeasures are referred to as treatments.

Treatment annotation: treatments can apply to most of the elements in a treatment diagram, including all types of core elements, threats, and vulnerabilities. Figure 10.9 shows

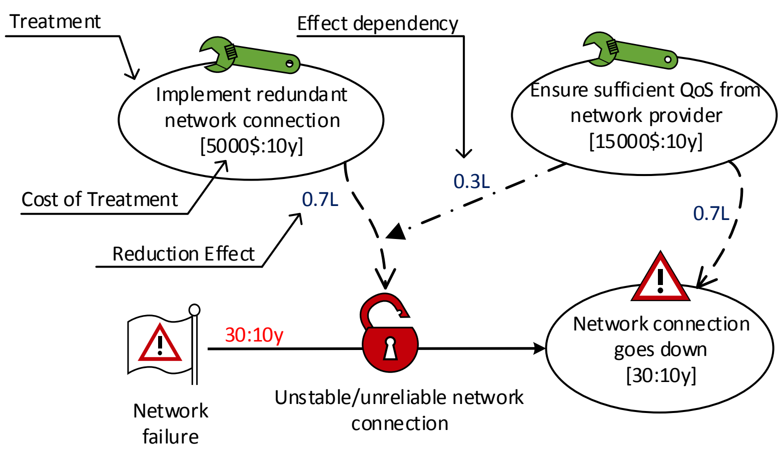

Treatment
Effect dependency
Implement redundant network connection [5000$:10y]
Ensure sufficient QoS from network provider [15000$:10y]
Cost of Treatment
0.7L
0.3L
0.7L
Reduction Effect
30:10y
Network failure
Unstable/unreliable network connection
Network connection goes down [30:10y]

Figure 10.9: Annotated diagram

an example in which a treatment *implement redundant network connection* treats the scenario *network connection goes down*, which was initiated by *network failure* by exploiting the vulnerability *unstable/unreliable network connection*.

Expenditure annotation: the treatment expenditure, annotated as value inside the treatment bubble, is the total expenditure spent for a treatment in a period of time. For instance, in Figure 10.9, the expenditure for *implementing a redundant network connection* is 5000$ in ten year.

Reduction effect annotation: following Step 1, reduction effect in the CORAS instantiation is annotated on *treats* relations as a pair $\langle fr, cr\rangle$, where fr, cr are frequency reduction and consequence reduction, respectively. For example, in Figure 10.9, the frequency of network failure is thirty times in ten years, annotated as 30:10y. In a CORAS diagram, we suffix the value of fr and cr with the letter *'L'* and *'C'*, respectively, to distinguish between them. The treatment *implement redundant network connection* only reduces the frequency (not consequence) of *unreliable network connection* by 0.7 at cost 5000USD:10y. This means the reduced frequency is $(1-0.7)\cdot 30{:}10y = 9{:}10y$.

Effect dependency annotation: in Figure 10.9, to mitigate *network connection goes down*, we could *ensure sufficient Quality-of-Service (QoS) from network provider* with the cost of 15000USD:10y. This, however, reduces the effect of a redundant connection. These two treatments are countervailing. Ensuring such QoS will reduce the reduction effect of a redundant connection by 0.3 as annotated in the figure.

As summary, Figure 10.10 shows the treatment diagram resulting from annotating the risk diagram in Figure 10.8. Note that the likelihood annotations in Figure 10.10 are after the application of the analysis of Step 2, which is explained next.

10.3.3 Applying Step 2 – Treatment Analysis

The analysis employs an instantiated version of the calculus for risk graphs. Here, we exemplify the propagation of likelihoods and reductions through an example taken from the annotated treatment diagram of the eHealth scenario. Particularly, we show how to do the propagation for risk *"Loss of Monitored Data"* (LMD). The result is presented in Figure 10.10. For clarity, we use following acronyms for text in the diagram:

- TDI*: "Transmission of monitored Data is Interrupted"*
- NCD*: "Network Connection goes Down"*
- HGD*: "Handheld Goes Down"*
- NF*: "Network Failure"*
- HHW*: "Handheld HW failure"*
- IRN*: "Implement Redundant Network connection"*
- EQS*: "Ensure sufficient QoS from network provider"*
- IRH*: "Implement Redundant Handheld"*

Here we describe the frequency propagated for LMD. First, NF initiates NCD with frequency $30{:}10y$. The treatment IRN would reduce this frequency by $0.7L$. However, due to the effect dependency of EQS to IRN, the likelihood reduction of IRN is changed to $0.7L \cdot (1 - 0.3L) \approx 0.5L$. Hence, IRN reduces the frequency propagated to NCD to $30{:}10y \cdot (1 - 0.5) = 15{:}10y$. EQS would reduce the frequency of NCD by $0.7L$. So, the frequency propagated to NCD is $15{:}10y \cdot (1 - 0.7) = 4.5{:}10y$. Second, HHW initiates HGD with frequency $10{:}10y$. This is propagated to HGD. IRH treats HGD with likelihood reduction $0.7L$. Hence, frequency propagated to HGD is $10{:}10y \cdot (1 - 0.7) = 3{:}10y$. Since NCD and HGD are independent and both of them *lead-to* TDI, the frequency propagated to TDI is $4.5{:}10y \cdot 0.8 + 3{:}10y \cdot 0.9 = 6.3{:}10y$. Finally, the propagated frequency of LMD is $6.3{:}10y \cdot 0.8 = 5.04{:}10y$.

Likewise we can calculate the frequencies of the entire diagram. Figure 10.10 shows the complete diagram with frequencies calculated and annotated. Note that in Figure 10.10 we have calculated the likelihoods when all treatments are taken into account. However, due

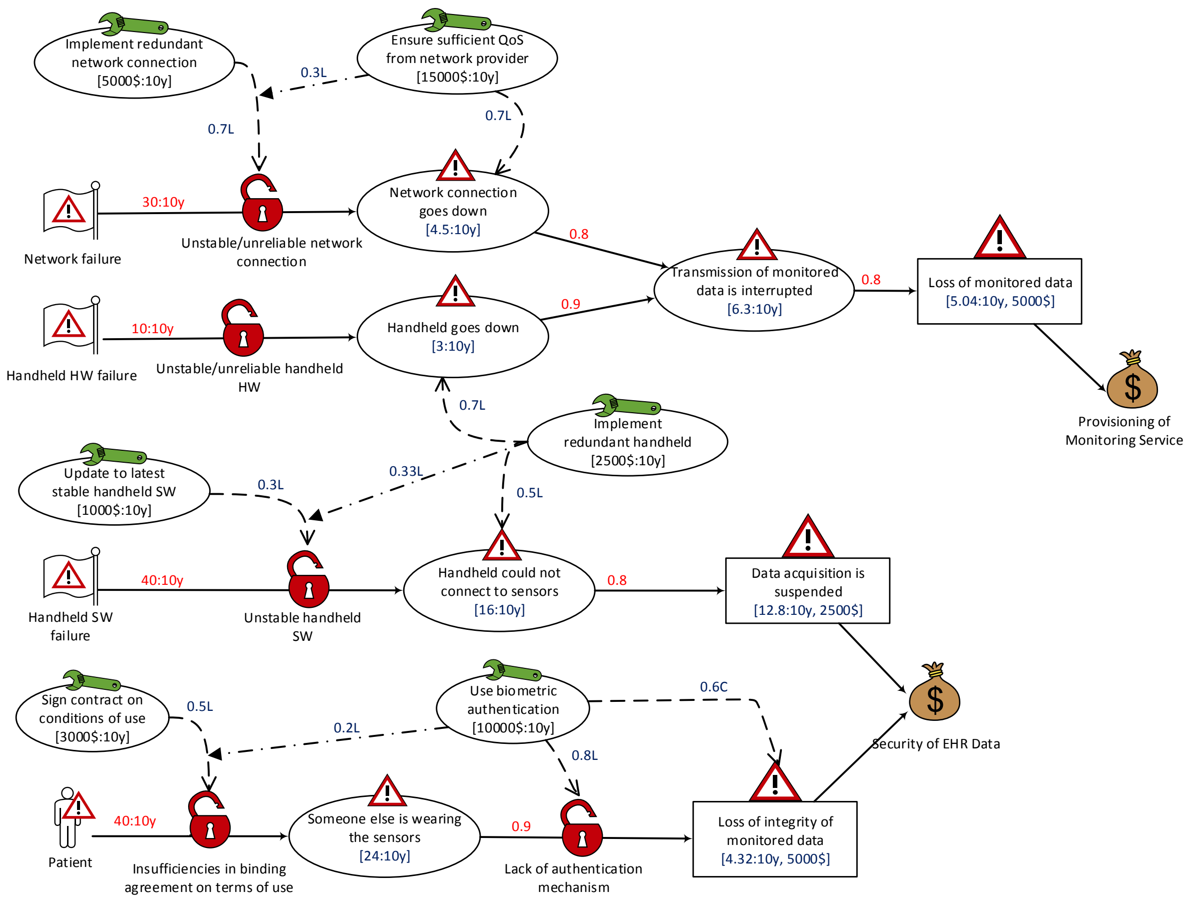

Implement redundant network connection [5000$:10y]
Ensure sufficient QoS from network provider [15000$:10y]
0.3L
0.7L
0.7L
Network failure
30:10y
Unstable/unreliable network connection
Network connection goes down [4.5:10y]
0.8
Transmission of monitored data is interrupted [6.3:10y]
0.8
Loss of monitored data [5.04:10y, 5000$]
Provisioning of Monitoring Service
Handheld HW failure
10:10y
Unstable/unreliable handheld HW
Handheld goes down [3:10y]
0.9
0.7L
Implement redundant handheld [2500$:10y]
Update to latest stable handheld SW [1000$:10y]
0.3L
0.33L
0.5L
Handheld SW failure
40:10y
Unstable handheld SW
Handheld could not connect to sensors [16:10y]
0.8
Data acquisition is suspended [12.8:10y, 2500$]
Security of EHR Data
Sign contract on conditions of use [3000$:10y]
0.5L
0.2L
Use biometric authentication [10000$:10y]
0.6C
0.8L
Patient
40:10y
Insufficiencies in binding agreement on terms of use
Someone else is wearing the sensors [24:10y]
0.9
Lack of authentication mechanism
Loss of integrity of monitored data [4.32:10y, 5000$]

