UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Design and Evolution of Sociotechnical Systems

A Requirements Engineering Perspective

Fatma Başak Aydemir

Advisor

Prof. Paolo Giorgini

Università degli Studi di Trento

Co-Advisor

Prof. John Mylopoulos

Università degli Studi di Trento

April 2016

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisors Prof. Paolo Giorgini and Prof. John Mylopoulos for the continuous support of my Ph.D study and related research, for their patience, motivation, immense knowledge and wisdom. Their guidance helped me in all the time of research and writing of this thesis.

Besides my advisors, I would like to thank the rest of my thesis committee: Prof. Giacomo Cabri, Prof. Roberto Sebastiani, Prof. Munindar Singh, and Dr. Anna Perini, for their insightful comments and the lively discussion during my defense. I would like to thank Prof. Munindar Singh also for giving me the opportunity to visit his team and the access to the research facilities at North Carolina State University. The discussions I had with him and his team broaden my horizons and helped me with my research.

I am grateful for collaborating with inspirational researchers during my Ph.D. studies. Dr. Tong Li, Dr. Fenglin Li, and Dr. Amit Chopra were great collaborators with their ideas and hard work. Dr. Fabiano Dalpiaz supported me in doing research, as well as in the writing process with his hands-on approach. Dr. Jennifer Horkoff has been a role model and inspired me with her attention to details, willingness to take initiative, and strong research ethics.

I also thank my friends from the Multiagent Systems Laboratory at the Boğaziçi University. Dr. Özgür Kafalı and Dr. Akın Günay encouraged me throughout my Ph.D studies and contributed to the final version of my thesis with their comments. I'm grateful for Prof. Pınar Yolum for helping me having my first steps into research during my masters studies and providing support during my Ph.D.

I received great emotional support from friends. Fellow Ph.D students in the Software Engineering and Formal Methods froup empathized with me in good and hard times. Dr. Ramona Marfievici shared her wisdom on life and secrets about Trentino with me. We experienced the importance of stamina together. I also cannot thank enough to Dr. Begüm Demir who took care of me and became my family in Trento.

Last but not the least, I would like to thank my family: my parents and to my brother supporting me in every step of my life. This thesis is dedicated to them.

Abstract

Sociotechnical systems are systems of systems where social, technical, and organizational systems interact with each other to satisfy their requirements. The interplay of social and technical systems blurs the borders in between them, and the constant change within and outside the sociotechncial systems create difficulties to manage the overall evolution. This thesis explores the methods to model, analyse, and evolve the requirements of sociotechnical systems. We propose a systematic design process and a formal language to aid social systems refine their requirements into not other requirements but also social interactions to generate system as well as interaction specifications. Although such specifications are useful to generate interaction protocols among systems, they haven't been investigated in detail by the requirements engineering community. We then explore the design space created during the design process with artificial intelligence planing to discover sequence of actions to satisfy requirements with minimal cost.

We adopt an iterative approach for handling requirements evolution and focus on the problem of selecting the optimal set of requirements for the next release. We capture synergies among requirements in goal-oriented requirements models and transform the next release problem into a multi-objective satisfiability modulo theories/optimization modulo theories problem and solve it using an external reasoner. We apply a similar approach for risk analysis using goal models. We model goals, risks, and treatments in three layers and solve multi-objective risk analysis problem with SMT/OMT reasoning. We evaluate our proposal with self-evaluation studies, a case study and scalability experiments and report results. The novelty of these two approaches is the combination of satisfiability analysis with multi-objective optimization for goal models.

Keywords requirements engineering, sociotechnical systems, design, social commitments, planing, next release problem, multi-objective optimization, satisfiability modulo theories, optimization modulo theories, risk analysis, goal-models

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Sociotechnical Systems

Traditional institutions such as government bodies, health-care systems, universities, and many others have been increasingly relying on technical systems to generate and consume information, to communicate with others, and run tasks that are too dangerous, difficult, or repetitive for humans. The rise of certain technologies enable contemporary institutions, such as massive open online courses and open source software communities, where the technical infrastructures are indispensable parts of the organization. Hardware and software systems have been integral components of institutions big and small to operate and reach their goals.

Eric Trist, Ken Bamford, and Fred Emery coined the term sociotechnical systems (STSs) to describe English coal mines, referring to the interrelatedness of human, technical, and organizational aspects [1], [2]. There are several examples of sociotechnical systems in our daily lives, some of which are mentioned above. A smart city is a sociotechnical system, where citizens, province, and law-enforcement officers are few of human and organizational systems, and traffic sensors, garbage collection machines, and lighting infrastructures are examples of technical systems. Similarly, an e-commerce website is another example of sociotehnical systems where the management, employers, purchasing department and customers are social systems that are supported by the technical systems such as the web site, mobile app, source management software and so on.

Günther Ropohl [3] describes sociotechnical systems using general systems theory [4]. In this sense, sociotechnical systems are systems-of-systems where where social systems interact with and through technical ones. Each system that is part of a sociotechnical system is an *action system*, that takes actions to reach its goals, changes its environment, and communicates with others during its operation. According to Ropohl, a sociotechnical system itself is also an action system. We adopt his definition for sociotechnical systems.

Systems that compose a sociotechnical system coordinate, collaborate, and communicate with each other. These activities are necessary for them to satisfy their local goals as well as global (sociotechnical system) goals. On different levels of abstraction, each system of a sociotechnical system can be a sociotechnical system itself, creating a hierarchy,

suggesting that organizations as a whole can be part of sociotechnical system. This is an application of the principle of *excluded reductionism* from general systems theory.

There are three ways of achieving goals of sociotechnical systems in terms of organizational relations: hierarchy, heterarchy, and responsible autonomy. Military and religious organizations exemplify strict hierarchy. In heterarchical sociotechnical systems control shifts around systems based on their capabilities (personality, skills, experience). Consortia established for European projects operate heterachically, where different partners lead the work packages that match to their qualifications and expertise the best. Finally, in case of responsible autonomy the systems have autonomy to decide what to do, but are accountable for the outcome of their decision. Research groups (generally) have responsible autonomy; they can decide on the venues to present their works, candidates to hire, proposals to write and so on. In general, the combination of these three aproaches in various degrees are applied to sociotechnical systems [5].

In a sociotechnical system actions may be executed by technical (hardware or software) or social (humans or organizations) systems, hence the principle of equi-functionality applies. Due to technological advances especially in the artificial intelligence area, the border between social and technical systems become blurry. Technical systems do not take final decisions or set global or local goals as of now, but they highly support social systems for these actions. Technical systems are capable of executing human actions, in some cases performing even better than humans.

In summary, sociotechnical systems are systems-of-systems composed of both social and technical systems. Internal systems as well as the global sociotechnical systems are goal oriented, and capable of taking actions to reach their goals. There is a hierarchical structure that is formed by the ways inner systems collaborate with each other. Yet the social systems still set goals and make final decisions and technical systems are assigned to tasks that are repetitive, dangerous, or difficult for humans.

1.2 Design and Evolution of Sociotechnical Systems

Sociotechnical systems are composed of heterogeneous systems, which might be software or hardware systems, human systems, organizations, and combinations of these. As we discuss above, these systems interact with each other in various ways to achieve the local and global goals. Thus, one of the identifiers of a system is its interactions with others. A system may participate in multiple sociotechnical systems through the interactions it forms. For example, a company interacts with other companies, its customers, and government bodies to conduct business. It may operate in different countries, hence participate in different sociotechnical systems.

An underlying technical infrastructure supports interactions between systems, software is part of this infrastructure. Ideally the software should be transparent, satisfying requirements of the system without being obtrusive. Pervasive [6] and ubiquitous [7] computing paradigms has shifted software design towards this direction, hiding technical details from social systems that use the software. Another significant paradigm that has

facilitated interactions is distributed computing. Distributed computing removed physical barriers for social systems to engage in with each other [8]. Service–oriented computing [9], grid computing [10], application programming interfaces (APIs) [11], among many others, exemplify distributed computing, enabling multiple social and technical systems to use the same software products, form interactions in various forms: sharing their resources, trading goods, and building collaborations.

When a technical system is designed, the requirements of individual systems are considered, but the interactions among the systems are often not taken into account, which results in undesired outcomes. A great example is the 'Flash Crash', where the Dow Jones Industrial Average dropped by over 600 points that resulted in a total loss of $800bn in market value on May 6, 2010 [12]. Although the market recovered within minutes, the 'Flash Crash' got a lot of attention from multiple disciplines from policy making to information systems engineering. The incident was caused by a single large block of sale by a fund management company, leading to a sudden drop and rise in the market. In this case, all technical systems worked as specified, however a legitimate exchange between a social system and the other caused tremondous negative side–effects for the overall sociotechnical system.

Designing social interactions has received attention from the areas of requirements engineering and agent–oriented software engineering. One of the most prominent framework is i^* [13], which is a goal–oriented requirements modeling framework and is the research baseline for this thesis. i^* describes an organizations as a set of actors that interact with each other through dependencies to satisfy their goals. The interaction may be in the form of achieving a goal, executing a task, or providing a resource for another actor. The dependencies in i^* do not capture the two way interactions between actors, they represent on which way an actor relies on the other. Also, i^* framework does not have a modeling process defined to build the social models. In this thesis, we propose a goal–oriented modeling language that captures two-way interaction among systems and a requirements refinement process to build social models.

1.3 Challenges

Designing sociotechnical systems is still an open challenge due to several reasons. *i.* The focus of each system might be different, it is likely that their local goals are *diverse*, *ii.* they may even be *conflicting* with local goals of other systems. *iii.* Due to heterogeneity of sociotechnical systems, participating systems have different technical backgrounds, different capabilities, and use different jargon to express their goals. Interaction is one of the core elements for satisfying local and global goals. We identify the challenge of designing a sociotechncial systems as follows:

C₁ Defining a systematic approach for the design process that considers the heterogeneous structure of sociotechnical systems, diverse goals of participating systems and rthe degree of *autonomy* of each system. The process should provide means to systems to

resolve their conflicts, guide systems through the design activity, let systems specify their local goals, and create a common understanding of the state of the design.

Sociotechnical systems are situated in dynamic environments. Environmental changes may force altering strategies for achieving goals, or even reconsideration of global and local goals. Change in sociotechnical system goals implies evolution of sociotechnical system. Apart from the environmental causes, internal factors may also force evolution. For example, an internal social system may change its local goals, a software product that is part of a technical system may have a new release. If not handled properly, sociotechnical system evolution may cause misalignment among systems, causing failures. In order to prevent such cases, evolution should be controlled by carefully selecting its direction among many possibilities. It is important to consider the objectives of the sociotechnical system, and how well each direction of evolution maximizes (resp. minimizes) these objectives. Challenges related to sociotechnical system evolution are as follows:

C_2 capturing existing and new goals, and synergies that will result from applying changes. There are synergistic relationships between goals apart from the traditional relationships, such as decomposition, make, help, hurt, and break. Synwrgystic relationships are active when both goals are satisfied, and they have impact on criteria other than the satisfaction of the goal, such as a reduction on cost, or increased utility. These synergies should be identified, and captured within goal models.

C_3 finding optimal solutions for the next iteration of evolution. Optimization in goal models has not been explored in detail, yet for the controlled evolution of requirements, optimization analysis is significant for the selection of requirements for the next iteration. Fo example, maximizing the utility while minimizing the cost could be an objective for a sociotechnical system design.

Studies show that risks are highly underestimated in IT projects [14]. Identifying incidental and intentional risks early in the design process is necessary to minimize loss, and propose treatments when possible. The complex structure of sociotechnical systems leads to more complicated relations among the requirements of the systems. For example, one system's requirements may create risks for others, but may help mitigating the impact of other risks. It is difficult for decision-makers to reason on these complex relations and discover the optimal solutions with respect to multiple criteria such as cost, gain, or risk aversion.

C_4 capturing the synergies among goals in terms of risk and mitigation. We need methods capture risk likelihoods and impacts on requirements models, as well as mitigations. Satisfaction of a requirement may create a risk for the satisfaction of another requirement, or may help to mitigate the impact of a risk.

C_5 analyzing risk for sociotechnical system and discovering optimal solutions inline with the risk management strategies of a sociotechnical system. After capturing the risk related synergies, risk analysis should be conducted on the requirements models in

the early stages. Many strategies might be followed for this task, such as minimizing cost, minimizing risk, an maximizing the mitigation impact. We need methods to assess and mitigate risk in requirements models.

1.4 Research Roadmap

Design and evolution of sociotechnical systems is a broad area that includes many challenges with respect to social and technical aspects. Section 1.2 discusses specific challenges that we are interested in within the scope of this thesis. Social interactions should be considered during the design process, as they are the key of satisfying sociotechnical goals. Systems should be part of the design process both to express their goals and to build the interactions needed to achieve those goals.

The evolution of sociotechnical systems should be controlled in order to prevent misalignment within sociotechnical systems, or worse, the collapse as a result. The 'economics' of evolution should be considered, the optimal solution that maximizes (resp. minimizes) the objectives of the sociotechnical system should be found. Risk analyses should be incorporated to these processes from early stages for improved risk management.

We tackle these challenges form a requirements engineering perspective. Our aim is to stand on the shoulder of giants of goal–oriented requirements engineering, revising and extending existing approaches for improving approaches for design and evolution of sociotechnical systems.

1.4.1 Research Questions

Our research objective is to *support design and evolution of sociotechnical systems from the requirements engineering perspective by providing languages, processes, reasoning mechanisms, and tools.* The overall objective of the thesis will be achieved by answering the following research questions.

RQ$_1$ What is an effective process for designing, analysing requirements, and generating specifications for sociotechnical systems?

RQ$_2$ How to analyze requirements models in order to select optimal plans for sociotechnical systems?

RQ$_3$ How to manage the incremental evolution of requirements of sociotechnical systems?

RQ$_4$ What is an effective method for risk analysis for sociotechnical systems?

In order to evaluate our approach towards answering these research question, we identify several success criteria. Table 1.1 summarizes our research questions and the success criteria.

Table 1.1: Research questions and success criteria

RQ	Success Criteria
RQ_1	SC_1 Have a systematic approach that is
	$SC_{1.1}$ able to conduct modeling requirements and interactions,
	$SC_{1.2}$ applicable to different domains,
	$SC_{1.3}$ equipped with a formal semantics.
RQ_2	SC_2 Have a systematic approach that is
	$SC_{2.1}$ able to capture requirements and interactions,
	$SC_{2.2}$ able to automatically create plans to execute at run-time.
RQ_3	SC_3 Have a systematic approach that is
	$SC_{3.1}$ able to capture requirements and synergies for next iteration,
	$SC_{3.2}$ able to support automated analysis for finding optimal solutions.
	$SC_{3.3}$ able to provide solutions in acceptable time.
RQ_4	SC_4 Have a systematic approach that is
	$SC_{4.1}$ able to capture risk and synergies in requirements models,
	$SC_{4.2}$ able to support automated analysis for finding optimal solutions.

In order to address RQ_1, 'What is an effective process for modeling requirements and interactions of sociotechnical systems?' we need a systematic approach that supports modeling activities to specify a sociotechnical system.

$SC_{1.1}$ *Conduct modeling requirements and interactions in few steps.* There should be a systematic modeling process that supports designing sociotechnical systems starting from requirements of participating systems, and builds a specification by refining the initial requirements, forming interactions among systems, and creating new systems when necessary. The process should respect the characteristics of sociotechnical systems such as heterogeneity, autonomy of individual systems, and open systems principle.

$SC_{1.2}$ *Applicability to different domains.* The process should be domain-independent, that is, it can be used to specify different sociotechnical systems from different domains.

$SC_{1.3}$ *Formal semantics.* The design produced as the product of the process should not only capture and convey the specification of a sociotechnical system, but also allow formal reasoning to answer questions, such as 'Is the design complete?', 'Are there any requirements that are not accounted for by a system?'. Having a formal semantics enables us to reason on the models.

RQ_2 'How to analyze requirements models in order to select optimal plans for sociotechnical systems?' concerns with automatically generating concrete action plans based on the specifications of sociotechnical systems. Our success criteria for this question includes the following:

$SC_{2.1}$ *Capture requirements and interactions.* The modeling framework should capture requirements and interactions within sociotechnical system and *visualize* them. Visual models better conveys actors, their requirements, their interactions, and a s a result a better overview of the sociotechnical system that is modeled.

$SC_{2.2}$ *Automatically create plans as a solution to the design problem.*A plan, which is a list of executable actions, should be generated based on the requirements and interaction models. The plan should respect the commitments specified at design-time and satisfy the requirements of the systems.

We ask RQ_3 'How to manage the iterative evolution of sociotechnical systems?' to make progress towards challenges C_2 and C_3 discussed in Section 1.2. We need a systematic approach that supports controlled, iterative evolution. Success criteria for RQ_3 are:

$SC_{3.1}$ *Capture requirements and synergies for next iteration.* We provide a conceptual model to capture requirements, both implemented and under consideration for the next iteration, and the synergies rise from including certain requirements together.

$SC_{3.2}$ *Support automated analysis for finding optimal solutions.* It is not trivial for requirements analyst to find a solution that *i.* satisfies requirements and *ii.* optimizes the objectives stated for the model. Automated analysis supports the analyst to make a decision for the next iteration of evolution.

$SC_{3.3}$ *Provide solutions in acceptable time.* The analysis should scale well with respect to different model sizes, number of alternative solutions available in the model, and the degree of connectivity of model elements.

RQ_4 'What is an effective method for risk analysis for sociotechnical systems' addresses C_4 explained in Section 1.2. We need a systematic approach that can be integrated into early phases of requirements modeling and analysis. Success criteria for RQ_4 are:

$SC_{4.1}$ *Capture risk and synergies in requirements models.* We provide a conceptual model to capture requirements, risks, treatments, and the synergies between these model elements.

$SC_{4.2}$ *Support automated analysis for finding optimal solutions.* It is not trivial for requirements analyst to find a solution that *i.* satisfies requirements and *ii.* optimizes the objectives stated for the model. Our approach supports several risk management strategies, finding optimal solutions for different objective functions.

1.4.2 Evaluation Activities

Our proposal towards answering the research questions presented in Section 1.4.1 has multiple components. We perform several evaluation activities to verfiy that these components as part of solutions we provide establish the success criteria also presented in

Section 1.4.1. The categories of evaluation activities that are carried out to evaluate one or more components are listed below.

E_1 *Self– evaluation study via scenarios.* For this activity we apply the design process, the language, and other artifacts to a scenario. The scenario can be derived from interviews with stakeholders (domain experts, end– users, and others), examination of a domain, or documents describing a domain. Scenarios from previous research work can be used. The aim of this activity is to demonstrate the applicability of the proposal in the given domain and scenario.

E_2 *Self– evaluation study of the visual notation.* The visual of the elements of the artifacts are analyzed according to visual design principles, mainly for the principles stated for modeling languages by Moody [15], [16]. The results discuss the cognitive and visual loads of models, and effectiveness of the visual notation.

E_3 *Case study.* We interact with the intended user groups, provide them with a scenario. End users build models following the design process and using the language and tools developed by us. We then collect feedback from users via interviews, questionnaires, and discussions and report them.

E_4 *Formal semantics.* We propose formal frameworks in order to provide unambiguous description mechanisms, verify and analyze models.

E_5 *Scalability study.* This activity evaluates the scalability of the proposed analysis with respect to increasing model size, the degree of connectivity of model elements, and the number of solutions available in the model.

Table 1.2: Satisfaction of Succes Criteria via Evaluation Activities

	Design			Operationalization		Evolution			Risk Analysis	
	$SC_{1.1}$	$SC_{1.2}$	$SC_{1.3}$	$SC_{2.1}$	$SC_{2.2}$	$SC_{3.1}$	$SC_{3.2}$	$SC_{3.3}$	$SC_{4.1}$	$SC_{4.2}$
E_1 S.E. Scenario	✓			✓		✓			✓	
E_2 S.E. Visual				✓					✓	
E_3 Case Study		✓								
E_4 Formal Semantics			✓			✓			✓	✓
E_5 Scalability								✓		

Table 1.2 maps the success criteria we present in Section 1.4.1 to evaluations activities conducted to verify their satisfaction.

1.5 Overview and Contributions

1.5.1 Designing Sociotechnical Systems with Protos

RQ_1, 'What is an effective process for modeling requirements and interactions of sociotechnical systems?' Protos is a requirements engineering approach that gives prominence to the interactions of systems and specifies a sociotechnical system in terms of the social interactions of systems. Protos addresses challenges with respect to design of sociotechnical systems stated in Section 1.2. The contributions of Protos beyond the state of the art are the following:

Requirements problem for sociotechnical systems. Zave and Jackson [17] characterize the requirements problem in terms of a set of domain assumptions, machine specification, and a set of stakeholder requirements. The problem is to come up with a specification that satisfies the requirements given the domain assumptions. Such characterization ignores the role of social systems towards a solution for the requirements problem, therefore insufficient to capture the requirements problem for sociotechnical systems.

We propose a new formulation of the requirements problem for sociotechnical systems. Instead of relying on a single machine specification, our formulation relies on a system specification and an interaction specification, that is, a protocol, for the satisfaction of requirements under domain assumptions. A system might be a social or a technical system, which does not have to reveal all of its local goals, such as the goals related to other sociotechnical systems than the one being specified.

Modeling Language. We provide a modeling language to capture the requirements and interactions. We separate systems parts of a sociotechnical system (teams) from the systems participating at the design process (stakeholders). Social commitments [18] capture interactions. We define refinement relations for each type of construct that maps one construct to another, which is assumed to be more concrete and *refined*.

Design Process. Protos defines a set of formal rules to be applied by stakeholders during design. Each rule aims at moving the design to a state that is more elaborated, better defined, and more refined. Applications of these rules to the initial set of constructs creates the *design space* of the sociotechnical system, with alternative *design paths*.

Evaluation. We have self–evaluated the design process and language using a car insurance scenario [19]. We have also conducted a case study with PhD students to further evaluate the effectiveness of the approach using London Ambulance Scenario [20].

1.5.2 Exploring Sociotechnical System Designs

RQ_2 'How to analyze requirements models in order to select optimal plans for sociotechnical systems?' We propose DEST (DEsigning Sociotechnical sysTems), a goal-oriented requirements modeling language, to represent systems (as actors) in a sociotechnical system, their goals, and their social interactions. The language also supports expressing the capabilities of a system to achieve goals, conflicts in goals, temporal ordering constraints on requirements, and priorities.