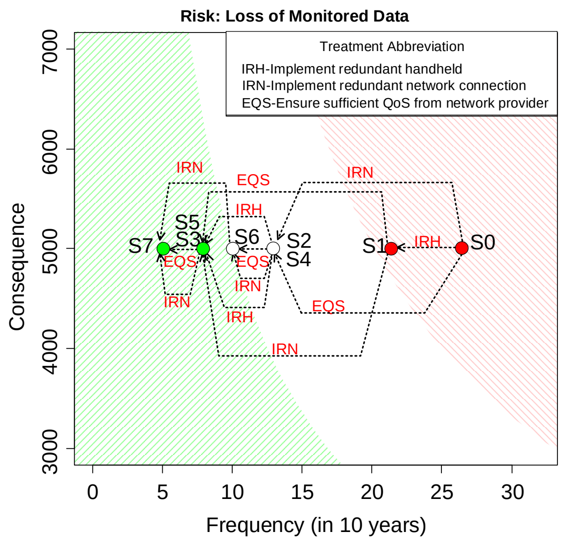
Figure 10.10: Annotated treatment diagram with frequencies propagated.

to the effect dependencies it may be that implementing all treatments is not the optimal alternative.

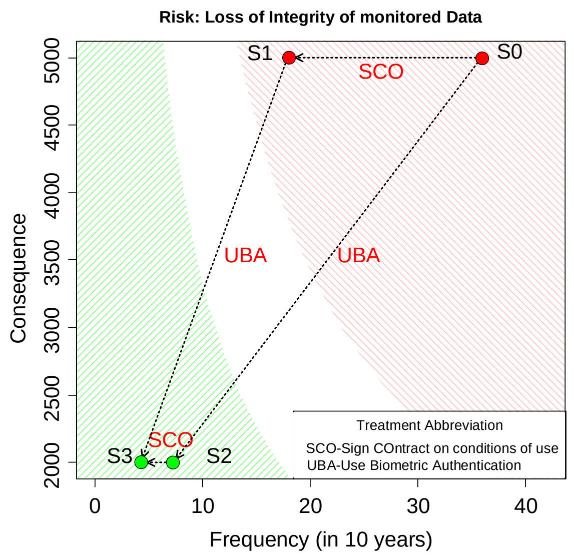
Figure 10.11 presents decision diagrams of risk LMD and LID (*i.e., Loss of Integrity of monitored Data*). The detail result of the countermeasureanalysis for risk LMD is provided in Table 10.1.

10.3.4 Applying Step 3 – Synergy Analysis

To facilitate the synergy analysis described in Step 3, we define the rc() function in (10.1) as follows: $\mathrm{rc}(r) = co \cdot f$, where co is the consequence and f is the frequency of the risk r. Having decision diagrams for individual risks, the synergy analysis described in Step 3 is detailed as below.



(a)



(b)

Figure 10.11: Decision diagrams of risks in the eHealth scenario.

STEP 3A We identify the set of global treatment alternatives based on the decision diagrams generated in the previous step and the expenditures of treatments. For each risk, we select the alleged cost-effective treatment alternatives. In particular, we choose $S3_{LMD}\{IRH, IRN\}$ (*i.e.*, state $S3$ of risk LMD), and $S7_{LMD}\{IRH, IRN, EQS\}$; for risk DAS, we choose $S3_{DAS}\{USW, IRH\}$, and $S2_{DAS}\{IRH\}$; for risk LID, we choose $S3_{LID}\{IRH, IRN\}$ and $S2_{LID}\{UBA\}$. We assume all of these alternatives are acceptable with respect to the acceptance criteria.

STEP 3B We calculate the overall cost for each global treatment alternative using (10.1). Table 10.2 reports the overall costs for these alternatives. According to this table, we select GS1 due to its smallest overall cost.

STEP 3C For the sake of simplicity, we assume that customers are satisfied with the recommendation. Therefore, GS1 will be chosen for implementation.

10.4 Modeling Evolution in Risk Graph

This section applies the risk-evolution approach described in Chapter 9 to the proposed countermeasure assessment method in this chapter to enable the modeling and reasoning about evolution in the proposed method.

Table 10.1: Analysis for the risk LMD.

The name of each treatment alternative is shown in the first column (Risk State). The *Frequency* column is number of occurrences in ten years. Both *Frequency* and *Consequence* columns are values after considering the treatments.

Ensure sufficient QoS from network provider
Implement Redundant Network connection
Implement Redundant Handheld

Risk/Risk State		Treatment		Frequency	Consequence
Risk : Loss of Monitored Data					
S0				26.4	5000
S1	•			21.36	5000
S2		•		12.96	5000
S3	•	•		7.92	5000
S4			•	12.96	5000
S5	•		•	7.92	5000
S6		•	•	10.08	5000
S7	•	•	•	5.04	5000

10.4.1 Challenges in Modeling Evolution in Risk Graph

A naïve approach to graphically model the context evolution into risk model, *i.e.,* risk graph, is to maintain risk graphs with respects to identified contexts in separated models. This approach is very easy to achieve without any modification to existing risk method. However it raises following challenges:

- *Synchronization.* We need to maintain multiple copies of the risk model, which equal the number of identified contexts. These models could share a large common portion. Therefore, any changes made to the shared part should be manually synchronized among models. Usually, a big and complex risk model is divided into diagrams. The number of diagrams to be maintained thus is multiplied by the number of identified contexts. Consequently, keeping these diagrams synchronized across contexts tends to be an overwhelm, frustrating, and error-prone task if it is done manually.

- *Different views support.* The use of separated model per each context seems to be bet-

Table 10.2: The global treatment alternatives in synergy analysis.

	Individual Risk			
Global Treatment Alternative	LID	LMD	DAS	Overall Cost
GS1{UBA,SCO,IRH,IRN,USW}	S3	S3	S3	101740
GS2{UBA,SCO,IRH,IRN,EQS,USW}	S3	S7	S3	102340
GS3{UBA,IRH,IRN,USW}	S2	S3	S3	104500
GS4{UBA,IRH,IRN,EQS,USW}	S2	S7	S3	105100
GS5{UBA,SCO,IRH,IRN}	S3	S3	S2	108740
GS6{UBA,SCO,IRH,IRN,EQS}	S3	S7	S2	109340
GS7{UBA,IRH,IRN}	S2	S3	S2	111500
GS8{UBA,IRH,IRN,EQS}	S2	S7	S2	112100

ter for local view of the evolution in each identified context, but apparently does not support well a global view of the evolution of the entire system. For example, users might want to see what will change during the evolution, and what are the differences among different evolution possibilities.

A modelling approach for evolution in risk graph should carefully consider these two challenges to better facilitate the evolution analysis of evolving risks.

10.4.2 Modeling the Evolution

We propose an extension to the conceptual model of the proposed risk-evolution approach described in Figure 9.1. The extension is to capture the evolution in risk graphs, see Figure 10.12, which is referred to as the meta-model of the evolutionary risk graph. This meta-model is presented as a UML class diagram. In Figure 10.12, a class is represented by a rectangle. The class name is inside the rectangle, and is followed by class members (if any), which are italic text, and are separated from the class name. The name of an abstract class is also in italic. The text formation for the class name determines the type of the class: normal-case text denotes a node, lower-case text denotes a relation, and underline text depicts a decoration of either node or relation.

In this meta-model, an *Evolutionary Risk Graph* is a collection of *Element*s. The *Element* abstract class represents all evolvable elements in a risk graph, which could be either an abstract *Vertex*, or an abstract *Relation*. With reference to Section 10.1.2, there are two kinds of

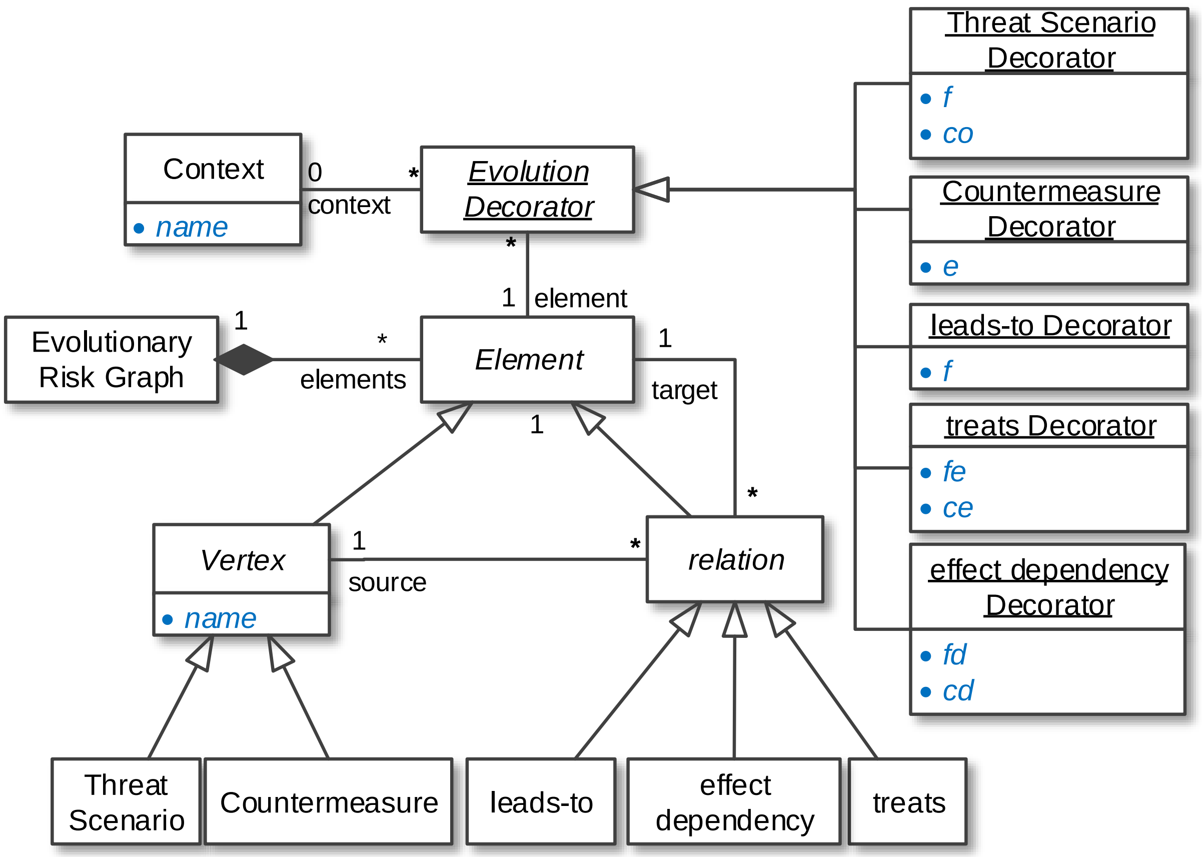
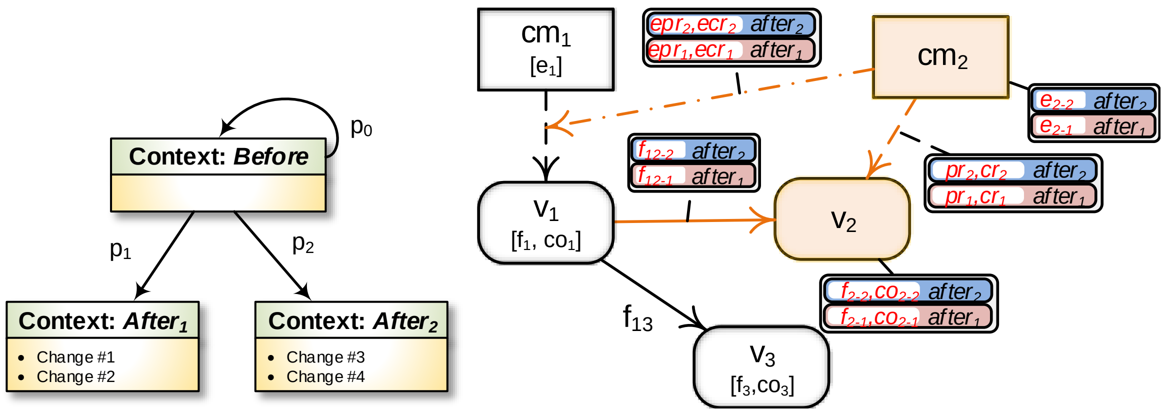


Figure 10.12: The meta-model of evolutionary risk graph.

vertices – *Threat Scenario* and *Countermeasure*, and three kinds of relations – *leads-to*, *treats*, and *effect dependency*. All evolvable properties of these kinds of *Elements* are decorated separately in *Evolution Decorator*. Each decorator keeps a link to a *Context* where the decorator is valid in. If a decorator does link to any *Contexts*, it means the *Element* associated to this decorator does not evolved within the study period. There are five types of decorators with respect to five types of *Elements*, *i.e.*, *Threat Scenario Decorator*, *Countermeasure Decorator*, *leads-to Decorator*, *treats Decorator*, and *effect dependency Decorator*. Each type of decorator has corresponding property members, for instance *leads-to Decorator* has a member f, which represents the conditional probability of a *leads-to* relation (see Section 10.1.2).

Example 10.1 Figure 10.13 illustrates an example of modeling evolution in a risk graph. On the left hand side, Figure 10.13(a) presents the context reference model (see section Section 9.2.3), where the current context *Before* could evolves to *After*$_1$, or *After*$_2$ with the levels



(a) Context evolution model (b) Risk graph with evolution decorator

Figure 10.13: An example of modeling evolution in risk graph.

of belief are p_1, p_2, respectively. Additionally, *Before* might be unchange with the belief of p_0.

On the right hand side, Figure 10.13(b) shows the risk graphs with evolution decorators annotated to modeling the evolution with respect to the context evolution model presented on the left. In Figure 10.13(b), two threat scenarios v_1, v_2, and the countermeasure cm_1 appear in all contexts, and stay unchange regardless of evolution. Thus, their corresponding properties are displayed inside their node shape. Meanwhile, the threat scenario v_2 and the countermeasure cm_2 appear in both *After*$_1$ and *After*$_2$ contexts, but not in the *Before* context. The properties of v_2 and cm_2 are therefore displayed separately where different values might be applied for different contexts. Consequently, the *leads-to* relation from v_1 to v_2, the *treats* relation from cm_2 to v_2, and the *effect dependency* relation from cm_2 are decorated accordingly. We additionally change the color of evolved elements to emphasize the evolution. ■

By modeling this way, evolution is incorporated into a single risk graph. Therefore, it avoids the *synchronization* challenge. However it might increase the complexity of the original risk graph. The *different view support* challenge could be addressed with the help of the graphical modeling tool, which will show or hide elements according to the expected view. For example, if users want to see the original risk graph, all evolved elements of which their decorators do not associate to the *Before* context will be hidden. Similar filter could be apply if users only want to see the risk graph in the *After*$_1$ context.

10.5 Chapter Summary

We have presented a risk-graph-based countermeasure assessment method to select a cost-effective countermeasure alternative to mitigate risks. The method required input in the form of risk models represented as risk graphs. The method analysed risk countermeasures with respect to different properties such as the amount of risk mitigation (Reduction Effect), how countermeasures affect others (Effect Dependency), and how much countermeasures cost (Countermeasure Expenditure). We have developed a set of formal rules extending the existing calculus for risk graphs. These new rules propagated the likelihoods and consequences along risk graphs thereby facilitating a quantitative countermeasure analysis on individual risks, and a synergy analysis on all the risks. The outcome was a list of countermeasure alternatives quantitatively ranked. These alternatives were represented not only in tabular format, but also in graphical style (Decision Diagram).

We have exemplified the method to CORAS by an example of the eHealth domain to select cost-effective treatments. Notations and rules have been adapted to comply with CORAS. The example demonstrated that the method could work with existing defensive risk analysis methods whose risk models could be converted to risk graphs.

We have enriched the method with the capacity of modeling on evolution in risk graph. This is done by applying the risk-evolution approach in Chapter 9 to the method.

In the next part, we are going to evaluate how the proposed artifacts could meet the desired success criteria and summarize the entire dissertation.

Part IV

Discussion and Conclusion

CHAPTER 11

DISCUSSION

This chapter discusses how the proposed artifacts and evaluation activities presented in the previous chapters could fit the success criteria. The chapter also discusses the relation of the proposed framework to the literature.

THIS chapter discusses how the proposed framework could fulfill success criteria described in Section 2.1 (Section 11.1), and we presents a set of steps to apply the proposed framework within the development of software system (Section 11.2). Additionally we discuss how the framework relates to the literature (Section 11.3).

11.1 Fulfillment of the Success Criteria

Table 11.1 summarizes the fulfillment of success criteria by evaluation activities described in Section 2.2. In this table, “done" and “partly" mean that the evaluation activity has fully or partly met the success criteria, respectively; “todo" means that the evaluation activity will be considered as future work. The detailed discussion is provided in the subsequent subsections.

Table 11.1: Summary of the success criteria fulfillment.

	Modeling			**Reasoning**		
Evaluation Activity	SC1.1	SC1.2	SC1.3	SC2.1	SC2.2	SC2.3
(E1) Provide model and case study	**done**		*todo*			
(E2) Conduct empirical study	**done**		*todo*	*todo*		
(E3) Provide formal semantics		**done**		**done**	**done**	
(E5) Perform formal analysis					**done**	**done**
(E4) Implement CASE tool	**done**[(1)]		partly[(2)]		**done**[(3)]	**done**[(3)]
(E6) Run simulation					partly	
(E1) Adapt the framework to evolving risks			**done**		*todo*	

SC1.1 *Able to effectively capture the requirements evolution and its uncertainty.*

SC1.2 *Accompanied with a formal semantics of the evolution uncertainty.*

SC1.3 *Potentially applicable to a variety of requirements and system models.*

SC2.1 *To provide a set of metrics with formal semantics for reasoning about evolution uncertainty.*

SC2.2 *Able to automate (with formal analysis and tool-support) the reasoning that can enumerate and quantitatively assess individual design alternatives.*

SC2.3 *Able to support the incremental modeling of evolution.*

(1): provide GUI for modeling and reasoning.