Formal semantics. We formalize the notion of a sociotechnical system configuration and design plan, and shows how these concepts are useful for describing how to design an STS from scratch. A plan is a sequence of actions that leads to the establishment of a network of interactions among the subsystems of the STS that fulfills their requirements. We describe the planning of finding a plan as an AI (artificial intelligence) automated planning problem, and encode it in PDDL (planning domain definition language) [21].

Evaluation. For the evaluation of our approach, we have built a travel reimbursement scenario by interviewing administrative personnel, project investigators, students, and professors at the University of Trento. We have used this scenario to self-evaluate the modeling language. We have also evaluated the visual notation.

1.5.3 Evolution of Requirements

RQ_3 'How to manage the iterative evolution of sociotechnical systems?' We support controlled evolution of sociotechnical systems in order to prevent misalignment among systems. Our starting point is software evolution, which is handled in iterations, in forms of 'updates', 'versions', 'releases', and so on. The problem of selecting the optimal set of requirements to be included in the next release of a software product is stated as the next release problem, first as a single objective optimization problem by Bagnall et al. [22], and later as a multiobjective optimization problem by Zhang et al. [23]. These approaches follow search–based software engineering and apply genetic search algorithms to disocver optimal solutions. We propose a more expressive modeling approach to capture the hiearchy among requirements as well as several interpedendencies between them.

Modeling Language. We provide a goal–oriented requirements modeling language to capture requirements in consideration for the next release as well as the requirements that have been implemented. We also represent 'synergies' arising from having certain requirements together in a solution. One example of such a synergy is one requirement increasing the customer value of another, such as implementing a graphical user interface to increase the value of a functionality. Our meta-model also enables representing exclusions, and temporal constraints. Requirements may have associated reward and cost values, and synergies have intensities to determine optimal solutions during analysis.

Formal Semantics. We transform goal oriented requirement models into Satisfiability modulo theories (SMT) / optimization modulo theories (OMT) clauses. It is possible to specify lexicographic order of objectives, linear multi–objective functions, or a combination of both.

Next Release Tool. Next Release tool is a standalone Eclipse RCP application. It has a graphical modeling editor to help users to build syntactically correct models. It also checks the well-formedness of the models. It automatically converts next release models into SMT/OMT formulas, and runs external OMT solver OptimathSAT [24] to obtain the optimal solution that satisfies the mandatory requirements respecting constraints in the model. The tool highlights the solution in the model and produce textual reports.

Evaluation. We have interviewed with a research programmer and investigated bug reports for a research tool to build a scenario for self–evaluation. We have conducted three

different scalability experiments to evaluate the scalability of our approach with respect to model size, model connectivity, and number of alternative solutions.

1.5.4 Risk Analysis

RQ_4 'What is an effective method for risk analysis for sociotechnical systems' We extend the three layered-approach of Asnar et al. [25] with relations to capture synergies in the models. Asset layer includes the initial requirements, which are threatened by the risks represented in the event layer. In order to mitigate these risks, treatments are considered in the treatment layer. As for the synergies, risks may increase the likelihood or the severity of impact other risks. Similarly, treatments reduce the likelihood, or the severity of impact.

Formal Semantics. We map requirement–risk models in Satisfiability modulo theories (SMT) / optimization modulo theories (OMT) clauses. We rely on OptiMAthSAT [24] as SMT/OMT solver.

Risk Management Strategies. We implement several optimization schemes that serve for different risk management strategies from risk aversion to cost aware and to more opportunistic strategies. For each strategy we identify the combination of the OMT optimization scheme and demonstrate how different strategies can be combined together.

Evaluation. We use the Loan Origination Process (LOP) scenario presented in [25] to self–evaluate the modeling language. We also provide an analysis of the visual notation.

1.6 Publications

We list the published work related to this thesis here.

International Conferences

1. Amit Chopra, Fabiano Dalpiaz, Fatma Başak Aydemir, Paolo Giorgini, John Mylopoulos, and Munindar Singh (2014), *Protos: Foundations for Engineering Innovative Sociotechnical Systems*, In Proceedings of the 22$^{\text{nd}}$ IEEE International Requirements Engineering Conference, pages 53–62.

2. Fatma Başak Aydemir, Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos (2014), *Exploring Alternative Designs for Sociotechnical Systems*, In Proceedings of the 8$^{\text{th}}$ IEEE International Conference on Research Challenges in Information Sciences, pages 1–12.

3. Jennifer Horkoff, Fatma Başak Aydemir, Feng-Lin Li, Tong Li, John Mylopoulos (2014), *Evaluating Modeling Languages: An Example from the Requirements Domain*, In Proceedings of the 33$^{\text{rd}}$ International Conference on Conceptual Modeling, pages 260-274.

4. Fatma Başak Aydemir, Paolo Giorgini, and John Mylopoulos (2016), *Risk Analysis with Goal Models*, In Proceedings of the IEEE 10th International Conference on Research Challenges in Information Sciences, (*to appear*).

International Workshop and Demos

1. Fatma Başak Aydemir, Paolo Girogini, and John Mylopoulos (2014), *Designing Sociotechnical Systems with Protos*, In Proceedings of the 7th *i** Workshop.

2. Fatma Başak Aydemir, Dagmawi Neway Mekuria, Paolo Girogini, and John Mylopoulos (2015), *Next Release Tool*, 34th International Conference on Conceptual Modeling, [online], `http://er2015.dsv.su.se/files/2014/07/demo-3.pdf`

Under preparation

International Conferences

1. Fatma Başak Aydemir, Dagmawi Neway Mekuria, Paolo Giorgini, and John Mylopoulos, *Scalable Solutions to the Next Release Problem: A Goal-Oriented Perspective*, Submitted to 24th IEEE International Requirements Engineering Conference.

2. Jennifer Horkoff, Fatma Başak Aydemir, Evellin Cardoso, Tong Li, Alejandro Mate Morga, Elda Paja, Mattia Salnitri, John Mylopoulos, and Paolo Giorgini, *Goal-Oriented Requirements Engineering: A Systematic Literature Map*, Submitted to 24th IEEE International Requirements Engineering Conference.

1.7 Organization of the Thesis

The rest of the thesis is organized as follows.

- Chapter 2 presents the related work for this thesis. We review goal-oriented requirements engineering languages and methodologies in Section 2.1. We investigate the state of the art for modeling social interactions in Section 2.2. Section 2.3 presents research for software evolution and risk analysis techniques and risk modeling languages are examined in Section 2.4.

- Chapter 3 presents Protos approach for designing sociotechnical systems.

- Chapter 4 includes our work on sociotechnical system design. The meta–model is presented in Section 4.1. Section 4.2 lists the actions that are part of execution plan. PDDL mapping is described in Section 4.3. Section 4.4 reports on the results of the evaluation of the virtual notation.

- Chapter 5 provides our modeling language (Section 5.2), encoding of next release models into SMT/OMT (Section 5.3), and different kind of analyses possible for the next release problem (Section 5.4). The scalability analysis results are presented in Section 5.5.

- Chapter 6 proposes a goal-oriented risk analysis framework that includes interdependencies among treatments and risks in terms of likelihood and generate optimal solutions with respect to multiple objectives such as goal rewards, treatment costs, or risk factor. Section 6.1 presents our meta–model for requirement–risk models. Section 6.3 lists different types of analyses supported by our approach.
- Chapter 7 draws conclusions for this thesis, describes ongoing work, and presents future directions.

Chapter 2

Related Work

Ross and Schoman state that requirements analysis must state *why* a system is needed, *what* system features satisfy this need, and *how* the system is to be constructed [26]. Zave defines requirements engineering as the branch of software engineering that deals with the real-world goals for, functions of and constraints on software systems in [27]. More recently Nuseibeh and Easterbrook describe requirements engineering as the process of the process of discovering the purpose of software systems by identifying stakeholders and their needs and by documenting these in a form that is amenable to analysis, communication, and subsequent implementation [28]. van Lamsweerde lists the activities withing requirements engineering as domain analysis, elicitation, negotiation and agreement, specification, specification analysis, documentation, and evolution [29].

Goal–Oriented Requirements Engineering (GORE) is a requirements engineering approach where goals are used to used to conduct the activities stated by van Lamsweerde [30]. Goals have been used in artificial intelligence before their adoption by requirements engineering. There are several definitions of a goal in the requirements engineering literature. van Lamsweerde defines a goal as an objective that the system should achieve through cooperation cooperation of agents in the software-to-be and in the environment [30]. Anton describes goals as high-level objectives of the of the business, organization or system that aid decision making at various levels within an enterprise [31]. The common perspective is that goals represent objectives, desired states of affairs.

Goal models are hierarchical structures where top goals are refined into lower level goals. Higher level goals capture *raison d'être* behind lower level ones, and lower level goals capture *how* higher level can be achieved. van Lamsweerde [30] and Lapouchnian [32] provide an overview for the goal–oriented requirements engineering literature. In the next section we review the most influential goal modeling approaches.

2.1 Goal–Oriented Requirements Engineering

2.1.1 Goal Modeling

NFR framework [33], [34] captures and reasons on *non-functional requirements* (NFRs), global requirements such as maintainability, reliability, performance and so on. This work highlights the importance of non-functional requirements and use them to justify decisions during software development process. The authors state that a qualitative or quantitative approach can be adopted, yet obtaining quantitative measurements are too hard, so they present a qualitative treatment of NFRs. There are five major components of the NFR Framework. *i.* a set of goals that represent NFRs, design decisions, and arguments in support or against other goals, *ii.* a set of link types that relate goals to other goals, *iii.* a set of methods to refine goals into other goals, *iv.* a set of rules to infer interaction among goals, and *v.* a labeling procedure to decide the level of satisfaction of an NFR. Actually, instead of satisfied or achieved, the framework uses the term *satisficed*, a term coined by Simon in [35], for there is no clear cut criterion for full satisfaction of an NFR. During the design process goals are refined into other goals using the refinement rules, as a result Soft-goal Interdependency Graph (SIG) is built. The requirement analyst can select between alternative results by applying label propagation algorithms to verify how well each alternative satisfice the high-level NFRs.

Jureta et al. propose Techne [36] modeling language. In Techne requirement statements are labeled, therefore categorized into five categories: goals, quality constraints, softgoals, tasks, and domain assumptions. In Techne quantitative treatment for softgoals are adopted. Softgoals are approximated into quality constraints in the models, and preference relations are used to represent preferred approximation by the requirements analyst. Our evaluation of Techne [37] shows that this concept is difficult to realize during early requirements engineering stage.

Li et al. [38] treat NFRs as *qualities* over requirements. Qualities are captured as mappings, and have a domain and codomains in requirements models. The authors provide a set of refinement operators in [39] to incrementally refine informal, vague, ambiguous requirements into a formal, practically satisfiable and measurable specification.

KAOS (Knowledge Acquisition in automated Specification) [40] is a goal–oriented requirements engineering methodology. The main focus of KAOS is to formalize requirements which are conceptualized as goals. KAOS is presented as a supporting methodology fro requirements acquisition phase, where the main objective is to understand *why* a system is needed. In KAOS models, each node represent either a goal, action, agent, entity, or event and links between these nodes represent relations between these concepts. The meta-model of KAOS has three layers: the meta-level presents the domain independent abstractions such as agents, actions, and relations, the domain level describes concepts specific to a domain, and the instance level refers to specific instances of concepts presented in the domain level. KAOS methodology the instances of the concepts presented in the meta-model (goals, objects, actions, agents) are identified and reduced until they are concrete enough, and then agents are assigned responsibilities based on their capabilities.

KAOS methodology later extended to handle obstacles [41], and to build anti-models, models that capture intentions of an attacker [42], and to focus on security requirements [43], and so on.

GBRAM (Goal–Based Requirements Analysis Method) [44] is a method to identify, analyze, elaborate, and specify requirements. GBRAM starts by exploring available information related to requirements (policies, interviews, transcripts, and so on). Next, goals and the responsible agents are identified, and the goals are classified and dependency relations are built. 'Refine' activity refers to pruning of the goal set. Then, the prune goal set is elaborated by analyzing obstacles and constructing scenarios to uncover hidden goals. Finally the goals are operationalized, they are transformed into operational requirements for the final requirements specification.

Rolland et al. [45] use goals to discover scenarios, and scenarios to discover goals. They call a goal and a scenario pair a *requirement chunk* (RC). Requirement chunk models capture RCs, where high-level, fuzzy RCS are refined into concrete RCs. The authors make a distinction of refinement and AND/OR decomposition. The conceptual meta-model is supplemented with a set of guidelines to help discovery of goals and scenarios.

i^* [13], [46] is an agent–oriented goal modeling framework for business process reengineering, organizational modeling, and requirements engineering. In other approaches such as KAOS and GBBRAM, goals exist regardless of agents, and agents are assigned goals towards the final steps of the requirements process. On the other hand, goals existed only if an associaoted agent exists in i^* models, so agents have goals, and they depend on each other to satisfy them. i^* has a broad view of agents, that are actors. Agents are concrete actors with certain capabilities. Roles can also be actors, they are abstract actors with responsibilities and expectations captured as goals, or positions which are socially recognized roles. Actors have intentional, they have their goals. There are two types of goals, hard and softgoals, the difference is that softgoals do not have a clear cut criteria for achievement as opposed to hardgoals. Hardgoals, softgoals, resources, and task may contribute positively or negatively to other intentional elements. Strategic Dependency (SD) graphs capture dependencies among actors. Strategic Rationale (SR) models capture the rationale of every actor in terms of hardgoals, softgoals, resources, tasks, dependencies and relations between these. i^* framework has been very influential in the goal-oriented requirements engineering and has been extended in multiple ways: SI/*/ [47], to better capture security and trust requirements, URN as a combination of GRL and use case maps [48], and Tropos [49] to name the few.

Tropos [49] is an agent–oriented software engineering methodology based on i^*. Tropos spans five phases: early requirements analysis, late requirements analysis, architectural design, detailed design, and implementation. Tropos relies on i^* meta-model yet it puts some additional restrictions such as allowing decompositions only between homogeneous elements.

2.1.2 Analyzing Goal Models

Goal models are used to capture and communicate requirements, yet their structure naturally allows reasoning on the achievement of goals as well as some domain properties such as security and trust. There are number of analysis methods proposed in the literature of goal–oriented requirements engineering.

Satisfaction analysis concerns whether goals in the model are achieved or denied. The procedure is to assign a set of initial values to the model and propagate these values either forward or backward direction of the links present in the model.

Amyot et al. [50] propose a quantitative reasoning technique that is based on forward propagation of labels for URN, which combines GRL and use case maps. The labels are determined by a predefined set of rules that combines evidence from contribution links, which can be make, help, some positive, unknown, negative, break, and hurt. There are seven satisfaction levels (labels), which are denied, weakly denied, weakly satisfied, satisfied, conflict, unknown, and none.

Chung et al. [34] propose a forward reasoning technique to reason on NFRs, similar to the technique presented in [50] there are several labels to represent the degree of satisfaction. Instead of relying of predefined set of rules to combine the evidence from contribution links, this method relies on human interference.

Giorgini et al. devise forward propagation qualitative and quantitative reasoning techniques techniques [51]. Conflicting contributions are combined according to a set of rules. The results are presented as achieved/denied for the qualitative analysis or in terms of numeric values for the quantitative analysis.

Sebastiani et al. [52] encode the satisfaction analysis as a satisfiability problem and allows user to define additional constraints on the models. If the problem is satisfiable the solver returns an assignment for the model. There might be multiple assignments, in order to distinguish between these weights are assigned to goals, and minimum weight assignment is returned by the solver. Asnar et al. [53] apply this technique [52] to reason on risk in goal models and to find minimum cost solutions, where cost values are assigned to task as their weights. Wang et al. [54] adapt the same technique to diagnose run-time failures using goal models.

Giorgini et al. present forward and backward reasoning applied in Tropos in [55]. The techniques presented in [55] is a combination of those presented in [51] and [52] applied to Tropos models, so contribution links are automatically combined. For satisfaction and denial, partial (P) and fully (F) labels are used.

Horkoff and Yu [56] transform goal models in conjunctive normal form (CNF) and iteratively apply SAT solver and human intervention. Human judgment is used to decide for the combination of the conflicting contribution links. The approach is applied to i^* models that are formalized by the authors.

Letier and van Lamsweerde [57] present techniques for specifying probabilistic degrees of goal satisfaction. For each refinement in the goal model refinement equations are defined over quality variables, and quantitative values of quality variables are propagated bottom-up. Having different equations for each refinement, the authors are able to distinguish

between alternatives in terms of quality values.

Some methods apply maps the design problem into AI planning problem to determine the sequence of actions to satisfy goals. Bryl et al. [58] create social interaction networks within goal models by finding plans in terms of cost. The authors later extend this approach and apply it to i^* and Tropos models [59] as a requirements analysis method. The authors define planning actions and encode the models in PDDL. The encoding of the actions (domain file) and the encoding of the modeling domain (the problem file) is given as input to an external planner and if there is a sequence of actions that satisfy the top level goals, the planner returns a plan. The author later defines metrics to evaluate the plans in [60]. Three metrics defined are *i.* the length of the proposed plan, *ii.* overall plan cost, *iii.* degree of satisfaction of NFRs. Asnar et al. [61] combine the planning-based approach presented in [58] and goal-based risk analysis technique presented in [53].

Liaskos et al. [62] distinguish between mandatory goals and preferences (nice-to-have goals). Preferences are goals of stakeholders whose satisfaction is desired yet their denial does not invalidate a solution. The authors use priorities to set a level of importance on preferences. The authors also introduce temporal preferences, temporal constraints over the sequences of goals of the mandatory decompositions. Goal models are formalized into hierarchical task network (HTN) and PDDL, and an external planner HTNPlan-P [63]. The preference score of a plan is the sum of prioritization (weights) of preferences satisfied by a plan. The plan with the highest preference score is returned by the planner.

Nguyen et al. [64] encode goal models in satisfiability modulo theories/optimization modulo theories and can perform not only satisfaction analysis but also multi-objective optimization analysis. It is possible to set global constraints, such as setting a budget limit for solutions, on the models as well as local constraints on goals, such as specifying a budget limit for the solution of a single goal. The framework allows prioritization of goals and is able to capture and reason on conflict relations. Goals can be mandatory and optional goals are supported. The framework is fully formalized and has tool support which is proved to be scalable. The tool automatically converts constrained goal models into SMT/OMT formulas and feeds these as in put to external solver OptiMathSat [24]. The optimal solution retrieved from the solution is highlighted on the graphical goal model. Although there have been attempts for single objective optimization analysis [25], [52] these approaches are not as expressive as constrained goal models in terms of expressing multiple global and local constraints, multiple objectives and objective schema (lexicographic and linear objective functions).

2.2 Modeling Social Interactions

Social interaction occurs when two or more social entities interact with each other. In physical world examples of social interactions are talking to a person, purchasing something from a shop keeper. Technical systems increasingly support social interactions, sending an e-mail, an instant message, sharing a photograph on Instagram or SnapChat, making an online purchase are all forms of social interactions supported by technical

systems. Social interactions are studied in different areas, our focus is on agents and multiagents systems, conceptual modeling, software and service engineering.

AUML (Agent Unified Modeling Language) [65] The first layer represents the overall protocol, using packages and templates. The second level includes interactions among agents. Sequence diagrams are directly borrowed from existing UML specification. Collaboration diagrams are diagrams similar to sequence diagrams. In sequence diagrams an ordered sequence of interactions can be followed from top to bottom, in collaboration diagrams the number of the interaction is labeled on the links between agents, which can be placed anywhere. Activity diagrams consist of pool of agents, and activities in these pools showing how they trigger other operations and events. Statecharts capture valid states, where agent interactions are labeled on the edges between states. Finally the third level concerns the internal processing of agents. In this layer, activity diagrams and statechart diagrams are used to depict internal processing of agents.

KQML (Knowledge Query and Manipulation Language) is an agent communication language (ACL), independent of content syntax and applicable ontology. There are three components of a KQML message: content, communication, and message. The content is the actual content of the message, and can be in any language. The communication part includes meta information such as the identity of the sender of the identifier of the message. The message part identifies the interaction protocol and supplies a speech act. A speech act is an utterance that has performative function [66]. The syntax of KQML includes balanced parentheses lists.

FIPA Agent Communication Language [67] is another speech act-based agent communication language. Similar to KQML, the conent of FIPA ACL can be in any language. Sender, receiver, and reply-to fields are used to represents the agents included in the communication. Communicative act is the performative of the message. The content field is the content of the message. The receiver interprets the content together with the communicative act.

Singh [68] points that these languages do not solve the interoperability problem among heterogeneous agents. The author argues that the communicative acts are not clear since they are based on interpretation of the receiver or the sender agent. The languages do not cover all of the most common communicative acts, which are assertives, directives, commissives, permissives, prohibitives, declaratives, and expressives, focusing on assertives and declaratives only. Moreover, the languages rely on agents' internal state rather than providing means for communication independent of the internal beliefs, goals, desires, and intentions of agents. The author states that existing ACLs design autonomy (minimizing requirements on agent builders) and execution autonomy (agent's freedom of choosing its own actions), and communication is inherently public and depends on the social context of agents.

Kumar et al. [69] propose a landmark based protocol specification, where a landmark is a state of affairs that must hold during protocol execution. Protocols are joint action expressions that can be derived from partially ordered landmarks. The success of the protocol execution can be verified by the achievement of the landmarks. Protocols can be combined for more complex protocols. This approach can only work for cooperative

agents, and it is not designed for open systems.

Cheong and Winikoff [70] propose a goal–oriented approach to agent interaction. Hermes approach has three parts: a methodology for for designing goal-based interactions, failure handling mechanisms, and a process for mapping design artefacts to an executable implementation. The authors argue that message–based protocols restrict autonomy of agents, and limit the flexibility (shortcuts) and robustness of agent interaction. Goal–oriented interaction is described in terms of interaction goals, available actions, and constraints. Message sequences among agents are not predefined, but they emerge during interaction. The methodology is consisted of six steps: role and goal identification, interaction goal hierarchy, action identification, action sequences, message identification, and message definitions. Action failures can be recovered by re-performing an action. If an interaction goal fails due to an action failure, the interaction can be terminated or agents may request a rollback to the previous interaction goal. This approach may fit for the greenfield design of a multiagent system, but it does not consider an open system where new agents with different interaction goals can enter to the multiagent system.