(2): provide GUI templates for modeling.

(3): provide implementation for reasoning.

11.1.1 The Modeling Approach of the Framework

SC1.1 *Able to effectively capture the requirements evolution and its uncertainty.* We have fulfilled this success criteria by carrying out two evaluation activities on the evolution rules, which are used to model requirements evolution and its uncertainty.

- We have run a self-evaluation for these rules on a "big" requirements model taken from an industrial project (see Section 7.5). This is an instance of Self-evaluation study (E1).

- We conducted a series of empirical studies on participants with various levels of expertise knowledge. We have analyzed artifacts produced by participants. The analysis outcomes have revealed that the proposed framework is effective in modeling requirements evolutions. Moreover, the prior domain knowledge and framework knowledge do not significantly impact the effectiveness of the framework. The details of the study settings as well as the analysis of artifacts are presented in Chapter 8 . This is the materialization of Empirical study (E2).

SC1.2 *Accompanied with a formal semantics of the evolution uncertainty*. We have fulfilled this success criteria by proposing an interpretation for the evolution uncertainty. The interpretations is based on game theory. We explain the evolution uncertainty by the game between three different players: Domain Expert, Company, and Expert. The semantics of evolution uncertainty is accounted by the belief of Domain Expert on the monetary value, which he/she is willing to pay for the implementation that addresses a particular forecasted changes. This interpretation is detailed in Section 5.2.

SC1.3 *Potentially applicable to a variety of requirements and system models*. We have partially fulfilled this success criteria as follows:

- We considered requirements models by which we apply evolution rules to capture evolution at a very high level of abstraction. A requirements model is treated as a collection of entities and relations. This level of abstraction could present requirements models in any graphical RE languages. The proposed evolution rules could capture evolution in such level of abstraction of requirements models. Thus they could be able to do so for any requirements models in any graphical RE languages. This is detailed in Section 5.1.

- We have adapted the proposed framework to another field rather than RE – the risk assessment realm. The adapted variant of the proposed framework could be able to early dealing evolving risks in software systems. The adapted framework is built upon any existing risk assessment methods, which could produce outputs that are compliant with the adapted framework. The details of this adapted framework are presented in Chapter 9 . This is an instance of Self-evaluation study (E1).

- We have proposed a generic risk assessment method to support the selection of cost-effective risk countermeasure alternatives. The proposed risk assessment method is built upon risk graph, thus it can represent for any risk assessment methods, which are compliant with risk graph. We then apply the adapted framework to the proposed risk assessment method. Thus it is an evidence that adapted framework for risk evolution is also able to apply to many risk assessment methods. The detail of the risk assessment method and the application of the adapted framework to this method are discussed in Chapter 10 . Moreover, we have also adapted the proposed framework to the field of software product line engineering, in particular the evolution of feature model. However we do not present that work here, but in a separated technical report [TM13a].

11.1.2 The Reasoning Approach of the Framework

SC2.1 ***To provide a set of metrics with formal semantics for reasoning about evolution uncertainty.*** We have fulfilled this success criteria by proposing three evolution metrics, namely Max Belief, Residual Disbelief, and Max Disbelief. The formal definition of these metrics as well as their semantics are presented in Section 5.3.

SC2.2 ***Able to automate (with formal analysis and tool-support) the reasoning that can enumerate and quantitatively assess individual design alternatives.*** We have fulfilled this success criteria by proposing a series of algorithms to perform the reasoning about evolution uncertainty. We employ hypergraph as an intermediate data structure to represent requirements models. We do *not* intend to substitute any RE languages by this hypergraph, but transform requirements models in a particular RE language to hypergraph. This transformation only preserves information about the fulfillment of top requirements. Based on hypergraph we develop algorithms to identify design alternatives and compute evolution metrics for these alternatives.

We have performed a formal analysis on the soundness and complexity of the algorithms. The details of algorithms and their formal analysis are described in Chapter 6 . Moreover, we have implemented a proof-of-concept prototype of a CASE tool in which we implement these algorithms. The details of this prototype are described in Chapter 7 .

We have conducted a benchmark of the algorithms on a large example (of approximately 150 goals with 5 observable rules). The example yields a number of 864 design alternatives. Such number is too large for a manual selection of alternatives. Additionally, we have run a simulation on the performance of the algorithms, see Section 7.4

SC2.3 ***Able to support the incremental modeling of evolution.*** We have fulfilled this success criteria by employing a caching mechanism for the calculation of proposed metrics. We develop algorithms that handle incremental modification made to requirements models. These algorithms will trigger different reactions corresponding to different actions of modification. These reactions will update the data for metric calculation for the only modified parts of the requirements models instead of redoing the computation from ground up. This enables an efficient computation of metrics with respect to incremental modification made to the requirements models. The details of these algorithms are discussed in Section 6.2.3.

11.2 How to Apply the Proposed Framework

In this section, we propose potential steps to apply the proposed framework. Basically these steps are hooked into the software development process, particularly, where the requirements model has just been produced. Each time an alternative decision or a potential evolution is identified, they could be captured by using the techniques of the underlying method. The detail of the steps are as follows:

INPUT. The input is a requirements model (expressing in a particular RE language), and all possible changes to the model as well as their likelihoods of occurrences. Since there could be many parallel and competing changes, a revision of the input requirements model may have many variants.

STEP 1. In this step, changes are examined to revise the input requirements model to produce the *future* revision of requirements model for the system-to-be-next. The input requirements model and variants of its future revisions together with the likelihoods of occurrences changes are formulated into *observable rules*.

STEP 2. In this step, different design alternatives of both input requirements model and its future variants are derived to formulate *controllable rules*. These design alternatives could be automatically derived in some RE languages (*e.g.*, i*, KAOS).

STEP 3. In this step, the proposed metrics are calculated for each of design alternatives. The design alternatives and their metrics values are used to support the decision making process. The following cases might happen:

> **STEP 3A.** If decision maker could choose a design alternative, move forward to next phase of the development process.
>
> **STEP 3B.** If decision maker could not choose any design alternatives, go to either STEP 1 to elaborate more design alternatives, or STEP 2 to revise future revisions of the input requirements model.

OUTPUT. The output includes all artifacts generated in these steps, which are an evolutionary requirements model of the system (*i.e.*, the input requirements model plus variants of its future revision and evolution rules), and a 'good' design alternative by decision maker.

Whenever a new change is identified, we might need to start over from STEP 1 to STEP 3 to select a new design alternative.

The framework will involve following actors: designer, domain expert, and decision maker. The roles of these actors within the proposed framework are identified as follows:

- The domain expert could be a trained consultant who possesses deep knowledge about the domain. S/he will forecast potential changes in future to the system, as well as his/her belief about the likelihoods of occurrences of these changes.

- The designer evaluates the foreseen changes from domain expert to formulate possible evolution possibilities of the system. For each possibility, the designer may revise the original requirements model (*i.e.,* the model built upon the current requirements and the current settings of the working environment) so that the new revision could cope with the expected new changes.

  The designer is also responsible for performing the reasoning on evolution uncertainty to aid decision maker.

- The decision maker takes different design alternatives and the reasoning output from the designer to choose a design alternative for the next step of the system-to-be-next.

11.3 How The Proposed Framework Relates to State-of-the-Art

11.3.1 Requirements Evolution

Among four types of change categorized by Lam and Loomes [LL98], we capture the requirement change and design change as observable evolution and controllable evolution. We do not capture the environment change and viewpoint change since this work mostly focuses on the evolution of software artifact (i.e., requirements model) rather than environment and viewpoint changes. While they have a meta and a process model but did not go beyond that than, we have a conceptual model supported by an automated reasoning. Among seven key issued, the proposed framework satisfies the *'modeling evolution'*, *'multiple change'*, and *'tool support'*; and the application of the proposed framework on risk domain partially addresses the *'risk assessment'* issue. We do not address the *change analysis*' not *extended traceability* as we do not focus on the nature of change and we only work on the requirements model artifact. The *impact assessment approaches* on other social domains would be an interesting direction for future work.

In comparison to studies on impacts of evolution [Rus+99; Has+05; Che+09; Lin+09], the proposed framework is different in the perspective that we assess the impact of potential changes in advance, rather than after changes happen. However, it is not limited that one can use the approaches in [Has+05; Che+09; Lin+09] to link the modeled requirements to changes in specification since these work complement to the proposed framework in this dissertation.

Fabbrini *et al.* [Fab+07] work on plain text requirements and use the Formal Analysis Context to detect the requirements inconsistency between different the evolution steps. Instead, we work on requirements models expressed by an existing RE language. We also have different evolution steps as they have; however, we focus on the modeling the requirements evolution uncertainty and reason about them to aid the selection of evolution-resilient design alternative but not checking the inconsistency.

The SECMER methodology [Pro12; Ber+11] detects the violation of security properties. Security properties in Si* requirements model are ensured on-the-fly as the modifications are being made. We also address the 'before-after' evolution perspective. The methodology however does not capture the evolution uncertainty while we do. We do not focus on ensuring security properties but selecting evolution-resilient design alternatives in advance.

The 'before-after' evolution perspective that we address is similar to the idea of situation 'before-the-change' and situation 'after-the-change' mentioned by Brier *et al.* [Bri+06]. However, their approach only captured the part of before model that is changes, but did not have any further specific reasoning about that. In the proposed framework, we do not only have modeling notions to capture evolution, but also a conceptual model and a reasoning backed up by automated algorithms.

The approach of Zowghi and Offen [ZO97] involves modeling requirements models as theories and reasoning changes by mapping changes between models. However, this approach has a significant overhead for the encoding of requirement models into logic. In contrast, the proposed framework can operate transparently on requirements models.

Ernst *et al.* [Ern+11] focus on unknown-unknown evolution, *i.e.,* evolution that we do not know what it is, and when it happens. Meanwhile, we focus effort on known-unknown evolution, *i.e.,* evolution that we know what it is, but do not know when it happens.

11.3.2 Empirical Studies

Runeson and Host [RH09] propose a broader set of guidelines about how to conduct qualitative research. We have based the evaluation empirical studies on those guidelines. Following those guidelines, we have involved researchers, students, practitioners (*i.e.,* domain experts) with different expertise and used semi-structured interviews, questionnaires and audio-video recordings to collect data in order to perform data triangulation. Kitchenham [Kit96] identify *screening, effects analysis,* and *benchmarking* as classes of research methods. We have used the *screening* method to evaluate the correctness of requirements produced by students (in Section 8.2.3.6). We also employ the *effects analysis* method to process and analyze the collected data.

Besides, there exist works reporting empirical studies on requirements evolution and the one that use the ATM domain as evaluation context (see Section 3.3.2). These studies are closely related to the empirical studies, which are described in Chapter 8.

A closely related work to the empirical evaluation studies (Chapter 8) is from Villela *et al.* [Vil+10] who reported on a quasi-experiment to assess the adequacy and feasibility of PLEvo-Scoping method [Vil+08]. The quasi-experiment took place in the form of two two-days workshops with the participation of domain experts. They used both quantitative and qualitative measures but did not have any statistical tests. Similarly, in the evaluation studies we have adopted a workshop for evaluating with ATM domain experts the effectiveness of the proposed framework to model requirements evolution and we have used both quantitative and qualitative measure to assess its effectiveness. However, in the evaluation studies described in Chapter 8 we were able to apply statistical hypothesis testing research approach to check the statistical validity of the hypotheses related to the research questions.

Compare to the case study by McGee and Greer [MG11], we also followed the guideline in [RH09] used hypothesis testing: we refined the research questions into a set of hypotheses and identified quantitative metrics like the size of change and the number of evolution rules to evaluate the effectiveness of the approach to model requirements evolution. In addition, before testing the hypotheses, we asked an expert in the ATM domain to evaluate the quality of the requirements models and of the requirements changes identified by the participants of the studies (see Section 8.3).

In the quantitative analysis (see Section 8.3), we employ the idea of delta requirements (by Herrmann *et al.* [Her+]) to study the complexity evolution rules where we focus on how big the change is in each evolution rule identified. We use the notion of delta requirements (*i.e.,* requirements evolution) to define the quantitative metric *size of requirements change* associated with the evolution rules identified by the participants of the studies. We used this metric to evaluate the effectiveness of the approach in modeling requirements evolution.

As claimed by Maiden et al. [Ncu+07; MR05; Mai+04], the workshop sessions can help to set up a common understanding and facilitated the interaction among people involved. We therefore adopt workshops with a similar structure to conduct the study with ATM experts (see Section 8.2.3.2). With respect to the creativity workshops by Maiden et al., the workshops in Chapter 8 consists of a *training* and *application* phase. We did not have a *brainstorming* phase that has been replaced with an *evaluation* phase where the quality of researchers' models was evaluated by the ATM domain experts. In addition, in the evaluation studies we identified quantitative metrics and used statistical hypothesis testing to assess the effectiveness of the proposed approach in modeling requirements evolution, while in the studies by Maiden et al., qualitative data were used to evaluate the RESCUE method.

A survey of the literature on empirical studies related to requirements change shows that there is little research that uses an empirical basis to understand the impact of requirements evolution, why and when it happens and to assess the effectiveness of methods to manage requirements change. As far as we are aware of, this study is one of the few studies that has used quantitative metrics and hypothesis testing to assess possible requirements changes and the effectiveness of the proposed framework to model such changes.

11.3.3 Selecting Countermeasure Alternatives for Evolving Risks

We are aware of few studies about evolving risks by Lund *et al.* [Lun+11b] and Solhaug and Seehusen [SS13]. They proposed general techniques and guidelines for managing risk in changing systems. However, they did not mention the uncertainty of evolutions (*i.e.,* likelihood of occurrence), and how to use this information to support the decision making process.

Applying the proposed framework to evolving risk in Chapter 9 is a pioneer effort to fill that gap. The basic difference between [Lun+11b; SS13] and the work reported in Chapter 9 is that they aimed to assess the changing risk on evolving systems, we aim to evaluate how well a countermeasure alternative could resist evolution during a period of time. These are two complementary aspects of evolving risks. Furthermore, their work only discussed the support in the *before-after* evolution perspective, meanwhile, we target both the *before-after* and *continuous* evolution perspectives. We make use of the potential changes in the current context and their uncertainty, together with the output of the risk assessment to support decision makers to select the most appropriate countermeasure alternative to implement.

We have metrics (Max Belief, Residual Disbelief, Max Disbelief) to evaluate the evolution-resilience of the risk countermeasure alternatives, which are the outputs of assessment.

Furthermore, the proposed method in Chapter 10 provides more information for the decision makers to select cost-effective countermeasure alternatives for a target software system within a particular context at a particular point of time. We have analyses (treatment analysis, synergy analysis) and decision diagrams to provide more insights about cost-effectiveness of countermeasure alternatives. The proposed method may be seen as a special case or refinement of the process suggested by Mehr and Forbes [MF73].

There exist guideline and methods [Sto+02; WHO09; But02] to evaluate countermeasure alternatives in terms of cost. However, these methods have not been designed to assess cost of countermeasures but rather cost (or consequence) of risks. The decision methodology of Chapman and Leng [CL04] includes analyses, methods, and framework for economic evaluation of countermeasure alternatives. However, it does not take into account the level of risk reduced by different alternatives. We are different from these work [Sto+02; WHO09; But02; CL04] since we take all information such as consequence of risks, cost of countermeasures, and level of residual risk to assess cost-effective countermeasure alternatives. The work of Norman [Nor10] also covers these information; however the artifacts are developed as spreadsheets, which are complicated to implement and follow. Meanwhile, the proposed method is a well-defined process and backed up with a reasoning; and the introduced artifacts are graphical. As the alternatives are more focused on the technical aspect, the output of the proposed method could be taken as the input for Real Options Thinking based assessment [Kul+99; AK99; LS07].

The SecInvest framework by Houmb *et al.* [Hou+12] is closely related to the method described in Chapter 10 because it also employs existing risk assessment techniques for risk level. It scores countermeasure alternatives with respect to their cost and effect, trade-off parameters, and investment opportunities. While we provide a systematic way to assess the effects of alternatives on risks, also take into account the dependency among countermeasures in an alternative, they however do not. The proposed method is sufficiently generic to be integrated within many existing risk analysis methods. We have demonstrated this by instantiating in the CORAS method for security risk analysis [Lun+11a] with concrete illustrative examples.

CHAPTER 12

CONCLUSION

This chapter summarizes the proposed artifacts and evaluation activities conducted in the dissertation. Based on this, the chapter discusses potential future research directions based on the outcomes of the dissertation.

WE have presented a framework to deal with evolution in requirements models. We have also showed that the proposed framework is not only applicable in the domain of requirements evolution, but also helpful in the realm of risk management. This chapter summarizes the major contributions of this dissertation (Section 12.1) and discusses potential future research based on the results (Section 12.2).

12.1 Summary

A key contribution in this dissertation is the framework for modeling and reasoning about evolution in requirements model in long-lived software systems. The aim of this framework is to provide a means for studying requirements evolution. The framework includes evolution rules to capture evolution in requirements model, particularly observable rules for capturing potential changes and their uncertainty, and controllable rules for capturing different reactions from the designing aspect (*i.e.*, design choices or design alternatives) to these changes. By incorporating evolution in requirements models, we allow designers to have a global view about the potential evolution of the system in future.

The evolution uncertainty is represented as evolution probability, which is the belief about the likelihood of occurrence of evolution. The evolution probability is associated with a semantic described by a game-theoretic approach. This game semantics provides a better understanding of evolution probabilities.

Based on evolution probability, we propose a set of metrics enabling quantitative reasoning on design alternatives to understand to what extent they can resist to evolution. In other words, we measure the belief in the game between domain expert, company, and the market that the implementation of a design alternative could still be operational when evolution occurs.

Moreover, the proposed reasoning is supported by a series of algorithms that automate the calculation of metrics, and a proof-of-concept CASE tool. The algorithms not only calculate metric values for a particular design alternative, but also enumerate possible design alternatives with the best metric values, *i.e.,* winner alternatives. Additionally the algorithms incrementally react to every single change made to requirements models. Thus, they provide an instant view of a list of winner alternatives. This facilitates the decision making process. We also provide the proofs for the soundness and complexity of reasoning algorithms.