Singh proposes social commitments in [18] to overcome challenges stated above regarding agent communication. The framework is called spheres of commitments and relies on eight assumptions set by the author: agents can be recursively composed of heterogeneous individuals or groups of agents, agents are autonomous but constrained by commitments, social commitments cannot be reduced into internal commitments, commitments are in general revocable, commitments exist and can be manipulated all in social context, commitments rely on the social structure of the multiagent system in which they exist, but they also help creating the social structure, the semantics of commitments must be distinguished from their pragmatics, and finally commitments are evaluated in the world not in the minds of agents. A commitment c is a four place relation $c=C(x,\ y,\ G,\ p)$ where x is the debtor agent, which is committed, y is the creditor agent, which receives the commitment, G is the context and y is the discharge condition. So $c=C(x,\ y,\ G,\ p)$ is a commitment from x to y in the context of G and for the proposition p. Commitments can be created, discharged (satisfied), canceled, released (eliminated), delegated, or assigned.

Yolum and Singh [71], [72] build flexible communication protocol using commitments. In their formal framework a base level commitment $C(x,y,p)$ refers to a commitment from debtor x to creditor y for the condition p. A conditional commitment $CC(x,y,p,q)$ denotes that if the condition p is brought out, x will be committed to y for q. The framework is formalized in event calculus [73]. A protocol specification is a set of initiates and terminates clauses, and the protocol execution is a set of happens clauses. The interaction is meaningful, meanings of the actions are captured via commitments, verifiable via commitments, and declarative. Later Desai et al. [74] specify interaction protocols for business processes using commitments, their framework is formalized in the π-calculus [75]. Telang and Singh [76] extend Tropos framework [49] with commitments, identifying goals and goal dependencies between roles, refining these into task and task dependencies. Finally, task dependencies are refined into conditional commitments, enabling verification of agent interactions at run-time. Dalpiaz et al. [77], [78] propose a conceptual architecture for handling adaptation in open systems, demonstrating their approach on Tropos framework

[49] extended with commitments. Chopra et al. reason on service–oriented architectures [79] and on protocols [80] using agents and commitments, their frameworks are formalized in Datalog [81].

2.3 Evolution

2.3.1 Software Evolution

Software evolution is a term in use since 1960s [82]. Later Lehman and his colleagues state overall eight laws on software evolution. The first three laws are presented in [83], which are *i.* a software product either undergoes continual change or become less and less useful, *ii.* as software evolves, its complexity increases unless some work done to main or reduce it, *iii.* software growth inevitably slows as it grows older. Two more laws, *iv.* the average rate of change in an evolving software system tends to stay constant (invariant) over the life time of a system, and *v.* the average incremental growth remains invariant or declines as systems evolve, are introduced in [84]. The sixth law, 'functional content of a software must be continually increased to maintain user satisfaction over its lifetime' is included in [85], Final two laws, *vii.* software will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment, *viii.* software development processes constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved are presented in [86]. These laws are effective on what Lehman et al. call type E systems, that are, software systems that deal with, or solve problems in real world. There is supporting and contradicting of these laws [87], yet our point is that software evolution is a valid concern for over 40 years.

Chapin et al. [88] state that software evolution spans the time between the initial creation of software to its retirement. Their definition of software evolution includes the term 'version', where time period between two versions may last from less then a minute to decades, and the later version having new properties or functionality from the prior version. In their classification, software evolution may effect business rules and software properties whereas software maintenance effects documentation and support interface additional to these. Software evolution can be enhancive, corrective, or reductive on business rules. Types of software evolution applicable on software properties are adaptive and performance. Maintenance types on business rules are the same as evolution types, and on software properties preventive and groomative in addition to adaptive and performance. Maintenance on documentation can be updative (the software does not change) and reformative (the software changes). Finally, maintenance on support interface can be of evaluative, consultive, or training activities. As the classification points out, evolution and maintenance activities can occur in different parts and layers of abstraction and as the paper states one activity may trigger other activities in different parts.

2.3.2 Requirements Evolution

Zowghi et al. [89], [90] specifically focus on the evolution of requirements. They propose a framework formalized in Telos [91]. Requirements are kept in a belief base, and the framework manages changes in requirements models (beliefs), checking also consistency and completeness. The approach has explained using a toy example and intended for systems monolithic software systems, making the assumption that the requirements should be complete and consistent. Such assumptions are not realistic for modern software, or sociotechnical systems.

Barber et al. [92] propose applying system engineering process activities (SEPA) for managing the evolution of requirements. The authors identify necessary steps to practice 'good' software engineering and manage evolving requirements as mapping new requirements into old ones, creating traceability links from the existing implementation to new requirements, implementing components for new requirements that are not covered by the old implementation, and integrating the old and the new implementation. A funnel metaphor is used to describe SEPA activities, where there is continual gathering of knowledge (requirements and domain knowledge) which are refined into component-based system design specification. The objective is to create a single technical system, the methodology does not handle relations among requirements, or requirements from multiple stakeholders. The infrastructure reference architecture classes diagram presented in the paper, people are clearly separated from technology. So the approach is not suitable for handling requirement for sociotechnical systems.

Anderson and Fellici [93] present an empirical investigation of controlled evolution of requirements. The study is conducted for the requirements of a technical system, a safety-critical avionics system. Two significant results for our research work are *i.* the number of requirements increases over software releases, *ii.* the evolution tends to show some dependencies between requirements. So requirements evolution is a valid challenge that should be attended over the course of software life cycle, and dependencies between requirements have a significant role during evolution. The authors define metrics for monitoring evolution of requirements in [94].

Zowghi and Gervasi [95], [96] propose a formal foundation to ensure consistency, completeness, and correctness of requirements. They present the requirements problem in terms of evolving requirements and domain assumptions. For sociotechnical system design where multiple systems are involved, conflict and inconsistency in requirements are inevitable. Rigid systems that do not allow modeling inconsistency and conflicts are not suitable to model and analyze requirements in sociotechnical systems.

Ernst et al. [97], [98] find incremental solutions to evolving requirements. The authors distinguishes between the evolving requirements and adaptive requirements. In adaptive requirement frameworks, overall set of requirements do not change, the implementation is monitored and alternative valid solutions are selected if/when a failure of satisfying requirements detected. The authors claim that evolving requirements result from unanticipated changes. We do not agree with this claim, for software has long been released in iterations, with each release having a different set of requirements, and such change

is anticipated, and it is part of the evolution of the requirements. The authors state a requirements problem for evolving requirements, and four different alternatives for valid solutions: the standard solution that ignores the implemented requirements and treat the problem as a greenfield design problem, the minimal change effort solution that selects the new requirement set based on its distance from the previous set, the maximal familiarity solution that selects the solution with the maximal reuse, and solutions that re-use past solutions. The proposed framework uses knowledge base [99] to store information acquired during requirements acquisition and domain modeling and query this information to discover and compare alternative solutions. The paper reports the results of scalability experiments.

Ali et al. [100] propose a goal–oriented requirements engineering approach to handle requirements evolution. The authors define requirements evolution in a broad way, for example, they classify switching to an alternative solution due to a failure caused by a change in the context. An 'autonomic' evolution is to automatically select a variant based on historical success/failure data, such a case is classified as adaptation by others, such as [98]. Designer supported evolution occurs when the requirements analyst modifies the goal model.

Ghaisas and Ajmeri [101] develop a knowledge–assisted ontology–based method to deal with requirements evolution. The authors advocate for knowledge re-use for 50% of requirements knowledge for similar system can be re-used completely or with minimal modification [102]. K-RE tool developed by the authors enable knowledge sharing between stakeholders using environmental context ontology, problem domain ontology, generic requirements ontology, and RE process ontology. The tool also helps analyst handling change requests, listing related requirements for a specific request.

2.3.3 The Next Release Problem

Bagnall et al. [22] define and solve the next release problem where each customer has their set of requirements, and a weight representing the customer's importance to the company. All possible software enhancements are represented in a directed, acyclic graph, where edges denote requirements dependencies. Each requirement has an associated cost. The problem is to find a set customers, whose requirements to be satisfied in the next release of the software, such that the sum of customer weights in the selected set is maximized subject to the sum of cost of requirements to be satisfied. This problem statement has attract attention and many solutions have been proposed. The authors apply three different methods: *i.* knapsack formulation, *ii.* greedy algorithms, and *iii.* local search algorithms.

Ruhe and Greer [103], [104] propose EVOLVE+ approach to deal with the release planning problem. The problem is similar to the next release problem, but the objective is to find solutions for multiple releases. The authors state that the overall goal is to find an assignment of requirements to increments that maximize the sum of weighted priorities. Similar to the formulation of the next release problem, the stakeholders have different weights representing their associated priorities. Each requirement also has an associated

effort and risk estimation as well as a resource constraint. Requirement dependencies are shown as a partial order. They propose an iterative approach that applies a genetic algorithm to determine L best solutions that respect constraints (temporal constraints and requirements dependencies). Saliu and Ruhe [105] apply analytic hierarchy process (AHP) [106] to determine the weights used for objectives. Ngo-The and Ruhe [107] integrate a multi-criteria decision aid method (ELECTRE IS) for deciding the final solution. Du and Ruhe [108] apply rough set analysis (RSA) [109] and dependency network analysis (DNA) [110] techniques to the framework.

Baker et al. [111] report on the performance of the search–based approaches for the next release problem. They compare simulated annealing and greedy algorithms and human judgment. Both search-based methods outperform expert judgment. Simulated annealing algorithm also performs better than greedy algorithm.

Zhang et al. [23] define the multi-objective next release problem, adding 'value' objective to 'cost' objective. In this formulation of the next release problem, requirements dependencies are ignored. The authors solve the problem in four different ways: *i.* non-dominated sorting generic algorithm-II (NSGA-II) [112], *ii.* Pareto GA algorithm, *iii.* single–objective GA, *iv.* random search. NSGA-II algorithm performs the best in a scalability experiment where the highest number of requirements and customers are 140 and 100, respectively.

Durillo et al. [113], [114] solve the multi-objective next release problem with three different algorithms: *i.* NSGA-II, *ii.* multi-objective cellular genetic algorithm [115], and *iii.* random search. In the scalability experiment number of requirements vary between 40 and 200 whereas the number of customers vary between 15 to 100. NSGA-II algorithm outperforms the others.

Jiang et al. [116], [117] use approximate backbone–based multilevel algorithm to solve the next release problem. Their approach reduce large problem instances into smaller ones. The experiment results show that such divide and conquer approach canperformance than direct solving approaches.

Sagrado et al. [118] compares genetic algorithms, simulated annealing, and ant colony optimization techniques for the solution of the next release problem. Ant colony optimization techniques demonstrate better performance than the other algorithms.

Veerapen et al. [119] apply integer linear programming techniques to single and bi-objective next release problem. They report that exact solutions for large single objective and small bi-objective problems can be found in reasonable time using linear programming techniqes thanks to advancements in solvers. On large bi-objective instances, execution times can still be significant. The Integer Linear Programming-based approximate algorithm outperforms the NSGA-II genetic approach on large bi-objective instances.

Pitangueria et al. [120] transform the multi-objective next release problem to satisfiability problem. They use stakeholder dissatisfaction risk, cost, and value objectives. Two data sets are used to experiment with this approach, the first one has 50 requirements and 4 stakeholders whereas the second has 25 requirements and 9 stakeholders. The authors compare their approach with NSGA-II algorithm which performs better in terms of time but finds suboptimal solutions.

2.4 Risk Assessment

2.4.1 Risk Analysis

Uncertain events are defined over their likelihood to happen and severity. probabilistic risk analysis (PRA) is used for quantitative analysis [121]. Quantitative approaches such as FMECA (Failure Mode, Effects, and Critical Analysis) [122] use qualitative values. For example frequent, reasonably probable, occasional remote, and extremely likely is used for distinguishing frequencies for likelihood. Expectancy loss or risk factor, which is the product of the likelihood of an event and the severity, is used to determine the criticality of a risk.

CORAS [123] is a risk analysis framework that models, analyzes, and treats risks. In CORAS, each risk is analyzed independent of system or malicious actor objectives. Each risk has a single impact and likelihood, so the effect of the same event on multiple objectives are considered as separate risks.

Feather [124] proposes Defect Detection and Prevention (DDP) as a three layered (objectives, risks, and mitigation) approach. Each risk has a likelihood as the probability of occurrence, and the severity of a risk is represented by an impact relationship between the risk and an objective. Mitigation and risks have affect relations. There are no dependencies captured within objectives, risks, or mitigations. The author later models risks as fault trees, providing a more structured representation. 'What if' question is investigated by turning mitigations on and off.

2.4.2 Goal–Oriented Risk Analysis Approaches

KAOS [40] is extended with the concepts of obstacle [41] and anti-goals [42]. Obstacles are situation that lead to goal failure, they create unintended risks for a system. On the other hand anti-goals are malicious intentions of a malicious actor, such as an attackers threatening a system. Van Lamsweerde et al. provide a systematic approach to derive and resolve obstacles in [41] and building anti-models to capture intentions of attackers and mitigate threats in [42].

Asnar et al. [25], [53] propose Goal–Risk (GR) Framework for modeling and reasoning about risk during requirements engineering process. Goal–Risk models have three layers. The asset layer includes requirements represented as goals. Goal can be AND/OR decomposed and contribute to each other as in i^* and Tropos. The event layer is consisted of events, each event has an associated likelihood to happen and and impact on some of the goals captured in the asset layer. The severity of the impact denotes how the event affect the satisfaction of a goal. Tasks presented in the treatment layer mitigate risks. Forward and backward label propagation algorithms are used in combination to decide satisfaction and denial of goals, multiple evidence is combined according to predefined rules. The framework discovers optimal results with respect to a single object, total cost of tasks and goals. Risk assessment process proposed by the framework has three steps: *i.* finding alternative solutions, *ii.* evaluating each alternatives against relevant risks, and

iii. assessing the countermeasures to mitigate risks. The framework is equipped with two tools, GR-Tool and SI* Tool.

Mayer et al. [125] et al. extend *i** with concepts related to security risks. The approach first identifies business assets in the models and then set security goals on these assets. 'Controls' are modeled to ensure the satisfaction of the security goals. Controls have an associated cost so the authors use cost-benefit analysis together with satisfaction analysis.

Matulevicius et al. [126] focus on handling security related risks in information systems. The framework extends Secure Tropos framework [127], an extension of Tropos framework focusing on modeling and analyzing security. Similar to [25], constructs in the language are categorized into three categories. Asset related constructs are actors, hardgoals, softgoals, resources, plans, and secure goals. Risk related constructs are the same but malicious versions of the original, shown using a different syntax in models. Impact relation is defined to represent the effect of the risk related constructs on asset related constructs. Treatment related constructs are the same as asset related concepts, with the addition of mitigates relation. Risk assessment process with asset identification, continues with risk analysis and then treatment identification. The framework provides satisfaction analysis only.

Siena et al. [128] model risks in open source software in terms of situations, risks, and goals and use label propagation algorithms to check whether the goals are satisfied under presence of risks. Their models do not include treatments for risks. Later Costal et al. [129] combine *i** and RiskML [128] to align business goals and risk in open source software and use existing reasoning methods [130] to analyze the achievement of stakeholder goals. The reasoning focuses on propagating the impact of risks to related goals.

Chapter 3

Protos: A Methodology for Designing Sociotechnical Systems

We define a *SocioTechnical System* as one involving interactions between humans and organizations (*principals*) facilitated by *technical artifacts*, including software. STSs under this definition include two or more autonomous parties, who ordinarily act and interact in ways that promote their respective agendas. Thus the participants may act in ways that are unexpected by others.

We further restrict our attention to STSs whose rules of encounter are formally represented, whether created via explicit engineering or otherwise. The benefit of an "institutionalized" STS is that its explicit rules of encounter facilitate reasoning by (prospective) participants about whether and how to participate in the STS. Examples of such STSs can be found in many domains, such as healthcare, finance, transportation, and commerce. STSs, especially those with explicit rules of encounter, provide a conceptual basis for innovation by the participants. (Other STSs may also support innovation but we limit our claims to institutional STSs.)

Innovations in STSs include the emergence of new social practices. For example, beginning from goals of passengers and transporters, we might design an airport as a hub where they can execute their transactions. However, an enterprising person could invent the notion of an airport as a venue for shopping in general (as Brendan O'Regan did in 1947). Similarly, the goals of providing capital for new ventures yield a market for public offerings of stocks, which have morphed into the financial systems of today. Notice that not every innovation is desirable and in general some participants would gain and some would lose from any innovation.

Classical Requirements Engineering (RE) approaches fail to sufficiently support innovation and thus do not help realize the innovative potential of STSs. Classical RE begins from an elicitation of the goals of stakeholders, followed by an analysis phase that produces a specification of a software system that would satisfy those goals given some assumptions about its operating environment. A limitation of such approaches is that the goals of the stakeholders used as a basis for modeling may have little in common with the goals of the participants in the STS (whom we call "principals"). We posit that a

goal-based treatment of STSs undercuts innovation. It offers up the following dilemma. Either the participants innovate but in an ad hoc manner (such as on social media today, where memes and conventions such as emoticons emerge sporadically) or the participants are regimented to the original goals and do not innovate at all. Regimentation is not viable for autonomous participants. In practice, ad hoc innovation is what we see most often.

Motivation. Our research question is as follows: {How can RE facilitate innovation in STSs? That is, how may the stakeholders of an STS systematically produce a specification that facilitates innovation}?

We posit that the RE effort be broken into two logically distinct phases: (1) coming up with the rules of encounter in an STS and (2) given the rules of encounter, coming up with models of participants (e.g., their policies) that determine how they participate. A common representation of the rules of encounter ties these two phases together. The first phase is carried out jointly by (or on behalf of) the stakeholders specifying the STS. The second phase is carried out separately by (or on behalf of) stakeholders for each participant.

A major benefit of the aforementioned common representation is that not only does it enable the interoperation needed to realize the STS but also decouples the participants with respect to what is not specified, and thus frees them up to innovate. As long as they comply with the rules of encounter, the innovation is not ad hoc.

STSs are exemplified by many practical settings. Let us consider a healthcare system, which we understand as involving interactions among physicians, nurses, patients, hospitals, insurance sellers, and regulatory bodies. There are two potential ways one can approach the requirements engineering of such a system given stakeholder requirements. The traditional or *regimented* approach is to specify a software module that all the principals would use. The other or *interaction-oriented* approach involves is to specify the interaction protocols that would support meeting the requirements.

An interaction protocol would describe how any principal adopting one or more *roles*, such as *hospital* or *physician*, would interact with others. In particular, the protocol would specify the social expectations that principals playing one of the roles could have of principals playing other roles. Each principal would be free to develop its own software in accordance with its requirements. For example, different principals playing the role of *hospital* could adopt different implementations and policies by which they conduct their business. We refer to each principal's software as its *agent*.

Traditional works on requirements engineering (RE) and methodologies [17], [131] generally tend to be regimented: they address the specification of software conceptualized as a machine that resides within STSs, not the sociotechnical system itself. Approaches such as Tropos [49] begin from modeling the sociotechnical system, but end up with a regimented software system.

We refine the above question further to emphasize the foundational aspects of RE, namely, /What is a suitable formulation and formalization of the STS design space that is conducive to the autonomy both of stakeholders and principals?/ To focus on the representational aspects, we place as out of our scope the important challenges of negotiation

among stakeholders and of collaborative and concurrent engineering of a sociotechnical system.

This chapter makes the following contributions:

- We introduce a way of specifying an STS as a protocol in terms of roles and their social commitments [18] to each other. Our approach applies to social expectations generally, though for concreteness, we focus on commitments here.
- We present a model for requirements satisfaction in which stakeholder requirements are satisfied by a protocol specification, modulo any stated assumptions.
- We propose a generalization of one of the foundational axioms of RE that accommodates the separation of concerns between specifying an STS and an individual principal specifying its agent.
- We present an abstract design process for STSs that extends the classical idea of refinement. We refer to this extended notion of refinement as *social refinement* to emphasize the refinement of requirements into commitment-based specifications. We illustrate the process in a systematic case study of the London Ambulance System.

The rest of the chapter is organized as follows. Section 3.1 introduces background on protocols and requirements refinement. Section 3.2 revisits the traditional RE problem to accommodate STSs and shows that specifying STSs and specifying agents are two independent problems. Section 3.3 motivates and formalizes a set of refinement reductions to obtain STS specifications from stakeholders' requirements. Section 3.4 applies our approach to the London Ambulance System. Section 3.5 summarizes our contributions, and outlines future directions.

Acknowledgements. This chapter is based on our collaborative work presented in [132].

3.1 Research Baseline

We adopt a scenario from automobile insurance based on Browne and Kellet's [19] real-life description.

3.1.1 Classical Formulation of the Requirements Problem

Zave and Jackson [17] characterize RE in terms of A (a set of domain assumptions), M (a software (machine) specification), and R (a set of stakeholder requirements). A requirements engineer's task is to come up with a software specification and the domain assumptions that together guarantee that the requirements are met, which Eq. 3.1 shows:

$$A, M \vdash R \tag{3.1}$$

3.1.2 Commitment Models

A commitment represents a directed social expectation between principals. We express a commitment as a four-tuple *C(debtor, creditor, antecedent, consequent)* the debtor is committed to the creditor that if the antecedent holds, the consequent will hold [133]. A commitment is detached when its antecedent holds, unless it has timed out. It is discharged when its consequent holds, unless it has timed out.

Example 3.1 *The insurance company commits to the car owner to reimburse any damages if the insurance policy is valid: C(Insurer, Car Owner, hasValidInsurance, getReimbursed).*

Example 3.2 *If the Car Owner has a valid policy, the above commitment is detached and the following unconditional commitment holds: C(Insurer, Car Owner, true, getReimbursed).*

If the car owner is reimbursed, the insurer's commitment is discharged. The commitment is violated if the car owner owns a valid policy but does not get reimbursed for damages.

A *protocol* specifies the rules of encounter among principals adopting roles in it. In general, a commitment is established by communication from its debtor to its creditor: it is thus autonomously created and becomes part of the social state of the interacting parties. However, we concentrate here on the commitments themselves without regard to the communications that bring them about. Thus, here, a protocol specifies a set of commitments, such as in Example 3.1. By adopting roles in a protocol, a principal would become the creditor of some commitments and debtor of others.

Commitments, like goals, are a high-level abstraction. However, unlike goals, commitments provide a standard of correctness for interactions among principals that is independent of their mental states. Thus, for example, a particular insurance company may intend to avoid paying for the damages of a particular car owner even if the owner has valid coverage. If the insurance company goes through with its intention, though, it would violate its commitments to the car owner.

3.1.3 Refinement

Refinement is a foundational concept in software engineering [134]. Goal refinement, in particular, is a fundamental concept for systematically extracting a specification from a set of requirements, e.g., in Tropos.

Design refinement is a relation between design problems p and p', where p' is an incremental improvement of p, $p \hookrightarrow p'$, and a solution for problem p' also constitutes a solution for problem p. A *design space* $(P, R_{\hookrightarrow}, p_0)$ consists of an abstract set of design problems, P; a root problem, $p_0 \in P$; and a refinement relationship $R_{\hookrightarrow} \subseteq P \times P$. A root problem is one that is not a refinement of any other problem.

In Zave and Jackson's terminology an engineer could begin from the requirements in R and progressively refine them to produce an implementable M. Software development, then, is concerned with implementing M.

3.2 Requirements Engineering for Sociotechnical Systems

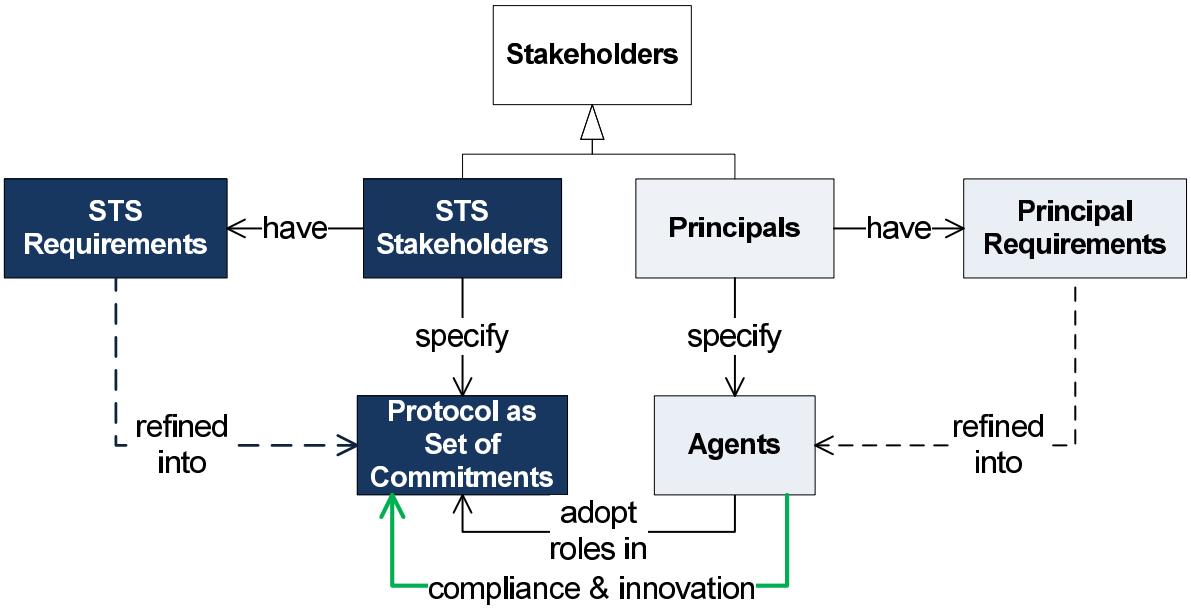


Figure 3.1: The overall design space for sociotechnical systems

Figure 3.1 illustrates the overall design space for STSs as discussed above. Stakeholders are of two kinds: STS stakeholders and principals (that is, runtime participants in the STS). The principals of an STS might have been involved as STS stakeholders in its design, but that is not necessary. The innovation opportunity of STSs arises partly because its principals may never have been imagined as its stakeholders during design, though the principals should have something in common with the STS stakeholders. The dark boxes (and relationships) capture the design space for STS stakeholders. The light boxes (and relationships) capture an individual principal's design space. Whereas STS stakeholders specify a protocol by refining STS requirements, a principal specifies an agent from its requirements. The principal, via its agent, adopts one or more roles in the protocol. By decoupling STS requirements from principal requirements and concomitantly decoupling protocol (STS) specifications from agent specifications, we enable a principal to comply or not with a protocol. Additionally, a principal may comply with a protocol while functioning in novel ways, thereby promoting innovation.

Below, for clarity, we identify STS stakeholders with roles (based on the intuition that stakeholder types and STS roles would normally coincide) and use proper names to identify

principals. Thus, for example, in an automobile insurance system, the stakeholders include *Insurer*, *Mechanic*, and *Car Owner*. Alessia and Cristina are particular mechanics; Great Insurance Co. is a particular insurance company.

3.2.1 Rethinking Requirements Engineering Characterization

The conception of the design space in Figure 3.1 necessitates a rethinking of Zave and Jackson's formulation. Eq. 3.1 is adequate for refining the stakeholders' requirements into a specification of a machine, i.e., the software that would reside in an STS. For example, we might specify software that would support a car owner to file a claim and an insurance staff member to approve the claim and assign a mechanic. The formulation captures the essence of regimented approaches, as illustrated by Example 3.3.

Example 3.3 *Let stakeholders Insurer and Mechanic get together to design software for an automobile insurance STS (we neglect Car Owner for simplicity). Let R_1 be the set of Insurer's requirements, containing only one requirement: any replacement parts ordered by a mechanic that are priced above \$500 must be approved by the Insurer. This requirement, following Eq. 3.1, would be refined into a software specification that disables for mechanics the functionality of ordering parts priced over \$500 unless Insurer's approval is recorded. The specification can be implemented using an access control module.*

If a principal who plays *Mechanic* wants to order parts without the insurance company's approval (e.g., because of urgency or because the approver—(staff member)—is on holiday), the above-mentioned software would not allow the mechanic to proceed. The mechanic would be forced to wait for approval—the software, effectively, regiments interactions [135].

What we would like instead is to come up with a specification that captures the essence of the stakeholder requirement and yet does not regiment the actions of any principal in the STS. This is possible if stakeholder requirements are refined into, not a machine, but a protocol specification. Any principal who adopts a role in the protocol is free to specify (and implement) its own agent in accordance with its own requirements. Eq. 3.2 captures the above intuition. Ag and S are the sets of agent and protocol specifications, respectively. Eq. 3.2 separates the essential nature of the interaction in the protocol from the autonomous decision-making of the participants involved in its enactment.

$$A, Ag, S \vdash R \tag{3.2}$$

Note that when Ag is a singleton and S is thus a null protocol, Eq. 3.2 reduces to Eq. 3.1. Therefore, Eq. 3.2 generalizes the traditional requirements specification problem equation, Eq. 3.1, to a setting of multiple autonomous principals. Example 3.4 illustrates Eq. 3.2.

Example 3.4 *Consider R_1 from Example 3.3. Let S_1 be the protocol specification* {C(*Mechanic*, *Insurer*, p 500, getApproval(*item*) *precedes* order(*item*))}. *For simplicity, let A_1 (the set of assumptions) be empty. Let $Ag_1 = \{i, m\}$; and i and m be the agents of Great Insurance Co. and*

Alessia, respectively. Suppose Alessia has designed m to obtain approvals before ordering any parts. The overall system is correct, that is, $A_1, Ag_1, S_1 \vdash R_1$.

Example 3.5 *Modify Example 3.4 such that Alessia's agent m never asks for preapproval, so it would violate the above-mentioned commitment to obtain preapproval. Let's refer to this agent as* m' *and let* $Ag'_1 = \{i, m'\}$. *Then* R_1 *wouldn't be satisfied either, i.e.,* $A_1, Ag'_1, S_1 \nvdash R_1$.

3.2.2 Modularity of the Design Space

We can achieve the separation of design spaces illustrated in Figure 3.1. A protocol provides a solution for the requirements, independently of the agents who would ultimately play roles in that protocol to instantiate the STS. That is, given some assumptions A_S of the domain, protocol S ensures R under the assumption E that all principals adopting roles in the protocol satisfy their commitments. In other words, the following holds.

$$A_S, E, S \vdash R \tag{3.3}$$

Example 3.6 *Returning to our example, under the assumption E (here meaning that any principal playing Mechanic would satisfy the commitment to get approval),* $A_1, E, S_1 \vdash R_1$.

Assuming E during STS design does not mean that any principal who wants to play a role in the STS must be compliant. Its purpose is to capture precisely the intuition that it is the *satisfaction* of the commitment (and not, for instance, its *creation*) that satisfies some requirement. Also note that E is not relativized to the particular STS (that is why there is no E_1 as there are R_1, S_1, and A_1). The assumption E is purely formal: it assumes the satisfaction of whichever commitments there are in the specification; in particular, assuming E is not a matter of choice for the stakeholders.

We are concerned with the process of designing protocols from requirements, as in Eq. 3.3 (and the dark boxes in Figure 3.1). Example 3.7 illustrates how agent specifications, as in the lighter boxes in Figure 3.1, fit into the overall requirements satisfaction argument.

Example 3.7 *We know that* $A_1, E, S_1 \vdash R_1$. *Cristina wants to play Mechanic in* S_1. *Cristina's requirements are* R_c *and she wants to design an agent c accordingly.* R_c *may or may not contain requirements imposed by the protocol. Let's say* R_c *contains the requirements and c is specified appropriately following Eq. 3.1, that is,* $A_c, c \vdash R_c$. *As before, let i (Great Insurance Co.'s agent) play Insurer and, further, that* $A_i, i \vdash R_i$. *Putting everything together, we get* $A_i, A_c, A_1, \{i, c\}, S_1 \vdash R_1$. *This is like Eq. 3.2:* $A = A_i \cup A_c \cup A_1$; $Ag = \{i, c\}$, *and* $S = S_1$. *(The assumption E does not figure in the final equation as it is "replaced" by actual agent specifications.)*

Example 3.7 demonstrates that even the simple modification of introducing a protocol yields interesting payoffs. The protocol provides a precise description of the nature of the interactions among principals but leaves open the possibility of enacting the protocol in

multiple ways based on their own requirements, thus enabling innovation. For example, this approach accommodates heterogeneous software for different mechanics, and enables ordering before getting the approval (although a mechanic who does so may lose money if the insurance company denies the authorization).

3.3 Social Refinement

We seek a theory of design for STSs founded on the concept of refinement. Since the concepts in terms of which STSs are conceived (e.g., roles, protocols, and commitments) are fundamentally different from those of traditional software engineering(e.g., goals, functions, and actions for requirements, statements and variables for programs), our task is to define new refinements that are specific to STS design problems.

Below, we introduce our model elements, use them to define a design configuration, and introduce a design process.

3.3.1 Model Elements

The following are the key primitives of Protos.

Proposition, which represents a state of the world in the domain of the STS. A proposition may be atomic or composite (representing the conjunction or disjunction of propositions). The set of all propositions is $\mathcal{P}$.

Stakeholder, an autonomous entity present during the design process of an STS.

Team, one or more stakeholders who function as a cohesive unit for design purposes. That is, no team is empty. We denote that τ_i is a subteam of τ as $\tau_i \sqsubseteq \tau$. We write a team τ that comprises subteams τ_i as $\tau = \bigsqcup_i \tau_i$. We do not consider the subtleties of organizational structures, though our approach could be enhanced to incorporate such models. The set of all teams is $\mathcal{T}$, and includes individual stakeholders as unary terms.

Commitment, a social relationship between a debtor team and a creditor team, referring to an antecedent proposition and a consequent proposition. The set of all commitments is $\mathcal{C}$. Thus, $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T} \times \mathcal{P} \times \mathcal{P}$.

Refinement, a reflexive, transitive, antisymmetric relation between propositions. We notate refinement as $\hookrightarrow$ where $a \hookrightarrow b$ means that a refines into b (that is, b is a refinement of a). Thus, $\hookrightarrow \subseteq \mathcal{P} \times \mathcal{P}$. The refinement of commitments and other constructs follows from the above. We lift refinement to design configurations where one configuration refines into another.

Conflict, an irreflexive, symmetric relationship over propositions. We notate conflict as $\oplus$ where $p \oplus q$ means that p conflicts with q. Thus, $\oplus \subseteq \mathcal{P} \times \mathcal{P}$.

Requirement, a representation of an expectation that a team would like to achieve a proposition. The set of all requirements is $\mathcal{R}$. $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{P}$. The notation $\mathsf{R}(\tau, \pi)$ means that $(\tau, \pi) \in \mathcal{R}$.

Onus, a representation of the assumption that a team takes on the onus of ensuring a proposition. $\mathsf{O}(\tau, \pi)$ means that team τ takes on the onus for ensuring proposition π. We include onus assertions with other assumptions.

3.3.2 Design Configurations

We use these model elements to talk about design episodes. A design episode proceeds from one design configuration to another by systematically applying refinements. We define the following concepts to help us describe design episodes.

Requirements are given as R, a finite set of assertions describing what each team (stakeholder) wants to achieve in the STS. The teams are autonomous, each holding a stake in its own requirements, though a team's requirements may hold for its subteams. Thus, $R \subseteq \mathcal{R}$.

Specifications are given as S, a finite set of assertions describing how the STS to be will function. As motivated above, these assertions describe the interactions in the STS but not the internal details of any of its participants. The interactions are naturally captured as social relationships among the participants. Thus, $S \subseteq \mathcal{C}$.

Domain assumptions are given as A, a finite set of assertions that must hold true in order to ensure that the specifications will satisfy the requirements. The set of possible domain assumptions is $\mathcal{A} = \{p | p \in \mathcal{P}\} \cup \{\mathsf{O}(\tau, p) | \tau \in \mathcal{T} \text{ and } p \in \mathcal{P}\} \cup \{a \hookrightarrow b | a, b \in \mathcal{P}\}$.

Needs are given as N, a finite set of requirements. That is, $N \subseteq \mathcal{R}$. N represents the set of requirements that are yet to be addressed during the design episode.

The foregoing leads to a definition of a design configuration. %

Definition 3.1 *Given a set of stakeholder teams $\mathcal{T}$ and a set of propositions $\mathcal{P}$, a* design configuration *is a tuple $\langle S, A, N \rangle$, where $N \subseteq \mathcal{R}$; $S \subseteq \mathcal{S}$; and $A \subseteq \mathcal{A}$.*

Table 3.1 contains an illustration of the complete process of deriving specifications from requirements for an automobile insurance scenario. C, I, M, G, and F represent the stakeholders—/Car Owner/, *Insurer*, *Mechanic*, *Manager*, and *Finance*, respectively. *Manager* and *Finance* are subteams of *Insurer*; the former approves the payments and the latter makes the transactions. In Table 3.1, the S, A, and N cells of each row constitute a design configuration. We refer to the table to illustrate our formal definitions.

The following definitions are needed in Section 3.3.3 to specify how a design process may begin, terminate, and whether its outcome is consistent and meets the requirements.

Definition 3.2 *A design configuration* $\langle S, A, N\rangle$ *is* initial *for requirements* R *if and only if* $N = R$*,* $S = \varnothing$*, and* $A = \varnothing$*.*

In Table **??**, Row 1 refers to the initial design configuration. *Car Owner*'s requirement is to be prepared for the emergencies: {R(c, prepared)}, *Insurer* has the requirement of selling insurance: $\mathsf{R}(I, sold)$, and *Mechanic* wants to get paid for his repair services.

Definition 3.3 *A design configuration* $\langle S, A, N\rangle$ *is* final *if and only if* $N = \varnothing$*.*

In Table 3.1, Row 8 refers to the final design configuration.

Definition 3.4 *A design configuration* $\langle S, A, N\rangle$ *is* consistent *if and only if* $A \cup S \nvdash \mathsf{false}$*.*

Definition 3.5 *A design configuration* $\langle S, A, N\rangle$ satisfies *requirements* R *if and only if* $A \cup S \vdash R$*.*

3.3.3 Design Process

The requirements of stakeholders feed a design (refinement) process, which we imagine as being conducted by stakeholders and facilitated by requirements engineers. The output of the process is an STS specification, and a set of domain assumptions. We model the design process as iteratively taking a design configuration and producing another, refined, design configuration through an application of one of the above reductions, beginning from an initial configuration and ending in a final configuration.

We initialize a design configuration from the requirements by treating each requirement as the technical debt of the relevant team. The design process iteratively addresses each need. A team may take on the onus for any of its needs. Alternatively, it may obtain a commitment from another team, in which case its need would change to the antecedent of the commitment. A design episode concludes when a configuration is obtained that resolves all needs of all teams.

Potentially, more than one reduction may apply on a given design configuration. That is, the design space can be large. Some explorations of it may end up in failure. An exploration is consistent when it constitutes a series of refinements from an initial to a final consistent configuration.

Definition 3.6 *A* design step *takes as input a configuration* $\langle S, A, N\rangle$ *and produces a configuration* $\langle S', A', N'\rangle$ *provided we can conclude* $\langle S, A, N\rangle \hookrightarrow \langle S', A', N'\rangle$*.*

In Table 3.1, going from one row to the next is a design step (the latter row is annotated with the applied reduction from Section 3.3.4).

Definition 3.7 *A* design path *for requirements* R *is a finite series of configurations* $\langle S_0, A_0, N_0\rangle \ldots \langle S_n, A_n, N_n\rangle$ *where (1)* $\langle S_0, A_0, N_0\rangle$ *is initial for* R*; (2)* $\langle S_n, A_n, N_n\rangle$ *is final and consistent; and (3) for each* i*,* $0 \leq i < n$*,* $\langle S_i, A_i, N_i\rangle \hookrightarrow \langle S_{i+1}, A_{i+1}, N_{i+1}\rangle$ *is a design step.*

Table 3.1: Insurance illustration refinement

	Specifications (S)	Assumptions (A)	Needs (N)	Refinement Type
1	$\varnothing$	$\varnothing$	R(C, prepared), R(I, sold), R(M, paid)	
2	$\varnothing$	$A_1 = \{$prepared $\hookrightarrow$ covered $\wedge$ eme $\}$	R(C, covered $\wedge$ eme), R(I, sold), R(M, paid)	Need refinement
3	C(C, I, covered , sold), C(I, C, sold , covered)	A_1	R(C, eme $\wedge$ sold), R(I, covered), R(M, paid)	Cyclic commitments
4	C(C,I, repaired, data $\wedge$ fee), C(I, C, data $\wedge$ fee, repaired)	$A_2 = A_1 \cup \{$ sold $\hookrightarrow$ data $\wedge$ fee, covered $\hookrightarrow$ repaired$\}$	R(C, eme $\wedge$ data $\wedge$ fee), R(I,repaired), R(M, paid)	Commitment refinement i
5	C(C, I, found $\wedge$fixed, data $\wedge$fee), C(I, C, data $\wedge$fee, found $\wedge$fixed)	$A_3 = A_2 \cup \{$ repaired $\hookrightarrow$ found$\wedge$ fixed $\}$	R(C, eme $\wedge$ data $\wedge$ fee), R(I, found $\wedge$ fixed), R(M, paid)	Commitment refinement i
6	C(C, I, found $\wedge$fixed, data $\wedge$fee), C(I, C, data $\wedge$fee, found $\wedge$fixed), C(M, I, paid, fixed), C(I, M, fixed, paid)	A_3	R(C,eme $\wedge$ data $\wedge$ fee), R(I, found $\wedge$ paid), R(M, fixed)	Cyclic commitments
7	C(C, I, found $\wedge$fixed, data $\wedge$fee), C(I, C, data $\wedge$fee, found $\wedge$ fixed), C(M, I, paid, fixed), C(I, M, fixed, paid)	$A_4 = A_3 \cup\{$ G $\sqsubset$ I, F $\sqsubset$ I, paid $\hookrightarrow$ transfer $\wedge$ app$\}$	R(C, eme $\wedge$ data $\wedge$ fee), R(I, found), R(M, fixed), R(G, app), R(F, transfer)	Subteams
8	C(C, I, found $\wedge$fixed, data $\wedge$ fee), C(I, C, data $\wedge$ fee, found $\wedge$ fixed), C(M, I, paid, fixed), C(I, M, fixed, paid)	A_5=$A_4 \cup\{$O(C, eme $\wedge$ fee $\wedge$ data), O (I, found), O(M, fixed), O(G, app), O(F, transfer)$\}$	$\varnothing$	Onus

Table 3.1 shows a design path from Row 1 to Row 8.

Definition 3.8 *A design path* $\langle S_0, A_0, N_0\rangle \ldots \langle S_n, A_n, N_n\rangle$ *for requirements* R *is* sound *if and only if* $\langle S_n, A_n, N_n\rangle$ *is consistent and* $\langle S_n, A_n, N_n\rangle$ *satisfies* R.

In Table 3.1, the design path from Row 1 to Row 8 is sound. Soundness relies on Theorem 3.1 that establishes that design paths following the reductions below are sound.

Recall Eq. 3.3 states that $A_S, E, S \vdash R$. It captures the problem of specifying a protocol from requirements. The design process described above conforms to Eq. 3.3. S_n and A_n in the design path for R map to S and $A_S \cup E$, respectively. If the design path is sound, we obtain the relation $S_n, A_n \vdash R$.

3.3.4 Social Refinement Types

Below, we list social refinement types supported by Protos. (Here the set operators associate to the left and to reduce clutter we do not place assertions in quotations.)

Need refinement: Based on proposition refinement. The intuition is that if p refines to p', a need for p can be met by meeting a need for p'. For soundness, we must record the assumption that p refines to p'.

$$\langle S, A, N \cup \{\mathsf{R}(\tau, p)\}\rangle \hookrightarrow \langle S, A \cup \{p \hookrightarrow p'\}, N \setminus \{\mathsf{R}(\tau, p)\} \cup \{\mathsf{R}(\tau, p')\}\rangle$$

Notice that we write the needs set as $N \cup \{\mathsf{R}(\tau, p)\}$ to ensure that a need $\mathsf{R}(\tau, p)$ belongs to the needs set. In the resulting configuration, we remove $\mathsf{R}(\tau, p)$ from N to make sure it is not present in the resulting needs set and introduce the refined need $\mathsf{R}(\tau, p')$ explicitly.