The proposed framework has been evaluated with a series of empirical studies that took place over a year to evaluate the modeling part of the framework. The evaluation studies relied scenarios taken from industrial projects in the ATM domain. The studies involve different types of participants with different expertise in the framework and the domain. The results from the studies showed that the modeling approach is effective in capturing evolution of complex systems. It is reasonably possible for people, if they are supplied with appropriate knowledge (*i.e.,* knowledge of method for domain experts, knowledge of domain for method experts, and knowledge of both domain and method for novices), to build significantly large models, and identify possible ways for these models to evolve. Moreover, the studies have shown that obviously there is a difference between domain experts, method experts, and students on the "baseline" (initial) model, but when it comes to model the changes with evolution rules, there is no significant difference.

We have adapted the proposed framework to the realm of risk assessment. We have proposed an approach to deal with evolving risks. The proposed approach operates at high level of abstraction. It could work with many existing risk-assessment methods. To address the representation of the evolution rules in complex risk models, we further apply the proposed risk-evolution approach to a risk assessment method. For this purpose, we have introduced a risk-graph-based countermeasure assessment method, then apply the risk-evolution approach to this method. The evolution rules are represented in risk graphs by tags decorated directly to elements of risk graphs.

In short, the proposed framework could be able to help to improve the evolution resilience of long-lived framework systems in the early phase of software development. Therefore the approach has potentially contributed to improve the sustainability of long-lived software systems.

12.2 Limitations and Future Work

The proposed framework takes anticipated evolution and evolution uncertainty as inputs to enrich the requirements model of a system-to-be. It is important that how evolution and its uncertainty could be elicited. The evolution uncertainty is a kind of subjective probability. The subjective probabilities are also exploited in other fields such as risk management. The more precise the beliefs, the more precise the reasoning. Thus it is necessary to have a well-defined process or guidance to achieve and evaluate these inputs. Lacking on this kind of process is a limitation of the dissertation. Hence, a promising approach to evaluate evolution uncertainty is based on the interpretation about uncertainty and the Analytic Hierarchical Process (AHP), which is used in the literature to prioritize requirements.

We have evaluated the effectiveness of the proposed framework in the ATM domain with the participation of researchers, ATM experts, and students. These studies are limited in a single domain and a particular RE language. Further empirical studies to continue evaluate the framework would be more interesting. In particular, future research could focus on the following issues:

- Replicate the empirical studies in another domain rather than ATM, and with different kinds of participants to see whether similar results could be obtained.
- Replicate the empirical studies with different RE languages rather than i*/Tropos to see whether there is any impact to the chosen RE languages to the outcome of the studies.
- Evaluate different aspects of the framework rather that effectiveness, for example, whether the method is easy to use (*i.e.,* Perceived Ease Of Use), whether participants want to apply the method in practice (*i.e.,* Intent of Use), so on and so forth. These aspects could be obtained by performing interview (with and/or without predefined questionnaire) with individual participants.

During the evaluation studies with ATM experts, we have suggested by users that a simpler graphical representation of evolution rules would help in the cases of big and complex requirements models. This suggestion should be addressed while applying the proposed framework to a particular RE language. The graphical representation of evolution rules apparently depends on the RE language the framework applies to. For instance, in goal-based languages the observable rules could be represented via a new construct (*e.g.,* observable node, see Chapter 7), the controllable rules are represented by the OR-decomposition; in risk graph, the evolution rules are represented by additional tags decorated to elements of risk graph. Therefore, it would be interesting to develop a more generic representation of evolution rules that could be applied to different RE languages.

The proposed framework has presented artifacts to capture evolutions in requirements models, and to facilitate the identification and assessment of design alternative. However, the process to apply the proposed framework in the software development is still very preliminary. More effort to extend the framework in this direction would be interesting.

The evolution metrics proposed by the framework is a single value for the entire period where evolution is examined. To address this limitation, it might be more interesting to measure evolution metrics as a time series especially for the continuous evolution perspective. By studying the evolution of evolution metrics we could provide extra information to support the selection of design alternatives.

Moreover, the proposed evolution metrics measure the resilience of design alternatives with respect to evolution. If the working environment of the system-to-be is too dynamic, we might need additional metrics. For example, further research could focus on the analysis that measures the cost to repair a design alternative.

We have proposed an adaptation of the proposed framework to address the evolution of risks in long-lived systems. It would be interesting to conduct empirical evaluation with industry. The settings of the empirical studies described in Chapter 8 could be used at starting points to conduct future empirical research.

We could also exploit the applicability of the proposed framework in the field of software product line engineering. We have reported such application in a separated report [TM13a].

Another interesting direction based on this framework is to relax its input assumptions. Currently, the framework requires all possible environment changes plus corresponding changes in requirements models anticipated. However, in practice, we might have information about potential changes, but not specific changes in requirements models. Thus, we need to extend and/or generalize the framework to make it able to deal with such situations.

References

[Adm09] Federal Aviation Administration. *System Wide Information Management (SWIM). Segment 2 Technical Overview*. Tech. rep. 2009.

[AK99] Martha Amram and Nalin Kulatilka. *Real Options: Managing Strategic Investment in an Uncertain World*. Havard Business School Press, Cambridge, Massachusetts, 1999.

[Ali10] Raian Ali. "Modeling and Reasoning about Contextual Requirements: Goal-based Framework". PhD thesis. University of Trento, 2010.

[And04] Chris Anderson. "The Long Tail". In: *Wired* (2004).

[AP01] M. D. Adler and E. A. Posner. *Cost-benefit analysis. Legal, economic and philosophical perspectives*. University of Chicago Press, Chicago, 2001.

[AP03] Annie I. Antón and Colin Potts. "Functional Paleontology: The Evolution of User-Visible System Services". In: *IEEE Transactions on Software Engineering* 29(2) (2003), pp. 151–166.

[Aus+83] G. Ausiello, A. D'Atri, and D. Saccà. "Graph algorithms for functional dependency manipulation". In: *Journal of the ACM* 30 (1983), pp. 752–766.

[Bas+86] V R Basili, R W Selby, and D H Hutchens. "Experimentation in software engineering". In: *IEEE Transactions on Software Engineering* 12.7 (July 1986), pp. 733–743.

[Ber+11] Gábor Bergmann, Fabio Massacci, Federica Paci, Thein Than Tun, Dániel Varró, and Yijun Yu. "SeCMER: A Tool to Gain Control of Security Requirements Evolution". In: *ServiceWave*. 2011, pp. 321–322.

[BM92] David G.W. Birch and Neil A. McEvoy. "Risk analysis for Information Systems". In: *Journal of Information Technology* 7 (1992), pp. 44–53.

[Boa+01] A. E. Boardman, D. H. Greenberg, A. R. Vining, and D. L. Weimer. *Cost-benefit analysis. Concepts and practice*. Ed. by Second edition. Prentice Hall, Upper Saddle River., 2001.

[BR88] V.R. Basili and H.D. Rombach. "The TAME project: towards improvement-oriented softwareenvironments". In: *IEEE Transactions on Software Engineering* 14.6 (1988), pp. 758–773.

[Bra+10] Gyrd Braendeland, Atle Refsdal, and Ketil Stølen. “Modular analysis and modelling of risk scenarios with dependencies”. In: *J. Syst. Softw.* 83.10 (Oct. 2010), pp. 1995–2013.

[Bri+06] J. Brier, L. Rapanotti, and J.G. Hall. “Problem-based analysis of organisational change: a real-world example”. In: *Proceedings of the 2006 international workshop on Advances and applications of problem frames (IWAAPF’06).* ACM, 2006.

[Bry+09] Volha Bryl, Fabiano Dalpiaz, Roberta Ferrario, Andrea Mattioli, and Adolfo Villafiorita. “Evaluating procedural alternatives: a case study in e-voting”. In: *EG* 6.2 (2009), pp. 213–231.

[But02] Shawn A. Butler. “Security Attribute Evaluation Method: a Cost-Benefit Approach”. In: *Proceedings of the International Conference on Software Engineering (ICSE’02).* Orlando, Florida: ACM, 2002, pp. 232–240.

[CF+09] Nelly Condori-Fernández, Maya Daneva, Klaas Sikkel, Roel Wieringa, Oscar Dieste, and Oscar Pastor. “Research Findings on Empirical Evaluation of Requirements Specifications Approaches”. In: *Proceedings of the 12th Workshop on Requirements Engineering (WER’09).* Valparaiso University Press, 2009, pp. 121–128.

[CH11] Isabelle Côté and Maritta Heisel. “A UML Profile and Tool Support for Evolutionary Requirements Engineering”. In: *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR’11).* 2011.

[Cha91] E. Charniak. “Bayesian networks without tears: making Bayesian networks more accessible to the probabilistically unsophisticated”. In: *AI Magazine* 12 (4) (1991), pp. 50–63.

[Che+09] Marsha Chechik, Winnie Lai, Shiva Nejati, Jordi Cabot, Zinovy Diskin, Steve Easterbrook, Mehrdad Sabetzadeh, and Rick Salay. “Relationship-based change propagation: A case study”. In: *Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering (MISE’09).* IEEE Computer Society, 2009, pp. 7–12.

[CL04] Robert E. Chapman and Chi J. Leng. *Cost-Effective Responses to Terrorist Risks in Constructed Facilities.* Tech. rep. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.

[Dal+04] Nikunj P. Dalal, William J. Kolarik, and Eswar Sivaraman. “Toward an Integrated Framework for Modeling Enterprise Processes”. In: *Communications of the ACM* 47.3 (2004), pp. 83–87.

[Dal11] Fabiano Dalpiaz. “Exploiting Contextual and Social Variability for Software Adaptation”. PhD thesis. University of Trento, 2011.

[Eas+07] S. Easterbrook, J. Singer, M.A. Storey, and D. Damian. *Selecting Empirical Methods for Software Engineering Research*. Ed. by F. Shull and J. Singer. Springer, 2007.

[Elb08] Anita Elberse. “Should You Invest in the Long Tail?” In: *Harvard Business Review* (2008).

[Erd+13] Gencer Erdogan, Fredrik Seehusen, Ketil Stølen, and Jan Aagedal. “Assessing the Usefulness of Testing for Validating the Correctness of Security Risk Models Based on an Industrial Case Study”. In: *Proceedings of the 2nd International Workshop on Quantitative Aspects in Security Assurance*. 2013.

[Ern+09] Neil A. Ernst, John Mylopoulos, and Yiquiao Wang. “Requirements Evolution and What (Research) to Do about It”. In: *Lecture Notes in Business Information Processing* 14 (2009), pp. 186–214.

[Ern+11] Neil A. Ernst, Alexander Borgida, and Ivan Jureta. “Finding incremental solutions for evolving requirements”. In: *Proceedings of the 19th IEEE International Requirements Engineering Conference (RE)*. 2011, pp. 15–24.

[Eur03] Eurocontrol. *ATM Strategy for the Years 2000+ vol. I and vol. II*. Eurocontrol. 2003.

[EUR03] EUROCONTROL. *EUROCONTROL, ATM Strategy for the Years 2000+*. 2003.

[Fab+07] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. “Controlling Requirements Evolution: a Formal Concept Analysis-Based Approach”. In: *Proceedings of the International Conference on Software Engineering Advances (ICSEA’07)*. 2007, p. 68.

[FB01] Robert France and Jame Bieman. “Multi-View Software Evolution: A UML-based Framework for Evolving Object-Oriented Software”. In: *Proceedings of the IEEE International Conference on Software Maintenance (ICSM’01)*. 2001, pp. 386–395.

[Fel+14] Michael Felderer, Basel Katt, Philipp Kalb, Jan Jürjens, Martín Ochoa, Federica Paci, Le Minh Sang Tran, Thein Than Tun, Koen Yskout, Riccardo Scandariato, Frank Piessens, Dries Vanoverberghe, Elizabeta Fourneret, Matthias Gander, Bjørnar Solhaug, and Ruth Breu. “Evolution of Security Engineering Artifacts: A State of the Art Survey”. In: *International Journal of Secure Software Engineering* (2014). To appear.

[Gra+09] Rober Graham, Nadine Pilon, Harmut Koelman, and Paul Ravenhill. “Performance Framework and Influence Model in ATM”. In: *Proceedings of the 28th Digital Avionics Systems Conference (DASC’09). IEEE/AIAA.* 2009, 2.A.5–1 –2.A.5–11.

[Has+05] J. Hassine, J. Rilling, J. Hewitt, and R. Dssouli. “Change impact analysis for requirement evolution using use case maps”. In: *Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE’05).* 2005, pp. 81–90.

[Hau+03] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. “Why Timed Sequence Diagrams Require Three-Event Semantics”. In: *Scenarios: Models, Transformations and Tools.* 2003, pp. 1–25.

[Her+] Andrea Herrmann, Armin Wallnöfer, and Barbara Paech. “Specifying Changes Only — A Case Study on Delta Requirements”. In: *Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ’09).* Springer-Verlag, pp. 45–58.

[Hou+12] Siv Hilde Houmb, Indrajit Ray, and Indrakshi Ray. “SecInvest : Balancing Security Needs with Financial and Business Constraints”. In: *Dependability and Computer Engineering* (2012), pp. 306–328.

[Iec] *IEC 60300-3-9 Dependability management – Part 3: Application guide – Section 9: Risk analysis of technological systems – Event Tree Analysis (ETA).* International Electrotechnical Commission. 1995.

[IEC90] IEC. *IEC 61025 Fault Tree Analysis (FTA).* International Electrotechnical Commission. 1990.

[ISO09] ISO. *ISO 31000 Risk management – Principles and guidelines.* International Organization for Standardization. 2009.

[Jür02] Jan Jürjens. “UMLsec: Extending UML for Secure Systems Development”. In: *UML.* 2002, pp. 412–425.

[Kit96] Barbara Ann Kitchenham. “Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods”. In: *ACM SIGSOFT Software Engineering Notes* 21.1 (Jan. 1996), pp. 11–14.

[KS99] C. F. Kemerer and S. Slaughter. “An empirical approach to studying software evolution”. In: *IEEE Transactions on Software Engineering* 25.4 (1999), pp. 493–509.

[Kul+99] N. Kulatilaka, P. Balasubramanian, and J. Strock. *Using Real Options to Frame the IT Investment Problem.* in: Real Options and Business Strategy Applications to Decision-Making, Risk Publications. 1999.

[Lab+13] Katsiaryna Labunets, Fabio Massacci, Federica Paci, and Le Minh Sang Tran. “An Experimental Comparison of Two Risk-Based Security Methods”. In: *Proceedings of the ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* 2013.

[LaM+08] M.J. LaMantia, Y. Cai, A. MacCormack, and J. Rusnak. “Analyzing the Evolution of Large-Scale Software Systems Using Design Structure Matrices and Design Rule Theory: Two Exploratory Cases”. In: *Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA’08).* 2008, pp. 83–92.

[Lam09a] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* John Wiley & Sons, 2009.

[Lam09b] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications.* A John Wiley and Sons Ltd. Publication, 2009.

[Leh80a] M.M. Lehman. “On understanding laws, evolution and conservation in the large program life cycle”. In: *Journal of Systems and Software* 1.3 (1980), pp. 213–221.

[Leh80b] M.M. Lehman. “Programs, life cycles, and laws of software evolution”. In: *Proceedings of IEEE* 68.9 (1980), pp. 1060–1076.

[Lin+09] Lan Lin, Stacy J. Prowell, and Jesse H. Poore. “The impact of requirements changes on specifications and state machines”. In: *Journal Software – Practice & Experience* 39.6 (2009), pp. 573–610.

[LK99] Pericles Loucopoulos and Evangelia (Vagelio) Kavakli. “Enterprise Knowledge Management and Conceptual Modelling”. In: *Proceedings of the 16th International Conference on Conceptual Modeling (ER’97).* 1999, pp. 123–143.

[LL98] W. Lam and M. Loomes. “Requirements evolution in the midst of environmental change: a managed approach”. In: *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR’98).* 1998, pp. 121–127.

[LS06] Mass Soldal Lund and Ketil Stølen. “A Fully General Operational Semantics for UML 2.0 Sequence Diagrams with Potential and Mandatory Choice”. In: *FM 2006: Formal Methods, 14th International Symposium on Formal Methods.* 2006, pp. 380–395.

[LS07] J. Li and X. Su. “Making Cost Effective Security Decision with Real Option Thinking”. In: *Proceedings of the International Conference on Software Engineering Advances (ICSEA2007)*. 2007.

[Lun+11a] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. *Model-Driven Risk Analysis: The CORAS Approach*. Springer, 2011.

[Lun+11b] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. “Risk Analysis of Changing and Evolving Systems Using CORAS”. In: *FOSAD*. 2011, pp. 231–274.

[Mai+04] Neil Maiden, Sara Jones, Sharon Manning, John Greenwood, and L. Renou. “Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study”. In: *Proceedings of the 16th Conference On Advanced Information Systems Engineering (CAiSE’04)*. Vol. 3084. LNCS. Springer Verlag, 2004, pp. 3–21.

[Mas+06] Fabio Massacci, John Mylopoulos, and Nicola Zannone. “Hierarchical hippocratic databases with minimal disclosure for virtual organizations”. English. In: *The VLDB Journal* 15.4 (2006), pp. 370–387.

[Mas+10] Fabio Massacci, John Mylopoulos, and Nicola Zannone. “Security Requirements Engineering: The SI* Modeling Language and the Secure Tropos Methodology”. In: *Advances in Intelligent Information Systems*. Ed. by Zbigniew Ras and Li-Shiang Tsay. Vol. 265. Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2010, pp. 147–174.

[Mas+12] Fabio Massacci, Deepa Nagaraj, Federica Paci, Le Minh Sang Tran, and Alessandra Tedeschi. “Assessing a Requirements Evolution Approach: Empirical Studies in the Air Traffic Management Domain”. In: *Proceedings of the 2nd International Workshop on Empirical Requirements Engineering (EmpiRE’12)*. 2012, pp. 49–56.

[Mas+13] Fabio Massacci, Federica Paci, Le Minh Sang Tran, and Alessandra Tedeschi. “Assessing a Requirements Evolution Approach: Empirical Studies in the Air Traffic Management Domain”. In: *Journal of Systems and Software* (2013). Article in press.

[MD00] Tom Mens and Theo DŠHondt. “Automating Support for Software Evolution in UML”. In: *Automated Software Engineering* 7, (2000), pp. 39–59.

[MF73] Robert I. Mehr and Stephen W. Forbes. “The Risk Management Decision in the Total Business Setting”. In: *Journal of Risk and Insurance* 40 (1973), pp. 389–401.

[MG11] S. McGee and D. Greer. “Software requirements change taxonomy: Evaluation by case study”. In: *Proceedings of the 19th IEEE International Requirements Engineering Conference (RE’11)*. 2011, pp. 25–34.

[ML05] S. Mannan and F.P. Lees. *Lee’s Loss Prevention in the Process Industries*. 3rd. Vol. 1. Butterworth-Heinemann, 2005.