Example 3.8 *Row 2 in Table 3.1 is obtained from Row 1 by refining* $\mathsf{R}(C, prepared)$ *into two other needs,* $\mathsf{R}(C, covered)$ *and* $\mathsf{R}(C, eme)$*, namely,accident coverage and emergency response, respectively.*

Commitment introduction i: If τ_1 has a need for q, τ_1 can address that need by obtaining a commitment from τ_0 to τ_1 whereby τ_0 commits to bringing about q provided p holds. Here, τ_1 takes on the need for p.

$$\langle S, A, N \cup \{\mathsf{R}(\tau_1, q)\}\rangle \hookrightarrow \langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, A, N \setminus \{\mathsf{R}(\tau_1, q)\} \cup \{\mathsf{R}(\tau_1, p)\}\rangle$$

Commitment introduction ii: If τ_1 has a need for q, τ_1 can address that need by obtaining a commitment from τ_0 to τ_1 whereby τ_0 commits to bringing about q provided p holds. In this case, we add an assumption that p will hold.

$$\langle S, A, N \cup \{\mathsf{R}(\tau_1, q)\}\rangle \hookrightarrow \langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, \{A \cup \{p\}\}, N \setminus \{\mathsf{R}(\tau_1, q)\}\rangle$$

Commitment refinement i: Based on the refinement of either or both of its antecedent and consequent. That is, if the antecedent or consequent of a commitment can be refined, then so can the commitment. As before, we record the refinements as assumptions.

$$\begin{aligned}\langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, A, N \cup \{\mathsf{R}(\tau_1, p)\}\rangle \hookrightarrow &\langle S \setminus \{\mathsf{C}(\tau_0, \tau_1, p, q)\} \cup \{\mathsf{C}(\tau_0, \tau_1, p', q')\},\\ &A \cup \{p \hookrightarrow p', q \hookrightarrow q'\},\\ &N \setminus \{\mathsf{R}(\tau_1, p)\} \cup \{\mathsf{R}(\tau_1, p')\}\rangle\end{aligned}$$

Notice that, the resulting commitment need not logically entail the original commitment. For example, a commitment to provide coffee for payment may be refined into a commitment to provide coffee for payment of Euros, though the second commitment is weaker than the original.

Example 3.9 *Row 4 in Table 3.1 is obtained by refining the commitments in Row 3. Specifically, buying a policy is refined into providing personal data and paying the fee, whereas providing coverage is refined into repairing the car. The commitments and needs in Row 4 reflect this refinement.*

Commitment refinement ii: Based on the refinement of either or both of the commitment's creditor and debtor.

$$\begin{aligned}\langle S \cup \{\mathsf{C}(\tau_0, \tau_1, p, q)\}, A, N \cup \{\mathsf{R}(\tau_1, p)\}\rangle \hookrightarrow \langle & S \setminus \{\mathsf{C}(\tau_0, \tau_1, p, q)\} \cup \{\mathsf{C}(\tau_{0'}, \tau_{1'}, p, q)\},\\ & A \cup \{\tau_{0'} \sqsubseteq \tau_0, \tau_{1'} \sqsubseteq \tau_1\},\\ & N \setminus \{\mathsf{R}(\tau_1, p)\} \cup \{\mathsf{R}(\tau_{1'}, p)\}\rangle\end{aligned}$$

Cyclic commitments: Given n teams ($n \geq 2$) where for each i, $0 \leq i < n$, team τ_i has a need for p_i, these teams can address their needs via (cyclic) commitments from τ_i to $\tau_{(i+1 \mod n)}$ whereby τ_i commits to bringing about $p_{(i+1 \mod n)}$ provided p_i holds. Reciprocal commitments are a special case of cyclic commitments when $n = 2$.

$$\begin{aligned}\langle S, A, N \cup \bigcup_i \{\mathsf{R}(\tau_i, p_i)\}\rangle \hookrightarrow \langle & S \cup \bigcup_i \{\mathsf{C}(\tau_i, \tau_{(i+1 \mod n)}, p_i, p_{(i+1 \mod n)})\}, A,\\ & N \setminus \bigcup_i \{\mathsf{R}(\tau_i, p_i)\} \cup \bigcup_i \{\mathsf{R}(\tau_{(i+1 \mod n)}, p_i)\}\rangle\end{aligned}$$

Example 3.10 *Row 3 in Table 3.1 is obtained from Row 2 by applying Cyclic Commitments. In Row 2, /Car Owner/ and /Insurer/ need coverage and sale of policies, respectively. To meet these needs, in Row 3, /Car Owner/ commits to buying a policy if /Insurer/ provides coverage for accidents and /Insurer/ commits to providing coverage if a policy is bought. Further, /Car Owner/ and /Insurer/ need to buy policy and provide coverage, respectively.*

Subteams: If a team τ has a need p, we can assign p_i to subteams τ_i; this is appropriate only because we assume that τ_i is a subteam of τ.

$$\begin{aligned}\langle S, A, N \cup \{\mathsf{R}(\tau, p)\}\rangle \hookrightarrow \langle & S, A \cup \{p \hookrightarrow \bigwedge_i p_i\} \cup \bigcup_i \{\tau_i \sqsubseteq \tau\},\\ & N \setminus \{\mathsf{R}(\tau, p)\} \cup \bigcup_i \{\mathsf{R}(\tau_i, p_i)\}\rangle\end{aligned}$$

Example 3.11 *Row 7 in Table 3.1 is obtained by applying Subteams. The payment need* $\mathsf{R}(I, paid)$ *is refined into approval and transaction, namely,* $\mathsf{R}(I, app)$ *and* $\mathsf{R}(I, transfer)$*. The manager and the finance department, which are the insurer's subteams, adopt these needs, that is,* $\mathsf{R}(G, app)$ *and* $\mathsf{R}(F, paid)$*.*

Onus: A team takes on the onus for some need locally; that is, it decides not to delegate that need to another team.

$$\langle S, A, N \cup \{\mathsf{R}(\tau, p)\}\rangle \hookrightarrow \langle S, A \cup \{\mathsf{O}(\tau, p)\}, N \setminus \{\mathsf{R}(\tau, p)\}\rangle$$

Example 3.12 *Row 8 is obtained by applying Onus: all stakeholders take on the onus for their remaining needs.*

Composition: Refinements compose in that, if part of a configuration is refined through a particular operation, so is an entire configuration through the same operation.

$$\frac{\langle S', A', N'\rangle \hookrightarrow \langle S'', A'', N''\rangle (S \cup S' \cup A \cup A') \nvdash \mathsf{false}}{\langle S \cup S', A \cup A', N \cup N'\rangle \hookrightarrow \langle S \cup S'', A \cup A'', N \setminus N' \cup N''\rangle}$$

3.3.5 Logic of Design Elements

For simplicity, we assume that a logic is available for reasoning about the various elements of a design configuration. In particular, this logic provides an inference relation, notated $\vdash$. The various elements of a design configuration are closed under $\vdash$, as specified by Table 3.2.

3.3.6 Example of Design Paths

Figure 3.2 illustrates the above definitions. The root is the requirement. Immediately below the root is the initial design configuration. Each of the leaves is a final configuration and satisfies the requirements. Each configuration is consistent. Each path is a well-formed design path and is sound for the requirements. The edge labels are informal descriptions.

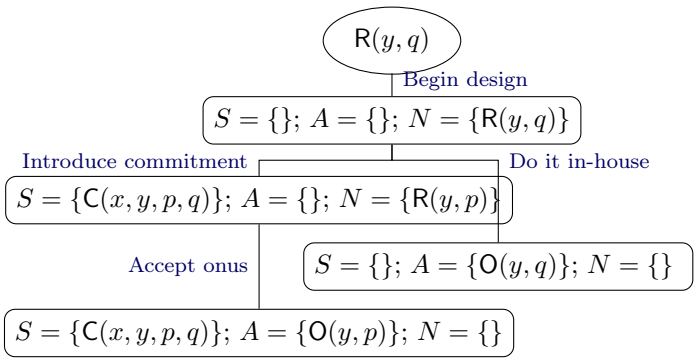


Figure 3.2: Paths through the design space.

Table 3.2: The underlying logic is propositional logic augmented with the following axioms pertaining to the symbols introduced in Protos.

Conflict means we cannot satisfy both	$\dfrac{pp \oplus q}{\neg q}$
A subteam's stronger need satisfies a team's need: and	$\dfrac{\mathsf{R}(\tau', p \wedge q)\tau' \sqsubseteq \tau}{\mathsf{R}(\tau, p)}$
A subteam's stronger need satisfies a team's need: or	$\dfrac{\mathsf{R}(\tau', p)\tau' \sqsubseteq \tau}{\mathsf{R}(\tau, p \vee q)}$
Subteams together cover needs that a team's need refines to	$\dfrac{\bigwedge_i \mathsf{R}(\tau^i, p_i)p \hookrightarrow \bigwedge_i p_i \bigwedge_i \tau^i \sqsubseteq \tau}{\mathsf{R}(\tau, p)}$
If a team takes on an onus, the corresponding need is covered	$\dfrac{\mathsf{O}(\tau, p)}{\mathsf{R}(\tau, p)}$
A conditional commitment along with its antecedent cover its consequent	$\dfrac{\mathsf{R}(\tau, p)\mathsf{C}(\tau', \tau, p, q)}{\mathsf{R}(\tau, q)}$
An unconditional commitment covers its consequent	$\dfrac{\mathsf{C}(\tau', \tau, \mathsf{true}, q)}{\mathsf{R}(\tau, q)}$

3.3.7 Soundness

We now establish the soundness of our formal model.

Theorem 3.1 (Soundness) *Let $P = \langle S_0, A_0, N_0\rangle \ldots \langle S_n, A_n, N_n\rangle$ be a design path for requirements R. Then P is sound.*

Proof sketch. Establish the invariant that $A \cup S \cup N \vdash R$ by structural induction: it holds for an initial configuration by construction and for each subsequent configuration by inspection of the reductions. In a final configuration, $N = \varnothing$: hence we have the result.

The above reductions do not consider conflict. The following reduction ensures that if τ_1 and τ_2 (could be the same team) have conflicting needs, at most one of those needs can be pursued.

Conflict introduction: Introduce an assumption of a conflict into a design configuration.

$$\langle S, A, N \cup \{\mathsf{R}(\tau_1, p), \mathsf{R}(\tau_2, q)\}\rangle \hookrightarrow \langle S, A \cup \{p \oplus q\}, N \setminus \{\mathsf{R}(\tau_2, q)\} \cup \{\mathsf{R}(\tau_1, p)\}\rangle$$

If we include the above reduction, Theorem 3.1 fails: although consistency is preserved, we can no longer establish that the original requirements are satisfied, because some may be dropped along the way. Incorporating conflict would lead us to formalize *satisficing* [35]. Then we may seek to establish that a design process satisfices the stated requirements.

Let us see how Protos would support design in the presence of conflicts. Imagine in Table 3.1 that *Car Owner* has an additional requirement concerning data privacy, that is, $\mathsf{R}(C, privacy)$. Conceptually, this requirement would conflict with the disclosure of personal data, that is, $\mathsf{R}(C, disclosure)$. In Protos, the stakeholder would introduce an assumption that $disclosure \oplus privacy$ via Conflict Introduction. Then, by Conflict from Table 3.2, we are guaranteed that one of the two propositions, i.e., one of the two requirements, cannot be met. In essence, the design process fails at this point.

3.4 Evaluation

We conducted a case study evaluation via a modeling session involving modelers other than the authors. To this end, we recruited eight modelers, all doctoral students in computing who are familiar with goal modeling, to participate in our modeling session. At the beginning of the session, we instructed the participants in Protos concepts, reductions, and design process.

We instructed the modelers to apply Protos to jointly create a specification of Kramer and Wolf's [20] case study of the London Ambulance System (LAS), which we described for the modelers. Kramer and Wolf's LAS scenario includes nine stakeholders, namely, the service consumer, call taker, call reviewer, LAS management, ambulance crew, call reviewer, radio operator, operator at ambulance station, dispatcher, and resource allocator. The requirement of the service consumer is to receive an ambulance when there is an incident. The call taker needs incident details to report the incident and the resource allocator requires the incident report. The radio has the requirement of the resource allocation information and the ambulance crew needs mobility instructions. For our modeling session, we merged the call taker and call reviewer stakeholders to ensure a one-to-one correspondence between the study participants and the stakeholders.

The study participants then designed the LAS STS starting from their respective requirements. Table 3.3 provides a partial design process with a few representative stakeholders where the call taker and the service consumer established a commitment and refined it prior to Step 1 of Table 3.3. Both stakeholders take the respective onuses which are added into the assumptions set in Step 1. In Step 2, the call taker and resource allocator apply commitment introduction *ii* and the call taker commits to the resource allocator to report information about the incident upon receiving such information. In Step 3 the call taker takes the onus for this commitment. Step 4 and 5 are compressed representations of the iterative design process that is similar to Step 2 and 3, where at each step commitment introduction *ii* is applied and then the oOnus is taken by the respective stakeholder.

Observations. The commitment network created by the participants following Protos superficially resembles the i^* strategic dependency diagram for LAS, as provided in [136]. However, the Protos modeling of the LAS offers distinct advantages over the i^* modeling of the LAS. One, the participants were able to collectively refine the commitments to make the interaction explicit and concrete. Two, the stakeholders were able to model conditional interactions via commitments, whereas an i^* dependency is not conditional. Three, the design reductions of Protos helped focus the design process; i^* lacks such a formalization. Four, removing the needs as the respective stakeholders took on the corresponding onuses helped the participants track the status of the design. Further, an empty needs set served as a clear stopping criterion for the design process; such a criterion does not exist in i^*.

Threats to validity. The reference document used for the LAS domains [20] includes a small number of stakeholders relative to other examples such as smart cities or a national health-care system. This somewhat narrow starting point and the participants' limited familiarity with the domain prevented them exploring designs that are not described in the reference document. Despite this, however, the participants elaborated details of interaction that are not present in the reference document by negotiating and using argumentation. The participants were collaborative and understanding to others' demands. In a real-life scenario one may expect more aggressive behavior from the stakeholders due to their conflict of interests that may even result in failure in the design process.

3.5 Chapter Summary

Protos is a novel RE process for sociotechnical systems that enables refining stakeholder requirements into commitment-based system specifications. Whereas other approaches are conceptually founded upon the notion of refining requirements into machines, Protos refines them into protocols. This brings modularity to the problem space: the problem of designing principals' software (agents) is related but separate from the problem of specifying protocols. Further, it demonstrates a generalization of Zave and Jackson's foundational characterization of RE.

We emphasize the point about the divergence of the requirements of the STS stakeholders from the requirements of the principals. Supporting this divergence is the key to innovation. Simply put, whatever the goals of the stakeholders might be we simply cannot install such goals in the decision models for the individual principals.

Protos opens up some interesting and important directions for further research. First, the emphasis on multiple stakeholders suggests a deeper study of conflict at design and run time, incorporating the notion of *satisficing* requirements, possibly in relation to notions such as social welfare of the stakeholders and bringing to bear techniques such as argumentation [137] for determining which of the conflictin requirements to satisfy and which to ignore. Second, the sociotechnical setting opens up challenges of vagueness, inconsistency, and defeasibility of requirements [36]. Third, we would need to explore modeling concepts geared for requirements in diverse STS settings, e.g., with respect to the

Table 3.3: A portion of the design process during the modeling session on the London Ambulance System

Step	Specification	Assumptions	Needs	Refinement
1	$\mathsf{C}(ct, sc, address \wedge status, incidentTaken)$	$A_1 = \{incidentReported \hookrightarrow address \wedge status, O(sc, address \wedge status), O(ct, incidentTaken)\}$	$\mathsf{R}(sc, ambReceived)$, $\mathsf{R}(ac, mobilityInfoSent)$, $\mathsf{R}(ro, resourceAllocated)$, $\mathsf{R}(ra, infoReported)$	Onus
2	$\mathsf{C}(ct, sc, address \wedge status, incidentTaken)$ $\mathsf{C}(ct, ra, incidentTaken, infoReported)$	A_1	$\mathsf{R}(sc, ambReceived)$, $\mathsf{R}(ac, mobilityInfoSent)$, $\mathsf{R}(ro, resourceAllocated)$, $\mathsf{R}(ct, infoReported)$	Commitment introduction *ii*
3	$\mathsf{C}(ct, sc, address \wedge status, incidentTaken)$ $\mathsf{C}(ct, ra, incidentTaken, infoReported)$	$A_2 = A_1 \cup \{O(ct, infoReported)\}$	$\mathsf{R}(sc, ambReceived)$, $\mathsf{R}(ac, mobilityInfoSent)$, $\mathsf{R}(ro, resourceAllocated)$	Onus
4	$\mathsf{C}(ct, sc, address \wedge status, incidentTaken)$ $\mathsf{C}(ct, ra, incidentTaken, infoReported)$ $\mathsf{C}(ra, ro, infoReported, resourceAllocated)$ $\mathsf{C}(ro, ac, resourceAllocated, mobilityInfoSent)$ $\mathsf{C}(ac, sc, mobilityInfoSent, ambReceived)$	A_2	$\mathsf{R}(ac, ambReceived)$, $\mathsf{R}(ro, mobilityInfoSent)$, $\mathsf{R}(ra, resourceAllocated)$	Commitment introduction *ii*
5	$\mathsf{C}(ct, sc, address \wedge status, incidentTaken)$ $\mathsf{C}(ct, ra, incidentTaken, infoReported)$ $\mathsf{C}(ra, ro, infoReported, resourceAllocated)$ $\mathsf{C}(ro, ac, resourceAllocated, mobilityInfoSent)$ $\mathsf{C}(ac, sc, mobilityInfoSent, ambReceived)$	$A_3 = A_2 \cup \{O(ac, ambReceived)$, $O(ro, mobilityInfoSent)$, $O(ra, resourceAllocated)\}$	$\varnothing$	Onus

sc: service consumer, ct: call taker, ra: resource allocator, ro: radio operator, ac: ambulance crew

normative relationships [138], and for which we can establish results such as completeness. Fourth, it would be crucial to develop a methodology and tools that support the Protos design process.

Chapter 4

Exploring Sociotechnical System Design Space

The previous chapter introduces Protos, a requirements engineering approach to support the design process of sociotechnical systems. The output of the design process is a specification of the sociotechnical system in terms of system interactions, domain assumptions made by systems, and finalized responsibilities of the systems to realize at run-time.

This chapter explores the alternative solutions in the design space and constructs a plan to execute to satisfy the selected solution. Section 4.1 presents a goal–oriented requirements modeling language to capture the requirements of the systems, systems presented as actors in line with i^* and Tropos frameworks. Section 4.2 defines the actions to be taken as part of a plan that fulfill system requirements. A plan is a sequence of actions through which participants requirements are satisfied [139]. We map the design problem into a planning problem as previously demonstrated by Gans et al. [140]. We devise plans that preserve the stability of the sociotechnical system by respecting the interactions agreed at design time, and minimize the cost. The details of implementation in PDDL are presented in Section 4.3. The results of self–evaluation of the visual notation and a brief analysis of the scalability of the approach is included in Section 4.4.

Acknowledgment This chapter is based on our work presented in [141].

4.1 *DEST*: A Modeling Language for Designing Sociotechnical Systems

DEST (*DE*signing *S*ociotechnical sys*T*ems) is a modeling language for designing sociotechnical systems. *DEST* focuses on the social components and their interactions; we reify technical systems through the social entity (their developer, owner, or user) that is liable for the effects of their interactions with other entities. We model each component as an actor, an autonomous entity that aims to satisfy its own requirements either by its own means or through social interactions with other actors.

DEST is based on and extends *i.* the i^* modeling framework, from which we take

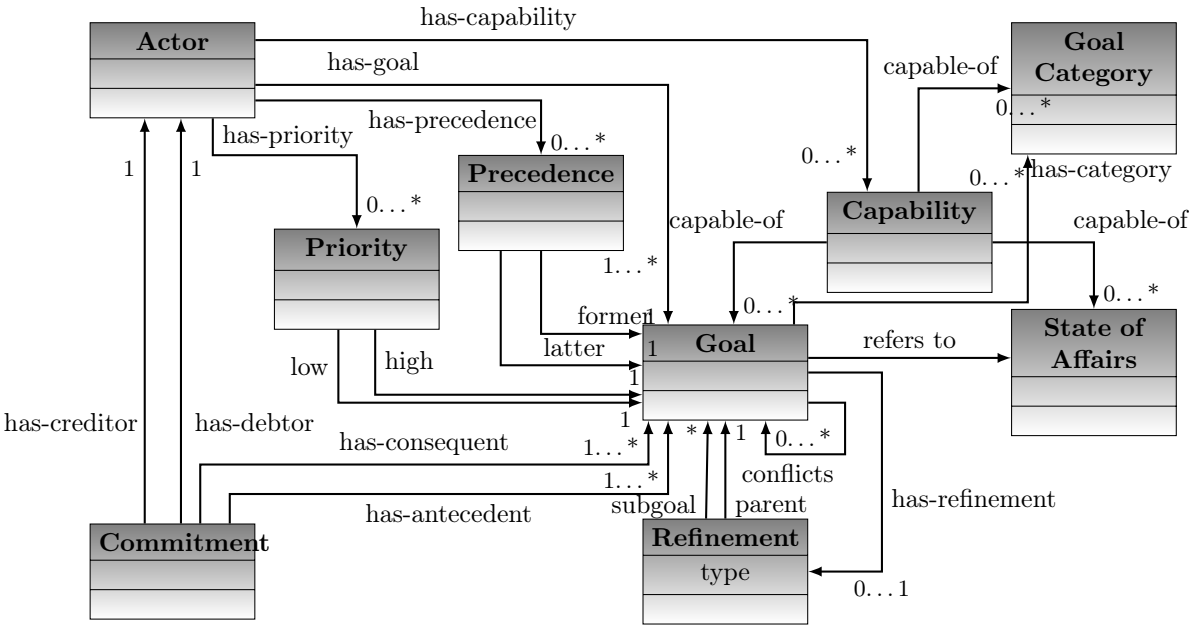


Figure 4.1: Meta-model of the *DEST* modeling language

the concepts of *actor*, *goal*, and *refinement*; *ii.* social *commitments* to represent the social interactions among the *actors*; and *iii.* advanced requirements, relations such as *priority* and *precedence* [36], [142]. Together, these elements support expressive modeling and reasoning for requirements and interactions during sociotechnical system design and evolution.