[Moo03] Daniel L. Moody. “The method evaluation model: a theoretical model for validating information systems design methods”. In: *ECIS*. 2003, pp. 1327–1336.

[MR05] Neil Maiden and Suzanne Robertson. “Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System”. In: *Proceedings of the 13th IEEE International Requirements Engineering Conference*. IEEE Press, 2005, pp. 105–116.

[NB12] A. Niknafs and D.M. Berry. “The impact of domain knowledge on the effectiveness of requirements idea generation during requirements elicitation”. In: *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE’12)*. 2012, pp. 181–190.

[Ncu+07] Cornelius Ncube, James Lockerbie, and Neil Maiden. “Automatically Generating Requirements from i* Models: Experiences with a Complex Airport Operations System”. In: *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ’07)*. Vol. 4542. LNCS. Springer Verlag, 2007, pp. 33–47.

[Nor10] Thomas L. Norman. *Risk Analysis and Security Countermeasure Selection*. CRC Press, Taylor & Francis Group, 2010.

[NT10] Viet Hung Nguyen and Le Minh Sang Tran. “Predicting Vulnerable Software Components using Dependency Graphs”. In: *International Workshop on Security Measurement and Metrics (MetriSec’10)*. 2010.

[Ome+12] Aida Omerovic, Anders Kofod-Petersen, Bjørnar Solhaug, Ingrid Svagård, and Le Minh Sang Tran. *Report on ESUMS Risk Analysis*. Tech. rep. A23344. SINTEF ICT, 2012.

[RH09] Per Runeson and Martin Host. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical Software Engineering* 14.2 (Apr. 2009), pp. 131–164.

[Rob+01] R.M. Robinson, K. Anderson, B. Browning, G. Francis, M. Kanga, T. Millen, and C. Tillman. *Risk and Reliability. An Introductory Text*. 5th. R2A, 2001.

[RR09] J.W Rittinghouse and J.F. Ransome. *Cloud computing: implementation, management, and security*. CRC Press, 2009.

[Rum+04] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[Rus+99] A.M. Russo, B. Nuseibeh, and J. Kramer. "Restructuring Requirements Specifications". In: *IEE Proceedings - Software* 146.1 (1999), pp. 44–53.

[RW11] Nornadiah Mohd Razali and Yap Bee Wah. "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests". In: *Journal of Statistical Modeling and Analytics* 2.1 (2011), pp. 21–33.

[Sch99] Bruce Schneier. "Attack trees: modeling security threats". In: *Dr. Dobb Journal of Software Tools* 24 (12) (1999), pp. 21–29.

[Sen00] A. K. Sen. "The discipline of cost-benefit analysis". In: *Journal of Legal Studies* 29 (2000), pp. 931–952.

[SES08] SESAR. *SESAR D3 – The ATM Target Concept*. Tech. rep. SESAR Initiative, 2008.

[Set+04] Raffaella Settimi, Jane Cleland-Huang, Oussama Ben Khadra, Jigar Mody, Wiktor Lukasik, and Chris DePalma. "Supporting Software Evolution through Dynamically Retrieving Traces to UML Artifacts". In: *Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)*. 2004.

[Sha+09] Glenn Shafer, Vladimir Vovk, and Roman Chychyla. "How to base probability theory on perfect-information games". In: *Bulletin of the European Association for Theoretical Computer Science* 100 (2009), pp. 115–148.

[Sof05] P Soffer. "Scope analysis: identifying the impact of changes in business process models". In: *Software Process: Improvement and Practice* 10.4 (2005), pp. 393–402.

[Sol+07] Bjørnar Solhaug, Dag Elgesem, and Ketil Stølen. "Specifying Policies Using UML Sequence Diagrams–An Evaluation Based on a Case Study". In: *Policies for Distributed Systems and Networks, 2007. POLICY '07. Eighth IEEE International Workshop on*. 2007, pp. 19–28.

[Sou+11] Vítor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos. "System identification for adaptive software systems: a requirements engineering perspective". In: *Proceedings of the 30th International Conference on Conceptual Modeling (ER'11)*. Brussels, Belgium, 2011, pp. 346–361.

[SS13] Bjørnar Solhaug and Fredrik Seehusen. “Model-driven risk analysis of evolving critical infrastructures”. In: *Journal of Ambient Intelligence and Humanized Computing* (2013), pp. 1–18.

[Sto+02] Gary Stoneburner, Alice Goguen, and Alexis Feringa. *Risk Management Guide for Information Technology Systems*. Tech. rep. National Institute of Standards and Technology. U.S Department of Commerce, 2002.

[Sys] *"OMG Systems Modeling Language (OMG SysML), Version 1.2"*. 2010.

[TM11] Le Minh Sang Tran and Fabio Massacci. “Dealing with Known Unknowns: Towards a Game-Theoretic Foundation for Software Requirement Evolution”. In: *Proceedings of the 23th Conference On Advanced Information Systems Engineering (CAiSE’11)*. 2011, pp. 62–76.

[TM13a] Le Minh Sang Tran and Fabio Massacci. *An Approach for Decision Support on the Uncertainty in Feature Model Evolution*. Tech. rep. (under submission to CAiSE’14). University of Trento, 2013.

[TM13b] Le Minh Sang Tran and Fabio Massacci. *Dealing with Known Unknowns: a General Approach for Modeling and Reasoning on Requirements Evolution*. Tech. rep. (to be submitted to the Software and Systems Modeling (SOSYM) journal). University of Trento, 2013.

[TM13c] Le Minh Sang Tran and Fabio Massacci. “UNICORN: A Tool for Modeling and Reasoning on the Uncertainty of Requirements Evolution”. In: *CAiSE Forum*. 2013, pp. 161–168.

[Tra+13a] Le Minh Sang Tran, Bjørnar Solhaug, and Ketil Stølen. “An Approach to Select Cost-Effective Risk Countermeasures”. In: *Data and Applications Security and Privacy XXVII - 27th Annual IFIP WG 11.3 Conference, (DBSec 2013)*. 2013, pp. 266–273.

[Tra+13b] Le Minh Sang Tran, Bjørnar Solhaug, and Ketil Stølen. “An Approach to Select Cost-Effective Risk Countermeasures Exemplified in CORAS”. In: *CoRR* (2013). http://arxiv.org/abs/1302.4689.

[Tra11] Le Minh Sang Tran. “Requirement Evolution: Towards a Methodology and Framework.” In: *CAiSE Doctoral Consortium 2011*. London, 2011.

[Tra13] Le Minh Sang Tran. “Early Dealing with Evolving Risks in Long-Life Evolving Software Systems”. In: *Advanced Information Systems Engineering Workshops – CAiSE Workshops*. 2013, pp. 518–523.

[VB07] Dániel Varró and András Balogh. “The model transformation language of the {VIATRA2} framework”. In: *Science of Computer Programming* 68.3 (2007). <ce:title>Special Issue on Model Transformation</ce:title>, pp. 214 –234.

[Vil+08] Karina Villela, Joerg Dörr, and Anne Gross. “Proactively Managing the Evolution of Embedded System Requirements”. In: *Proceedings of the 16th IEEE International Requirements Engineering Conference (RE’08)*. IEEE Computer Society, 2008, pp. 13–22.

[Vil+10] Karina Villela, Jörg Dörr, and Isabel John. “Evaluation of a Method for Proactively Managing the Evolving Scope of a Software Product Line.” In: *Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ’10)*. Vol. 6182. LNCS. Springer, June 22, 2010, pp. 113–127.

[Woh+12] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. *Experimentation in Software Engineering*. Springer, 2012, pp. I–XXIII, 1–236.

[Yu96] Eric Siu-Kwong Yu. “Modelling strategic relationships for process reengineering”. PhD thesis. Toronto, Canada, 1996.

[Yu99] Eric Yu. *Strategic Modelling for Enterprise Integration*. 1999.

[ZO97] Didar Zowghi and Ray Offen. “A Logical Framework for Modeling and Reasoning About the Evolution of Requirements”. In: *Proceedings of the 3rd Symposium on Requirements Engineering (RE’97)*. RE ’97. 1997, pp. 247–257.

[ZW98] M.V. Zelkowitz and D.R. Wallace. “Experimental models for validating technology”. In: *Computer* 31.5 (1998), pp. 23–31.

[EUR10] EUROCONTROL. *European Operational Concept Validation Methodology, E-OCVM Version 3.0*. 2010.

[Fed09] Federal Aviation Administration. *System Wide Information Management (SWIM) Segment 2 Technical Review*. Tech. rep. FAA, 2009.

[NES11] NESSoS project. *Deliverable D11.2: Selection and Documentation of the Two Major Application Case Studies*. Tech. rep. 2011.

[Pro08] Program SWIM-SUIT. *D1.5.1 Overall SWIM Users Requirements*. Tech. rep. 2008.

[Pro12] Project SecureChange. *Deliverable 3.3: ALGORITHMS FOR INCREMENTAL REQUIREMENTS MODELS EVALUATION AND TRANSFORMATION*. Tech. rep. 2012.

[Pro96] Project PROTEUS. *Deliverable 1.3: Meeting the challenge of changing requirements*. Tech. rep. Centre for Software Reliability, University of Newcastle upon Tyne, 1996.

[WHO09] WHO. *Risk Characterization of Microbiological Hazards in Food - Guidelines.* WHO - World Health Organization, FAO - Food and Agriculture Organization of the United Nations, 2009.