Figure 4.1 presents the meta-model of *DEST*. We explain the three fundamental concepts—goals, actors, and commitments—in the following subsections. Figure 4.2 illustrates the graphical syntax of *DEST* for the travel reimbursement sociotechnical system, whose scenario is constructed by us based on the system in practice at the University of Trento in 2013.

4.1.1 Requirements

We represent stakeholder requirements as goals. In addition we introduce priority and precedence relations between goals, so as to enable a fine-grained specification of the relative importance and urgency of the goals. Specifically, *DEST* supports the following elements to represent requirements:

Goal models a desired state-of-affairs that an actor wants the sociotechnical system-to-be to achieve. In a sociotechnical system the behavior of actors is determined by their respective goals.

Precedence: a type of relation between goals that specifies that one goal is to be satisfied/carried out before another. Precedence relation represent actors' temporal

constraints on the goals. For example, a project fund manager may require that a conference paper should be accepted to be published before its authors submit a travel authorization request.

Priority: a type of a relation between goals that indicates that one goal has higher priority than another. A student may prioritize spending its travel budget on plane tickets rather than on accommodation.

Conflict: a goal may conflict with another, that is, they cannot both be satisfied in the same sociotechnical system configuration.

Refinement: A goal may be AND/OR-refined to subgoals that are easier to fulfill than their parent. For example, a goal could be AND-refined into two subgoals for which there are actors capable of fulfilling them.

Capability: the actors ability to achieve a goal, a category of goals or a state of affairs.

Goal Category: each goal may belong in a category that indicates the type of capability needed to fulfill it. For example, in the travel reimbursement sociotechnical system, a goal category could be "authorization", to aggregate all goals that can be fulfilled through authorization the requests. An actor that has the capability of satisfying a certain category, does not need to declare his capability for each goal that belongs to that category.

Figure 4.2 describes a model built by using the concepts described above. For example, *'trip booked'* is a goal that is OR-refined into two other goals: *'booked by student' 'booked via travel agent'*, which implies that the child goals are easier than to satisfy the parent goal. In the model, a precedence relation is defined between the goals *'trip planned'* and *'paid by student'*, which states that the former should be satisfied before the latter. The goal *'expensive accommodation'* has priority over the goal *'expensive tickets'* so when taking actions in a plan towards satisfying the goals, the former goal has a higher priority. There is a conflict relation defined between the *'cash payment'* and the *'expensive tickets'* goals, therefore only one of them could be satisfied by the sociotechnical system.

4.1.2 Actors

An actor is an autonomous component of a sociotechnical system. An actor could be a human such as a student or a project fund manager, a social entity such as an airline or a hotel, or a technical system such as the information system used within a department. The following relations relate actors to their requirements:

has-goal: this denotes that an actor has a requirement to be satisfied by the sociotechnical system. In *DEST*, requirements consist of goals and relations such as precedence, priority, and conflict. A solution to the design problem should respect these relations and satisfy the goals of actors.

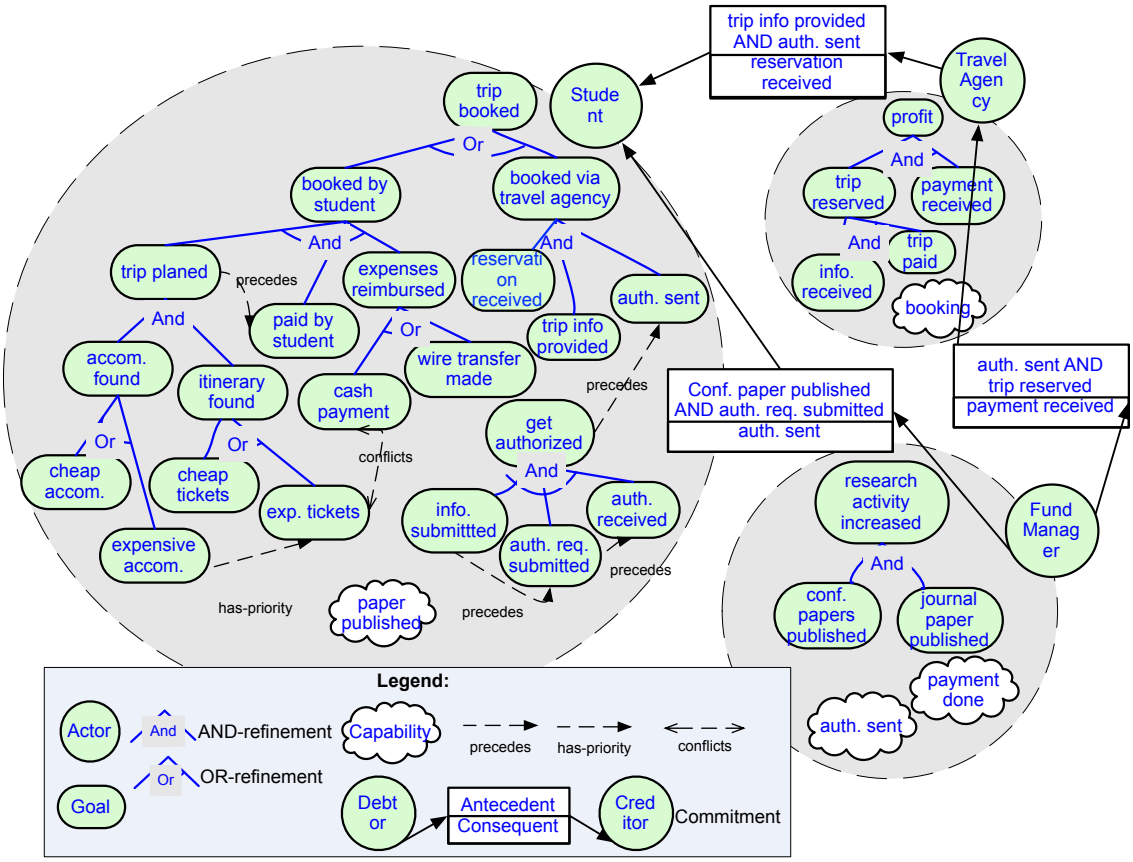


Figure 4.2: *DEST* model of the travel reimbursement sociotechnical system

is-capable-of: an actor is capable of (fulfilling) a goal if it can do so on its own, without any help from other actors. I an actor has some goals that she is not capable of satisfying, the actor must interact with other actors to get the goal satisfied by some other actor who is capable of satisfying the goal. The actor who has the capability to satisfy the goal may ask for the satisfaction of some of her own goals, as often in case of real life, which initiates a reciprocal social interaction between actors with various capabilities. There may be alternative solutions for a sociotechnical system where an actor still interacts with others to get a goal satisfy of which she is capable. For example, a travel agent may choose to use some intermediary travel company for bookings due to its lower cost although she has the capability of booking tickets and hotel rooms.

is-capable-of-category: an actor is capable of satisfying any goal that belongs in a particular category.

There are three actors in the model presented in Figure 4.2: student, project fund manager, and travel agent. Each actor has its requirements represented as goals, priorities,

and precedence relations within its actor boundary. Capabilities such as 'booking' and 'payment done' are shown in cloud shaped nodes within the boundaries of the actor who is capable of them.

4.1.3 Commitments

A commitment represents a contractual relation between two actors as they socially interact. Two actors get engaged in a commitment whenever their capabilities are insufficient for satisfying their requirements, or when interacting with others is more convenient than exercising an internal capability. A commitment is defined in terms of four relations:

has-debtor: the actor who is responsible for satisfying the consequent of the commitment. A debtor participates in the commitment because there is at least one goal listed in the antecedent that he wants to achieve. The debtor should be capable of satisfying all goals in the consequent.

has-creditor: the actor who is the beneficiary of the commitment. The creditor of the commitment is interested in the satisfaction of at least one goal listed in the consequent. The creditor should ensure that antecedent goals are satisfied, but need not be the one to do so.

has-antecedent: a set of goals whose satisfaction is a precondition for the commitment to be fulfilled.

has-consequent: a set of goals that the debtor is obliged to fulfill.

Figure 4.2 includes three commitments represented as split rectangles. For example, the commitment between the travel agent and student, where travel agent is the debtor that commits to book the trip if the student provides needed information about the trip and sends authorization for it.

4.2 Designing a Sociotechnical System

The modeling language introduced in Section 4.1 helps abstracting the requirements of actors in a sociotechnical system, relations among these requirements, actor capabilities and social interactions between actors that help fulfilling the requirements. Analysis of this model consists of exploring the space of alternatives for fulfilling requirements: actor root goals, while respecting precedence, priority, and conflict relations. Alternative solutions may follow different paths to satisfy the root goals. Differences in a path includes following different OR-refinement branches, using an actor's own capabilities to satisfy leaf goals, and activating one of the commitments described in the model. A sociotechnical system deploys one of the alternative solutions at run time as its *configuration*.

Definition 4.1 (Configuration) *A configuration* Cfg *of a DEST model* M *is a 3-tuple* $\langle \mathsf{G}, \mathsf{Cap}, \mathsf{Com} \rangle$*, where* G *is the set of (sub-)goals that the actors in the sociotechnical system are adopting,* Cap *is the set of capabilities that the actors are exploiting, and* Com *is the set of commitments that the actors have established.* G*,* Cap*, and* Com *are subsets of the set of goals, capabilities, and commitments in* M*, respectively.*

A configuration is *empty* at the beginning. Actors state their goals, their capabilities and relations among their goals as well possible refinements and commitments. A *valid* configuration includes the selected set of commitments and refinements that satisfies the top-level goals of the actors while respecting the specified relations among goals.

We now present the design process whereby a sociotechnical system is designed using *DEST*:

Definition 4.2 (Design process) *Let* M *be a DEST model that represents the space of alternatives of a sociotechnical system, and let* Cfg *be a valid configuration of* M*. A design process* $\mathsf{D}_{\mathsf{Cfg},\mathsf{M}}$ *is a list of actions* $(\mathsf{Act}_1, \ldots, \mathsf{Act}_n)$ *that, if correctly executed starting from an empty configuration, leads to the configuration* Cfg*.*

The actions that construct a design process correspond to the execution of one of the following *action types* on an element of a *DEST* model:

Adopt goal: an actor can adopt a new goal when he doesn't have the goal, and the goal is not already satisfied. As a result of this action, the actor intends to achieve (*has*) the goal.

Satisfy goal via capability: an actor can satisfy a goal if

- the actor has the goal,
- the goal has not been satisfied,
- no conflicting goals are satisfied,
- all goals that shall precede it are satisfied, and
- the actor is either capable of satisfying the goal itself or the category of the goal.

As a result, the goal becomes satisfied, and the actor does no longer have the goal.

Refine goal: an actor may AND/OR-refine a goal if

- the actor has the goal,
- the goal has not been satisfied, and
- there is a refinement for that goal in the actor's goal model.

As a result of the action, the actor does no longer have the parent goal, but he has the subgoals.

Create commitment: An actor may create a commitment playing the role of debtor if there is at least one goal in the antecedent that the actor wants to satisfy.

Accept commitment: An actor accepts a commitment as creditor if the actor wants to satisfy at least one of the consequent goals.

As a result, the actor drops its goal(s) listed in the consequent, and adds the goals in the antecedent.

Detach commitment: The debtor of the commitment detaches it when all goals in the antecedent are satisfied. Such condition binds the debtor to satisfy the consequent, so he adopts the goals listed in the consequent.

Discharge commitment: The creditor of the commitment discharges the commitment when all the goals in the consequent are satisfied.

All *action types* defined above are used at design time and executed on goals and commitments. Actors may *adopt* new goals during the design process. After adopting a goal, an actor may either satisfy the goal, or further *refine* it into other goals. If capable, the actor may choose to *satisfy the goal via its own capabilities*. If the actor is not capable of satisfying the goal or does not prefer to do so due to some constraints such as budget, the actor *creates a commitment* with another actor to get its goal satisfied. Creating a commitment is to activate the respective commitment described in the model and a declaration to the creditor of the commitment to satisfy the goals listed in the consequent of the commitment one of which may be adopted by the creditor. In return, the debtor actor expects the goals listed in the antecedent to be satisfied. If the creditor is interested in the commitment, that is, if there are some of its goals listed in the consequent of the commitment, the creditor *accepts* the commitment. Once the goals in the antecedent are satisfied, the commitment becomes *detached*. When the debtor actor satisfies the goals in the consequent the commitment is *discharged*. The list of actions that operate on commitments do not include two commitment operations that are defined in [18] canceling a commitment and violating a commitment. We deliberately omit these operations as they are deviations from the successful execution of commitments, which may occur at run run-time as exceptions but are not used at design time.

The output of the design process is a set of actions of whose execution leads to a valid configuration for the sociotechnical system. The precedes relation imposes constraints on the satisfaction order of goals; thus, the order of the actions in a design process is relevant and should be used as a guideline for the implementation of the sociotechnical system.

Figure 4.3 we provide a partial enactment of the design process for our example sociotechnical system in Figure 4.2. The plan sets up the interaction between the *student* and the *travel agency*. In this scenario, the *travel agency* wants to make profit by selling trips, and needs to receive necessary information for the trip as well as reimbursement for the trip (hotel and tickets). The *travel agency* is capable of making payments and booking trips. In the following steps, first, the *student* adopts and refines a goal (Steps 1-3). Then the *travel agency* creates a commitment which potentially matches its goals and the

1. *Student* **adopts** *'trip booked'*.
2. *Student* **OR-refines** *'trip booked'* to *'booked by student'* and *'booked via travel agency'*.
3. *Student* **AND-refines** *'booked via travel agency'* to *'trip info provided'*, *'authorization sent'*, and *'reservation received'*.
4. *Travel Agency* **creates commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.
5. *Student* **accepts commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.
6. *Student* **satisfies** *'trip info provided'*.
7. *Student* **satisfies** *'authorization sent'*.
8. *Travel Agency* **detaches commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.
9. *Travel Agency* **satisfies** *'reservation received'*.
10. *Student* **discharges commitment** *'C(Travel Agency, Student, trip info. provided AND auth. sent, reservation received)'*.

Figure 4.3: A partial design plan for the model described in Figure 4.2

capabilities to those of the *student*'s (Step 4). The *student* accepts the commitment (Step 5), and the two successfully A possible sequence of run-time interaction that instantiates the design plan is also shown (Steps 6-10). Among these steps, detaching (Step 8) and discharging a commitment (Step 10) are necessary for one actor to acknowledge that the other actor has, indeed, satisfied the antecedent (Step 8) and the consequent (Step 10).

It is generally the case that there are alternative ways to satisfy requirements for a sociotechnical system. This happens when a goal is OR-refined, when several actors have capabilities for the same goal, and when multiple commitments are possible for fulfilling a goal.

The problem of identifying configurations that fulfill a given set of requirements can be reduced to an automated planning problem [143], where a tool is employed to search the encoding of a *DEST* model for feasible design plans that respect the constraints. The planner returns a list of actions (a plan) to be followed at run time that results in the satisfaction of the requirements of the actors.

In order to identify the best plan, among many possible ones, we assign a *cost* value

to each action, which corresponds to the effort for an actor to execute that action. There are multiple ways of expressing cost: either a cost can be assigned to each action (e.g., to adopt(g_1)), or a standard value is assigned to action types, or simply, all actions are assumed to have unit cost. The best plan is the one that has minimal cost, among all possible plans. An alternative way to quantify the results of the actions is to add a utility to each action, in this case the best plan maximizes the total utility. How to assign these values are beyond the scope of this thesis, one of the existing methods from the literature could be applied for this task. Karlsson and Ryan [144] adopts a cost based approach for prioritizing requirements. Regan et al. [145] provide a method on eliciting reward information and use regret-reduction to choose suitable solutions. Their approach could be broadened to utility elicitation. In the next section we show how we map the identification of the best plan into an input for an off-the-shelf planner.

4.3 Implementation

The field of AI planning has found a number of applications in various areas such as multiagent systems and robotics. A planning problem includes the initial state of the world, the desired of the world, and the possible actions that can be taken throughout the plan [143]. Using the *DEST* language described in Section 4.1 and the actions in Section 4.2, we encode the problem of building a network of interactions for the actors in a sociotechnical system to a planning problem that satisfies actor requirements while minimizing cost. We encode the problem using the Planning Domain Definition Language (PDDL) [21], the de-facto standard input format for planners. We use PDDL v3.0 [146], which supports preferences and soft constraints that we need to implement the *has-priority* relation.

An of-the-shelf PDDL planner has two inputs: the domain description and the problem definition. We map the *DEST* concepts defined in Figure 4.1 into PDDL constructs in the domain description file. Mappings of the actions in Section 4.2 are also added to the domain description file. This file can be re-used for various problem definitions, each consisting of the model (instances of the concepts), the initial state of the sociotechnical system, metrics for the plan, and the instances of the requirements.

In the domain description file, we state the requirements for the planner, which are features of PDDL that the planner provides. For our purposes, we use `typing` to abbreviate the type declarations for the multiple objects of the same type, `adl` for using disjunctions, quantifiers in preconditions, and conditional effects, `:constraints` for the trajectory constraints and `numeric-fluents` to model costs. Among the concepts presented in Section 4.1 we map *Actor*, *Goal*, *Goal Category*, and *Commitment* into object types in PDDL, so `goal(trip-booked)` is translated that *trip-booked* represents an instance of the *goal* class in *DEST*. Since there is a one-to-one correspondence between *DEST* and PDDL for these four classes, we directly map their relations into namesake PDDL predicates. For example, a *has-goal* relation between *actor* `a` and *goal* `g` is mapped to `has-goal(?a - actor, ?g - goal)`. Note that we provide a template in the domain

Table 4.1: List of PDDL Predicates and Fluents

Predicates	**Predicates**	**Fluents**
(is-satisfied ?g - goal)	(and-ref ?g ?g1 - goal)	(actor-cost ?a - actor)
(or-ref ?g ?g1 - goal)	(has-goal ?a - actor ?g - goal)	(has-goal-cost ?a - actor ?g - goal)
(precedes ?g ?g1 - goal)	(has-priority ?g ?g1 - goal)	(total-cost)
(has-category ?g ?gc - gcat)	(conflicts ?g ?g1 - goal)	
(is-capable ?a - actor ?g - goal)	(is-capable-cat ?a - actor ?g - goal)	
(has-debtor ?c - comm ?a - actor)	(has-creditor ?c - comm ?a - actor)	
(has-ant ?c - comm ?g - goal)	(has-cons ?c - comm ?g - goal)	
(is-created ?c - comm)	(is-discharged ?c - comm)	
(is-detached ?c - comm)		

description file, and the question marks mean that these objects are variables for that predicate.

Refinement, *conflicts*, *precedence*, and *has-priority* are mapped through their relations with the four classes mentioned above into PDDL, rather than introducing namesake object types. We implement `and-ref`, `or-ref`, and `conflicts` to capture *refined-to* relation (specifying the refinement type) and *has-conflict* relation, respectively. For the *priority* class, we combine *high-* and *low-priority* relations into `has-priority`. Similarly, we define the `precedes` predicate for the *precedence* classes. Other than these predicates, we define auxiliary predicates to check the state of a goal or a commitment, such as `is-satisfied` for a goal and `is-created` for a commitment. The full list of the predicates,together with the object types and orders are given in the first two columns of Table 4.1.

In our implementation of the actions and the *DEST* language, in order to find an optimal plan, we use cost as a metric. The individual efforts performed by each actor are aggregated via in the `actor-cost` fluent. When an actor performs an action, the actor's cost increases by that action's cost. The cost of satisfying a goal by a particular action is kept in `has-goal-cost`. Finally, the overall cost of the plan is kept in `total-cost`. Those values are initiated in the problem definition files before the planning starts. All the fluents that we use are listed in the third column of Table 4.1. Various cost aggregation functions or other metrics could be defined and implemented in the PDDL language and we leave exploring the details of plan optimization as a future work. The implementation of the priority relation is also part of the future work where the priority violence is possible but not desired.

We define a PDDL action for each action in Section 4.2. Also, to determine the satisfaction of a goal through AND/OR-refinement, we implement the corresponding actions that satisfies the goals instead of relying on the PDDL's derived axioms due to performance issues. A PDDL action has three components: parameters, precondition, and effect. Below we provide the PDDL code for the `satisfy-goal` action: both preconditions (the actor has the goal, etc.) and effects (the goal becomes satisfied, etc.) correspond to those described in Section 4.2. Finally, the costs are increased by

- a variable cost for satisfying the goal `g` by the actor `a`, and
- a fixed cost for satisfying a goal.

```
;Actor satisfies a goal
(:action satisfy-goal
 :parameters(?a - actor ?g - goal)
 :precondition(and (has-goal ?a ?g)
   (not (is-satisfied ?g))
   (or (is-capable ?a ?g)
    (exists(?cat - gcat)
     (and (has-category ?g ?gcat)
     (is-capable-cat ?a ?gcat))))
   (not(exist(?cg - goal)
       (and (or (conflicts?g ?cg)
             (conflicts ?cg ?g))
             (is-satisfied ?cg))))
   (forall(?pc - goal)
         (imply(precedes ?a ?pc ?g)
         (is-satisfied ?pc))))
 :effect (and (is-satisfied ?g)
         (not (has-goal ?a ?g))
         ((total-cost) += (has-goal-cost ?a ?g) + <num>)
         ((actor-cost ?a) += (has-goal-cost ?a ?g) + <num>)))
```

Figure 4.4: PDDL action for an actor satisfying the goal of which it is capable

The second part of the implementation is the problem definition, where the instances of the objects and relations are defined and the initial values of the fluents are assigned, such as `(:init (=(total-cost) 0 ))`. Also, the goals that are adopted by the *actors* are stated in the `(:goal ...)` part to tell the planner to satisfy these particular *goals*. In this implementation we opt for a simple metric, the minimal total cost, which is encoded as `(:metric minimize (total-cost))`.

In order to handle evolution, we implement remove-actions: for each action `a` implemented, remove-action-`a` takes back the effects of the action `a`(except the cost). Moreover, in the case of evolution, the initial state is a definition of the current configuration (as opposed to the empty state for the initial design case).

4.4 Evaluation

Visual scalability. Figure 4.2 illustrates our running travel reimbursement sociotechnical system example in *DEST*. We deliberately choose a syntax similar to that of i^*/Tropos

to foster adoption from experts in the area of goal-oriented requirements engineering. The visual notation of *DEST* follows the design principles provided in [15], [16].

Semiotic clarity. There is a one-to-one correspondence between the symbols used in the visual notation and their referent concepts: each symbol is only used to represent a single concept from the language and each language concept is represent by only one symbol.

Perceptual discriminability. Concepts, such as actor, goal, capability and commitments are highly distinguishable as their respective symbols have clearly different shapes from different shape families. Goals belong in the oval family, whereas actors are from the circles. We introduced the new node shape for commitment, which is represented by split rectangles so the shape does not only belong to a different shape family from other symbols used in *DEST* visual syntax but it also distinctly different from the rectangle node used in i^* which represent resources since it is split in two. The other introduced node is the cloud-shaped node that corresponds to 'can-satisfy' relation. By using shapes from distinct shape families we ensure the visual distance between the node symbols are at least one.

To further increase the visual distance, positional cues are used. Goal and capability nodes are placed within the boundaries of an actor, actor name nodes are placed on the actor boundary and the commitments are placed between the actors. Since the goal and capability symbols are spatially close to each other, we use color as the secondary dimension for the visual distance. As light green is traditionally used for the goal nodes, we use white for the capability nodes. We depart from the i^* syntax for the actor names and color them in blush to increase the visual distance from goal nodes both of which are light green in the i^* syntax.

Complexity management. We omit representation of goal categories not to overload the visual syntax. Furthermore, in order to distinguish the goals that an actor is capable of satisfying himself, we place small cloud shape nodes on them. So, with the shape, size, color, horizontal and vertical positions being different, the nodes have a visual distance of five from each other.

The visual distance of the edges used in the notation is also greater than one. Edges to and from commitments are solid lines with an arrowhead filled with black. Since the direction of the edges naturally conveys the meaning for the debtor and the creditor we do not use textual labels on these edges to keep the notation light for the human eye. We follow the traditional representation for the refinements, which is solid blue lines, and the type label is positioned close to the parent node. Among the edges that represent the precedes, has-priority and conflicts, the edge that represents conflicts relations differentiates from the other since it is i bi-directional dashed ii has simple arrowhead, has-priority edge has a dotted line whereas precedes edge has the dashed style. So at least a visual distance of two is ensured among the edge types with the variables texture (line style), shape (arrowhead), and color. Textual labels also ensure that the three edge types are distinguishable from each other.

Overall, symbols used in the *DEST* visual notation have at least visual distance one from the other symbols. In terms of shape, color, and position the notation stays well

within the boundaries of the cognitive capacities, that is number of perceptible chunks as there are 3 relative positions with respect to the actor boundaries (in, out, on) and 2 with respect to the refinements (label is closer to the parent and at the intersection of the edges) so a total of 5 where the maximum value is ten to fifteen. Also there are 4 colors used in the notation where the maximum capacity for the color variable is seven to ten.

Scalability with respect to the size of the problem. To test the scalability of our approach with respect to the size of the problem, we manually create a *DEST* model that consists of three actors, four commitments, two OR-refinements, five AND-refinements, one precedence, one conflict and 20 goals. We then automatically generate 30 test models by replicating the original model. So the final model consists of 90 actors, 600 goals, 120 commitments, 60 OR-refinements, 150 AND-refinements, 30 precedence and 30 conflicts. We use sgplan version 5.22[1] as our choice of of-the-shelf planner and run our test files together with our domain file which includes our domain declaration in PDDL. The experiment is run on an Ubuntu 12.04 virtual machine with 2 GB memory hosted on a Mac OSX with 4 GB memory and a 2.5 GHz Intel Core i5 processor. The results of the experiments are summarized in Figure4.5 where y-axis shows the time in seconds to find a solution (plan) and x-axis shows the number of replicas in the test model. The first phase to find a solution is to read the domain description and problem files and to construct the model. The time spent for the first phase is indicated as 'parsing time' and shown by bright green line with cross in Figure4.5. Parsing time increases rapidly beyond 1000 model elements. The second phase is the planning phase in which the planner search for solutions in the search space, that is, the constructed model in the first phase. Planning time is shown by the blue line with squares in Figure4.5. The planning time constitutes a less significant of the total time spent. Parsing time has a more rapid increase than the planning time. The reported values highly depends on the performance of the chosen planner.

The results are promising, for we have artificially synthesized extra-large models that are way bigger than those that are typically created by modelers. However, more work is required to test scalability when increasing other dimensions of the model, such as complexity and connectivity.

4.5 Chapter Summary

We have presented a framework that supports exploring alternative plans for building and evolving a sociotechnical system. Our framework is model-driven, and uses our proposed *DEST* language for representing actor requirements in a sociotechnical system and the space of alternative designs.

In addition to introducing *DEST*, we have proposed techniques for building a network of interaction that fulfills participant requirements from scratch, and also to re-design such a network in case of evolution. For implementation, we have encoded plan generation to a problem a native language of an automated planner.

[1]http://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/

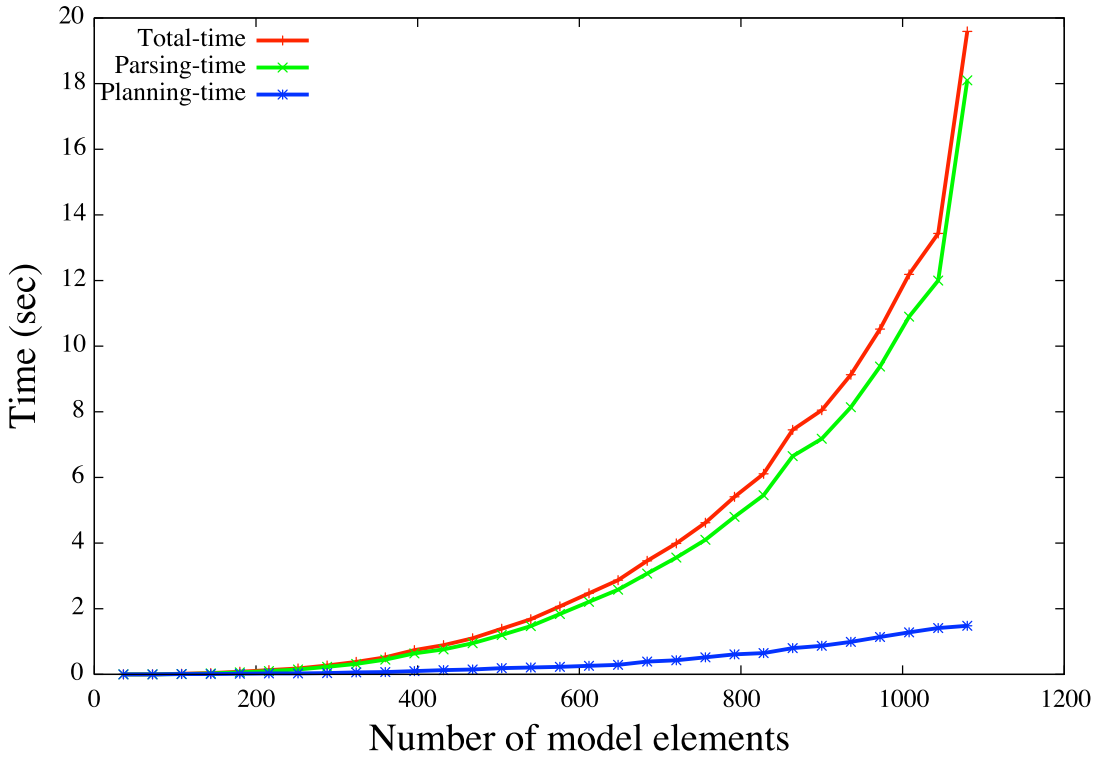


Figure 4.5: The results of the scalability experiments w.r.t model size

DEST visual notation has similar limitations in terms of readability as the other goal-oriented approaches do. As the *DEST* models become bigger, it is hard for humans to read the model as a whole. A dedicated editor with different types of views as in could be a solution to this problem. Eliciting cost values for single actions is a challenge for practical purposes, yet even more elaborate cost schemes, and the effect of actions on other actions are other future challenges that are need to be addressed. Finally, to obtain the quantitative effect of additions and removals during the evolution is another challenge to overcome. Trying to satisfice the requirements as suggested in [35] may help decreasing the complexity.

Chapter 5

The Next Release Problem

Evolution for software products is managed through releases. The next release for a given product is determined by gathering candidate new requirements over a time period (say, six months) and then selecting which of these are going to be implemented, taking into account logical constraints (such as mutual exclusion and precedence), as well as quality considerations such as minimize costs, maximize customer value, and the likes. Following the literature [22], we refer to this as the *next release problem* (NRP) for a software product.

The NRP has already been studied in the literature. In particular, [22] formalizes the problem as a single-objective optimization problem that is shown to be NP-hard. More recently, [23] formulates the problem as a multi-objective optimization problem, and tackles it through a combination of genetic algorithms. However, in many situations the qualities relative to which objective functions need to be defined are poorly understood and therefore hard to quantify. Software cost, for example, is notoriously hard to estimate even approximately. Customer satisfaction, as a competing quality, is even harder to boil down to numerical scores. Modern software development methodologies, such as agile methodologies often rely on relative rather than absolute estimations. Moreover, existing approaches represent requirements as flat collections of functions that do not take into account the hierarchical nature of requirements, therefore lose valuable information about the relations between requirements, possible alternatives, and even the raison d'être of some requirements.

We are interested in revisiting the NRP, framing it in terms of goal models where requirements are hierarchically structured and inter-dependent (conflicting/synergistic). The space of alternatives is then explored to discover pareto-optimal solutions by using combined qualitative and quantitative reasoning techniques founded on automated reasoning technologies, notably Satisfiability and Optimization Modulo Theories (SMT/OMT).

The main contributions of this chapter are:

- A goal-oriented framework for reasoning with NRPs that supports both logical and quantitative reasoning,
- An implemented tool that supports both modeling and reasoning with NPRs;

- Experimental results that demonstrate that the proposed framework scales to realistic-size problems.

5.1 Research Baseline

Goal Models. Goal models have been used to represent requirements for more than 20 years. Goals, softgoals, and tasks are the main concepts used in goal modeling frameworks such as KAOS [40], NFR [33], /i/* [13], and Tropos [49]. Goals/softgoals can be refined through AND/OR refinements into more concrete ones to render a hierarchical structure to goal models. Goals/softgoals can also be inter-related through conflict/synergy links.

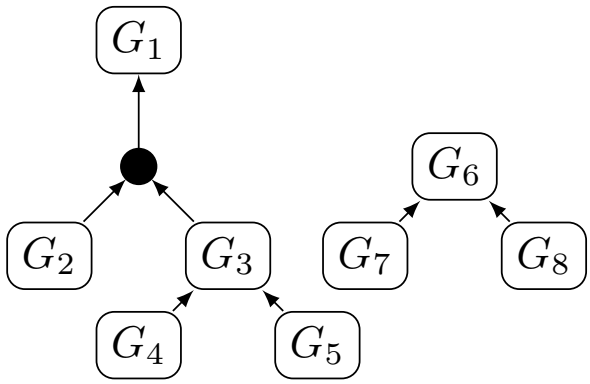


Figure 5.1: A simple goal model

Figure 5.1 presents a simple goal model. The black filled circle denotes the single AND-refinement in this example. A solution to a goal model consists of a set of leaf nodes which together satisfy all root goals and optimize an objective function. Traditionally, the objective is to minimize the number of leaf goals in the solution [147]. There are four solutions, including {G2, G4, G7}, {G2, G5, G7}, {G2, G4, G8}, {G2, G5, G8} in Figure 5.1 for this objective.

Satisfiability and Optimization Modulo Theories. Satisfiability Modulo Theories (SMT) is the problem of deciding the satisfiability of a quantifier-free first-order formula with respect to some decidable theory T (e.g., linear arithmetic over the rationals, LRA). An Optimization Modulo Theories over LRA problem is the problem of finding solution(s) to an SMT(LRA) formula which optimize some rational-valued objective functions, either singularly or lexicographically (with the objective functions prioritized). Very efficient SMT(LRA) and OMT(LRA) solvers are available, which combine the power of modern SAT solvers with dedicated linear-programming decision and minimization procedures. For instance, the solver OptiMathSAT [24] was able to handle problems with thousands of Boolean/rational variables in less than 10 minutes each.

Nguyen et al. [64] introduce constrained goal models (CGMs), extending the notion of goal models by assigning penalties or rewards to goals and their refinements and defining constraints for possible solutions. The authors provide several reasoning techniques to find

solutions that respect the defined constraints and optimize the given penalty or reward functions. A possible application is to assign cost for each goal node as a penalty and reason on the goal model for finding a solution that respects all constraints and optimizes given objective (penalty or reward) functions. The prototype tool developed in this work uses the OptiMathSat [24] solver for reasoning and provides a graphical editor for goal modeling.

Next Release Models. Bagnall et al. [22] model requirements as acyclic graphs where nodes represent requirements and edges denote that the source requirement is a prerequisite of the target one. Towards a more structured approach to modeling requirements for the next release problem, Carlshamre et al. [148] identify six inter-dependencies for requirements considered for the next releases of a software product and represent these as a matrix using spreadsheets as well as a graph where the nodes represent requirements and labeled edges represent inter-dependencies. Table 5.1 lists these inter-dependencies. Our proposal accommodates all six of these relations, though we don't always use the same name.

Table 5.1: Inter-Dependencies between requirements for the next release identified by Carlshamre et al [148]

Inter-Dependency	Meaning
REQUIRES	R_1 requires R_2 to function
AND	R_1 requires R_2 and vice versa
TEMPORAL	R_1 needs to be implemented before R_2
CVALUE	R_1 positively or negatively contributes to the customer value of R_2
ICOST	R_1 positively or negatively contributes the cost of R_2
OR	Only one of $\{R_1, R_2\}$ has to be implemented

5.2 The Next Release Problem

The next release problem concerns finding the optimal set of requirements to be implemented for the next release of a software product. There may be multiple criteria for the optimal solution, such as cost and customer value. In a given problem, not only the independent values assigned to requirements but also the inter-dependencies among requirements impact the optimal solution. For example, a requirement may decrease the cost of implementing of another requirement, or increase the customer value of yet another requirement. In this light, release engineers should consider inter-dependencies among requirements when calculating the optimal solutions. Our proposed framework includes the following concepts and relations for modeling NRPs:

Goal: Requirements are represented by goals. A *mandatory* goal must be included in the solution set of a goal model. Goals that are implemented in the previous releases are marked as *implemented*. A goal is either satisfied (included in the next release) or denied (not included). Top goals have associated *rewards*, leaf goals have *costs* and *customer*

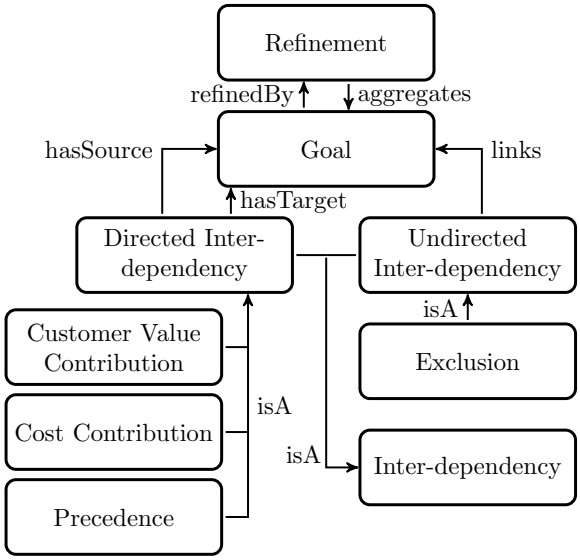


Figure 5.2: Metamodel for the NRP

values. Reward, cost, and customer value assignments are optional but it should be consistent (all relative or absolute) for all goals to obtain accurate results for the analysis.

Refinement: A refinement relates a set of child goals to a parent goal. A parent goal may have multiple refinements, each of which is an alternative way of achieving that goal.

Figure 5.3 shows a fragment of the goal model derived from interviews with a developer implementing a research tool. Goal nodes are represented as oval nodes. Refinement nodes that link multiple child nodes to a parent node are shown as black-filled circles. Refinement nodes that link one child node to a parent goal are omitted for visual simplicity. For example, the goal '/Analysis results are shown/' is refined into '/Graphical model highlighted/', '/Textual report shown/'.

Cost contribution: A *cost contribution* inter-dependency exists between two goals when the implementation of a goal has an effect on the cost of implementation of another goal as stated in [148]. The effect could be positive when the implementation of one goal increases the cost of the other, or it could be negative when it decreases the implementation cost of the other goal. Cost contributions can be cyclic, and they do not effect the well-formedness of a goal model. If both the target and the source goals are included in a solution, that is, both of them are *satisfied*, the link contributes to an objective defined for the solution. Each positive cost contribution means that the implementation of two goals together brings additional cost for the solution. On the other hand, each negative cost contribution indicates a reduction in the cost. The release engineer may opt for a qualitative analysis by not assigning any quantitative measure (weight) to this inter-dependency, or she can fine tune the model with such measures to capture the intensity of each inter-dependency.

In our graphical modeling language, instead of using a dedicated node and two edges (one incoming edge from the source and one outgoing edge to the target goal), we use

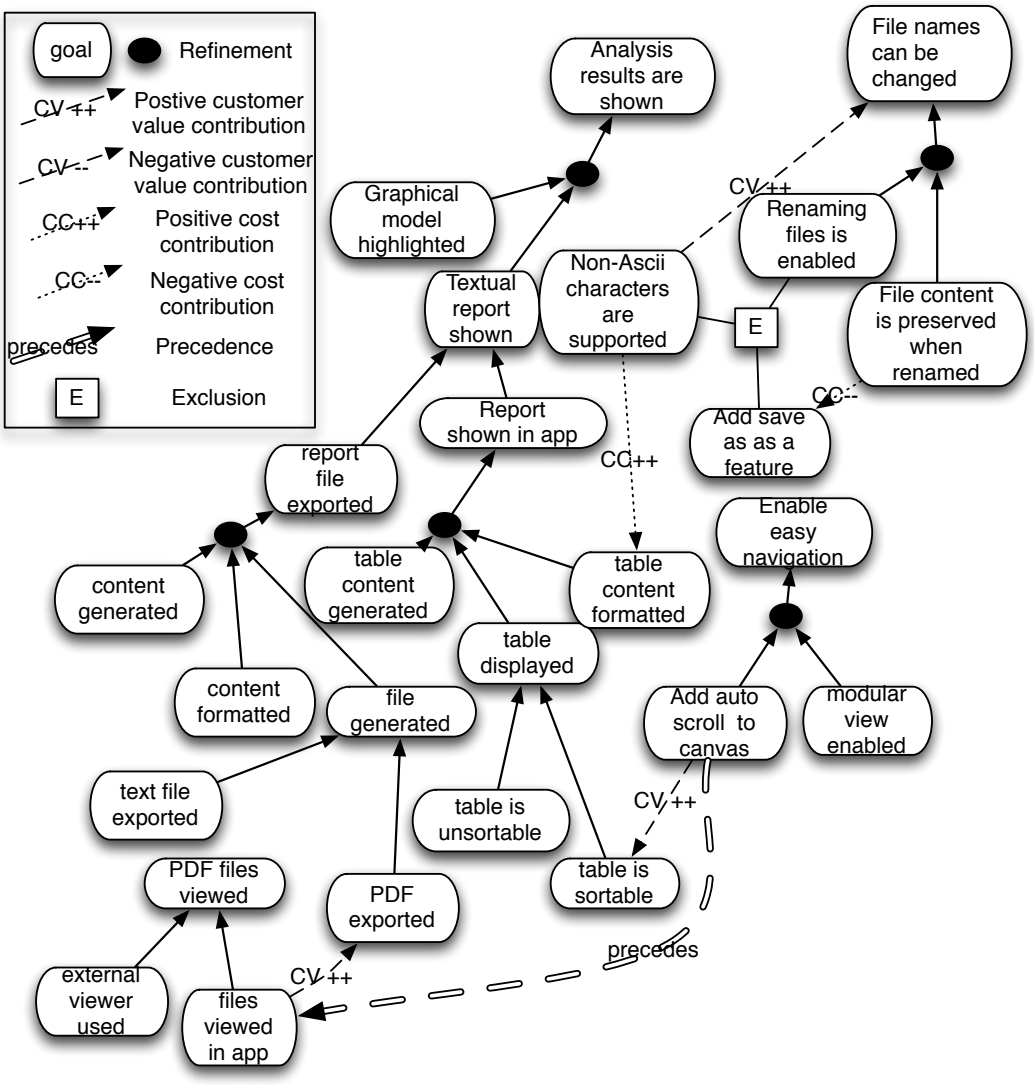


Figure 5.3: An example model

dotted line edges from source to target goals labeled as *CC++* for the positive cost contribution and *CC–* for the negative cost contribution to decrease the visual clutter. In the model presented in Figure 5.3, the support for ASCII characters has a positive cost contribution on the formatting of the table content and preserving the goal '/file content is preserved when renaming the file/' has a negative cost contribution on '/adding "save as" as a feature/'.

Customer value contribution: Implementation of a goal may increase or decrease the customer value of another goal, and such inter-dependencies between two goals is captured by a *customer value contribution*. Similar to cost contribution, customer value contributions could be cyclic. A positive customer value contribution indicates that the implementation of these two goals results in an increase in the customer value. However, a negative customer value contribution indicates that the implementation of the source goal decreases the customer value of the target goal. Similar to cost contribution, it is optional to assign any quantitative values to this inter-dependency. The release engineer may reflect the differences in severity of each inter-dependency by assigning absolute or relative values, or treat them the same by not assigning any value.

In Figure 5.3 customer value contribution inter-dependencies are shown via dashed line edges and labeled as *CV++* for the positive customer value contribution and *CV–* for the negative customer value contribution. For example, having an integrated PDF viewer in the application, which is denoted by '/files viewed in app/' goal, has a positive customer value contribution to the exporting reports in PDF format, which is represented as '/PDF exported/' goal node.

Exclusion: The release engineer may decide not to include certain goals together in the same release for various reasons, such as the limited developer-hours or the marketing strategy of the company. Such goals are related to each other with an n-ary *exclusion* inter-dependency and which states that those goals cannot be included in the same release together. By default, the exclusion inter-dependency states that at least one of the goals should be omitted from the release. The release engineer may also state that m out of n goals could be included in the release. Unlike cost or customer value contribution inter-dependencies, exclusion does not introduce any reward or penalty to the solution, rather it creates a constraint for valid solutions. Any selection of goals that does not respect the exclusion is not considered as a valid solution for the next release.

Exclusion inter-dependencies are shown with squares with the capital letter E in our graphical notation. In Figure 5.3 there is one exclusion that relates three goals together. According to the model, '/Non-Ascii characters are supported/', '/Renaming files is enabled/', and '/Add save as as a feature/' cannot be included in the same solution.

Precedence: When there is a temporal order for the implementation of two goals, a *precedence* inter-dependency exists between the two and denotes that the implementation of the source goal should precede the implementation of the target goal. Similar to exclusion, precedence inter-dependency imposes a constraint on valid solutions, the target goal (the goal that should be implemented later) cannot be included in the solution without the source goal (the goal that should be implemented before). A valid solution should respect the constraints introduced by the precedence and exclusion relations.

Precedence inter-dependencies are represented by dashed double-line edges and labeled as 'precedes' in Figure 5.3. In our sample model, it is stated that adding auto-scroll feature to the canvas should precede the implementation of the integrated PDF viewer.

Objective Function. The release engineer sets objectives to determine the optimal solution for the next release. The objective could be a single objective such as minimizing cost, or maximizing the top goal rewards, or maximizing positive customer value inter-dependency. Multiple objectives can be set by using a lexicographic ordering, where the solution is first optimized with respect to the first objective, and if there are multiple solutions with the optimal value, the solution with the optimal value for the second objective is selected, and so on. The release engineer can also combine multiple objectives by assigning weight to each of them in a linear multi-objective function, in this case differences in the scale of magnitude for the objectives should be considered.

The next release problem. Given a model whose elements are described above, the next release problem concerns finding a set of leaf goals that respects the constraints imposed by the model elements and optimizes the objective functions stated for the model while satisfying all mandatory requirements.

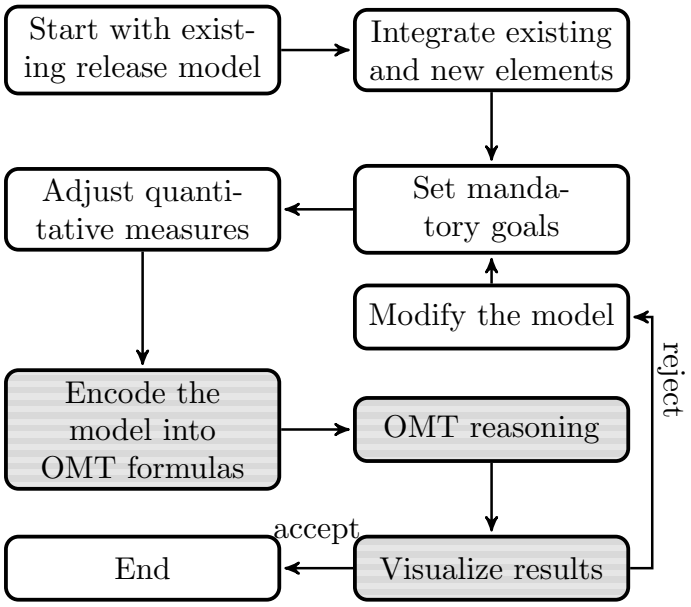


Figure 5.4: Process diagram for solving the NRP

Methodology. Figure 5.4 presents the process for solving the next release problem. The input is a requirements model that includes all implemented and candidate requirements. It is the decision of the release engineer to mark the implemented requirements as mandatory for dropping or changing them may introduce a high cost. On the other hand, it might be beneficial to re-evaluate some of the implemented requirements to explore alternative solutions. A proposed next release model is a model that represents the new requirements, refinements, and inter-dependencies proposed for the next release as well as the previous ones. The release engineer links previous and new model elements

via inter-dependencies and refinements as she sees fit. The merged model constitutes the search space for solutions that satisfy all mandatory requirements, however, there may be multiple solutions for a given goal model. At this stage, the release engineer may adjust quantitative measures or prefer a qualitative analysis in order to differentiate the optimal solution(s) from the rest. Due to its computational power and optimization capabilities, we rely on OMT reasoning techniques to find a solution. In order to use an OMT solver, the goal model should be encode *(i.)* encoded as OMT formulas and the optimization scheme should be set, *(ii.)* OMT reasoning should be applied to the formulas. Once the solutions are returned by the solver, they should be visualized on the graphical model so that the release engineer can interpret the results. If the engineer accepts the presented optimal release candidate as a solution the process ends. If the model is not satisfiable, that is, there is no solution satisfying the constraints and refinements, or the engineer does not find the presented release candidate(s) adequate, she modifies the model by adding or dropping requirements, changing the mandatory and optional goals, and adjusting the weights and re-run the automated processes to discover new release candidates. Our prototype fully automatizes encoding, reasoning, and visualization of the solutions (gray-filled nodes) in the process presented in Figure 5.4, and supports release engineers during modeling.

5.3 Encoding NRP to SMT

Our conceptual framework suplaments the constrained goal models presented in [64]. We adopt their syntax and semantics for goals and refinements, but enrich their conceptual model with inter-dependencies that better describe NRPs.

Definition 5.1 (Next Release Model) *A Next Release Model (NRM) is a tuple* $M \stackrel{def}{=} \langle B, N, D, \Psi, W \rangle$*, such that,*

- $B \stackrel{def}{=} G \cup R$ *is a set of atomic propositions where* $G \stackrel{def}{=} \{G_1, \ldots, G_n\}$, $R \stackrel{def}{=} \{R_1, \ldots, R_k\}$ *are respectively a set of goal labels and refinement labels.*
- N *is a set of numerical variables in the rationals;*
- D *is a goal graph, such that, all its goal nodes are labeled by a goal label in* G*, all its refinements are labelled by a refinement label in* R*;*
- Ψ *is an SMT(LRA) formula on* B *and* N*;*
- $W : G \mapsto \mathbb{Q}$ *is a function assigns a rational number to each goal.*

An NRM is a directed acyclic goal graph with goals as elements and refinements as grouped edges, weights assigned to goals, and constraints introduced in form of $SMT(LRA)$ (satisfiability modulo theories with linear arithmetic over rationals) formulas.

Goals and refinements. G, G_i represent goals and R, R_j represent refinement label as in $\{$'*Graphical model highlighted*', '*Textual report shown*' $\} \xrightarrow{R_5}$ '*Analysis results are shown*' . In Figure 5.3 labels are not shown for visual simplicity. As defined above goals and refinements are atomic propositions. A *mandatory goal* G_i is set to be *satisfied* by the release engineer so the encoding is $(G_i := \top)$. Each refinement R in an NRM where $\{G_1, \ldots, G_n\} \xrightarrow{R} G_p$ introduces the following formula, $(\bigwedge_{i=1}^{n} G_i \leftrightarrow R) \wedge (R \rightarrow G_p)$. For all refinements $\{R_1, \ldots, R_n\}$ of a goal G_p, $G_p \rightarrow \bigvee_{i=1}^{n} R_i$ is inserted.

Exclusion. Exclusion introduces a constraint that $G_1, \ldots, G_n$ cannot be satisfied all together, so it is encoded as $\neg(\bigwedge_{i=1}^{n} G_i)$.

Precedence. Precedence relation denotes that the implementation of the source precedes the implementation of the target node. The valid cases are that *i.* both are included in the solution, *ii.* both are excluded from the solution, and *iii.* the source is included but the target is excluded from the solution. Given G_1 precedes G_2, the encoding captures this relation is $(G_2 \rightarrow G_1)$.

Example. Figure 5.5a presents a simple model containing a precedence relation between G4 and G5 so the truth assignments that violate the encoding are the ones that include G4 but not G5. In Figure 5.5b, assuming that G1 is mandatory, two solutions are {G3, G4, G1, G5, G2}, {G3, G4, G1, G5, G6, G2}. Assuming the objective is to minimize cost and each leaf goal has equal costs, the first solution is returned by the solver. The release engineer may favor by assigning a negative cost. Then the second solution is returned by the solver. On the other hand, the only solution in Figure 5.5b is {G3, G4, G1, G5, G2} for G6 cannot be in the same solution as G4, and G4 is required to satisfy G1, which is mandatory.

(a) (b)

Figure 5.5: Sample models demonstrating precedence relation

Cost contribution. The encoding of the positive or negative cost contribution has two steps. Given that G_1 contributes to the cost of G_2,

1. a new atomic proposition $G_{1,2}$ is created, and a new constraint is encoded as $(G_{1,2} \leftrightarrow (G_1 \wedge G_2))$,
2. it's weight is set as $W(G_{1_2}) = Q$ where $Q \in \mathbb{Q}$. If the release engineer does not specify the intensity for the inter-dependency the weight is set to zero. We keep the

number of positive and negative cost contibution inter-dependencies included in the solution and use this value if the release engineer opts for qualitative analysis by not stating weights for this inter-dependency.

Customer value contribution Similar to the cost contribution, the encoding of the customer value contribution has two steps. Given that G_1 contributes to the customer value of G_2,

1. a new boolean $G_{1,2}$ is created, and its truth value is encoded as $(G_{1,2} \leftrightarrow (G_1 \wedge G_2))$,
2. it's weight is set as $W(G_{1_2}) = Q$ where $Q \in \mathbb{Q}$. If the release engineer does not specify the intensity for the inter-dependency the weight is set to zero. We keep the number of positive and negative customer value contribution inter-dependencies included in the solution and use this value if the release engineer opts for qualitative analysis by not stating weights for this inter-dependency.

Definition 5.2 (Release candidate) *Let $M \stackrel{def}{=} \langle B, N, D, \Psi, W \rangle$ be a next release model. A release candidate of M is an LRA-interpretation μ over $B \cup N$ such that $\mu \vDash \Psi$.*

Proposition 1 *Let $M \stackrel{def}{=} \langle B, N, D, \Psi, W \rangle$ be a next release model, μ a release candidate of M, and $\{obj_1, \ldots, obj_k\}$ LRA–terms occuring in Ψ. Then we have that:*

1. *for every $obj_i i$ in $\{obj_1, \ldots, obj_k\}$, μ minimizes [resp. maximizes] obj_i if and only if μ is a solution of the OMT(LRA) minimization [resp. maximization] problem $< \Psi, < obj_i >>$;*
2. *μ lexicographically minimizes [resp. maximizes] $< obj_1, \ldots, obj_k >$ if and only if μ is a solution of the OMT(LRA) lexicographic minimization [resp. maximization] problem $< \Psi, < obj_i, \ldots, obj_k >>$.*

An *optimal release candidate* is a release candidate that lexicographically minimizes (resp. maximizes) the objectives of a given next release model.

5.4 Reasoning

The encoding of the goal model to SMT/OMT ensures that the reasoner returns the global optimal solution for the next release problem. In this section we provide examples to better explain the reasoning process.

Mandatory goals. If there is no mandatory goal in the model, no rewards specified for the top goals, the optimal solution is the empty set. The release engineer ensures the inclusion of the intended goals to the solution set by setting them mandatory. Considering only the '/PDF files viewed/' goal and its refinements located in the lower left part of the model presented in Figure 5.3, and assuming that the top goals is mandatory, there are three possible solutions in the model. Either one of the two goals are satisfied, or

both of them satisfied so that their parent goal is satisfied as well, therefore included in the solution. Other objectives and quantitative values of the goals determine the optimal solutions.

Cost. In order to differentiate among these three solutions, cost assignment can be used. Assuming that the minimizing the cost is the objective, if both goals have a negative cost, that is, the inclusion of each goal further reduces the cost calculated by the reasoner, they are both included. A more realistic assignment is to assign positive costs to the leaf goals. In this case the reasoner returns the solution with the minimum cost. If the costs of the two are equal, the reasoner lists two separate solutions as they are both optimal. Cost assignment can be relative or optimal. If implementing the external viewer costs $5000 and has a cost assignment of 5000, and the in application viewing costs $500 and has a cost assignment of 500, the reasoner returns the cheaper option. However, the same result is obtained by simply assigning the relative costs, for example, 10 and 1, respectively. The reasoning for the customer value follows the same principles as those for cost.

Exclusion. There is an exclusion inter-dependency on the upper right side of Figure 5.3. Given that '/File renaming enabled/' goal is mandatory, it is certain that its two child goals must be included in the solution. So either one or both of '/Non-ASCII characters are supported/' '/Add save as a feature/' are omitted from the solution.

Cost and customer value contribution inter-dependencies. Positive and negative synergistic relations among requirements in terms of cost and customer value are captured by corresponding inter-dependencies in next release models. For example, implementing a graphical user interface for a function increases the customer value of the function, and the positive customer value inter-dependency from the goal of implementing the graphical user interface to the goal of implementing the function represents this case. It is possible to use these inter-dependencies in multiple ways in our approach. We acknowledge the fact that it is difficult to estimate the values of these inter-dependencies, so it is possible not to assign the severity of the inter-dependency. We keep the number of inter-dependencies included in the solution for each of the four kind and use these values for optimization if there is an objective of minimizing (resp. maximizing) any one of them. Similar to cost and customer value assignment, a relative assignment scheme is possible to differentiate the intensity of different contributions. Finally, it is possible to use the absolute values if the data is available. The release engineer can define objectives related to these contributions. An example is to minimize positive cost contributions where the reasoner returns solutions with minimal cost contribution, either in terms of number of negative cost contributions included in the solution or the sum of intensities of negative cost contributions. A more complex objective is to combine positive and negative cost contributions in a single objective function, minimizing NumberOf(CC++) − NumberOf(CC--), for example. Finally, the intensity of these contribution can be combined with the values assigned to goals, calculating a total cost where TotalCost = Sum(GoalCost) + Intensity(CC--) − IntensityCC(++).

Precedence. The precedence inter-dependency denotes a temporal order for the implementation of given goals. In the example model it is stated that '/Add auto scroll to canvas/' precedes '/files viewed in app/' on the bottom right part of the model. According to this inter-dependency, the solutions that include '/files viewed in app/' but not '/Add

auto scroll to canvas/' are not valid since the preceding goal is left out even the goal that is supposed to be implemented later is included in the solution.

Optimization. Above examples are individual cases where the optimal solution is based on a few values. However, in a model those values how to be aggregated in a way that represents the intention of the release engineer.

- Lexicographic optimization: A finite number of objective functions are optimized in lexicographic order. An example of a lexicographic order of objective functions is this: (*i.*) minimize goal cost, (*ii.*) maximize goal rewards, (*iii.*) maximize negative cost contributions, (*iv.*) maximize positive customer value contribution, (*v.*) minimize negative customer value contribution. According to this setting, the reasoner returns the solution that has the minimum goal cost where goal cost is the sum of costs of goals that are included in the solution. If there are several solutions with the same goal cost, the reasoner orders them by their goal rewards and returns the solution with maximal rewards. The process continues until the reasoner returns the single optimal solution, or all the solutions that are equally optimal.

- Linear objective function: Another option is to have a linear objective function that combines the relevant values in a way specified by the engineer.

- Combination of the two: Two approaches described above can be combined for more complex solutions. The release engineer may specify multiple linear objective functions and specify a lexicographic order among them combined with single objectives.

5.5 Evaluation

5.5.1 Next Release Tool

A prototype tool is implemented to support modeling and reasoning. Next Release Tool (NRT) is a standalone application that is available for Windows, MAC OS, and Linux operating systems. The component diagram of the tool is presented in Figure 5.7. Figure 5.6 includes a screenshot of the tool where the project manager is shown on the left side and the palette is located on the right side. The center of the tool is dedicated to graphical modeling and the bottom part is used to set the properties of the model, and displaying the results from the OMT solver.

Graphical Modeling. NRT provides a graphical editor based on Eclipse Modeling Framework (EMF) [1] and Graphical Editing Framework (GEF) [2]. Certain rules are set to prevent user mistakes when modeling, for example, a goal cannot be refined into itself so it is not possible to create both incoming and outgoing edges from a goal to a refinement node.

[1] `http://www.eclipse.org/modeling/emf/`
[2] `http://www.eclipse.org/gef/`

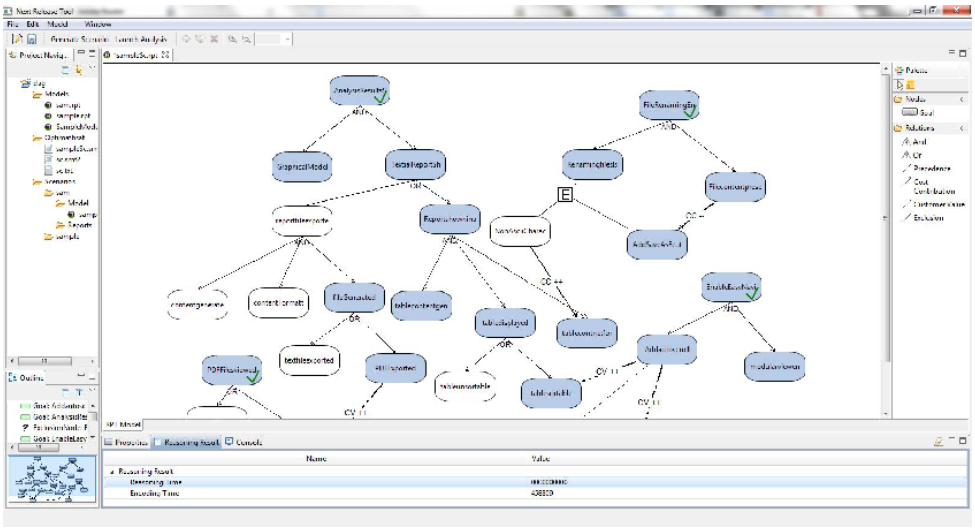


Figure 5.6: A screenshot from Next Release Tool

Well-formedness check. The tool automatically check the consistency of the drawn model with the semantics of the next release conceptual framework.

SMT/OMT Formula Transformations. The model elements and the optimization scheme stated by the user are automatically encoded as SMT/OMT formulas by the tool.

Automated Reasoning. The tool feeds OptiMathSolver with the encoding of the model and proper commands for the solver based on the and retrieves the results. The reasoning is done by OptiMathSolver.

Visualizing the results. The results retrieved by the solver are presented in two different ways. First, the results are reported in a written report. Second, the optimum release candidate is highlighted in the graphical model if it is found. Otherwise, the user is informed that no solution is found. In case of multiple optimal release candidates (pareto-optimal release candidates that has the same weight, which is the minimum among all release candidates) the user is presented with the first release candidate returned by OptiMathSolver, the implementation of presenting multiple solutions is an ongoing work.

5.5.2 Scalability Experiments

We set up three experiments to investigate the scalability of the Next Release Tool. All experiments are run on a Windows 64 bit machine with Intel(R) Core(TM) i7-3770 CPU 3.40Ghz and 8GB of RAM, and collected the reasoning time reported by OptiMathSAT to find the solution. For the experiments, we use OptiMathSAT version 3.5, which is the latest version available to public. The Next Release Tool, models used for the experiments, their corresponding encodings in the SMT-LIB v2 Language [149], and the results are available at `https://www.nextrelease.eu`.

Scalability with respect to problem size. We define the size of the problem

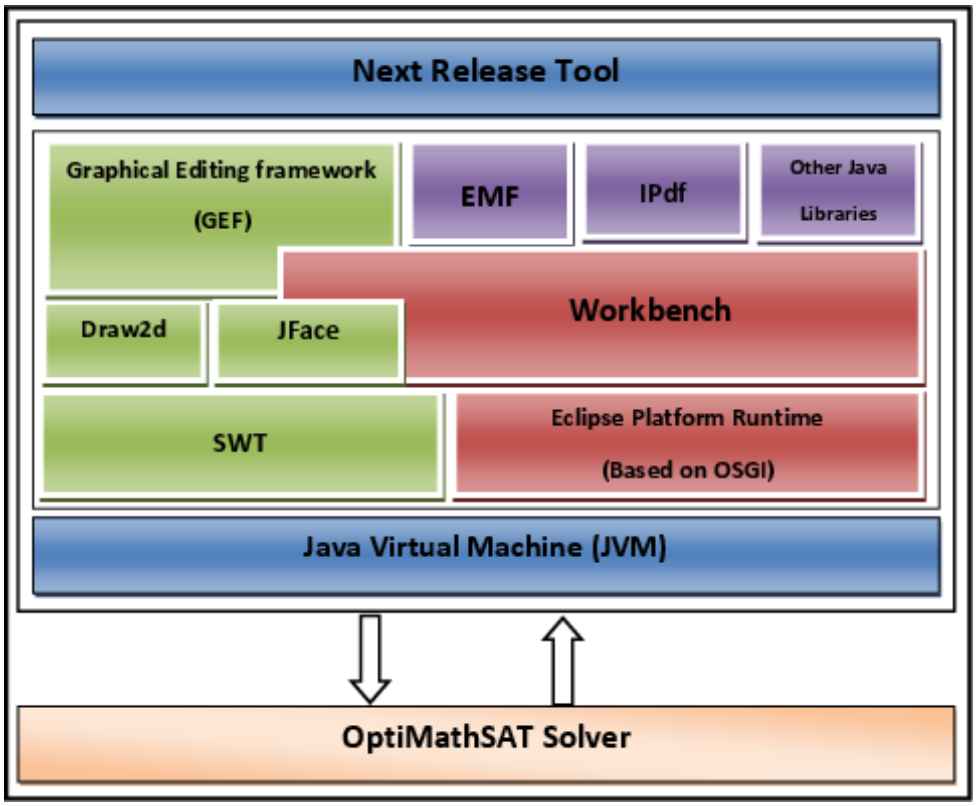


Figure 5.7: Component Diagram of Next Release Tool

as the number of model elements, that is total number of goals, refinements, and inter-dependencies. In order to figure out how our tool scales with respect to the size of the problem, we have created an input model which includes 13 goal nodes, nine refinements, and four inter-dependencies. Lexicographic optimization scheme is selected for the optimization. We have generated 75 cases by replicating the input model 10 to 750 times and connecting the replicas to a root goal via the same refinement element. As a result, we have generated test models of size from 255 to 17275 model elements. We have run the experiment five times. Creating goal models for representing and analyzing requirements is a human-centric activity for it involves understanding and capturing. Applying patterns or following catalogers may ease the process, but it cannot be fully automized. Manual creation of goal models limits their size and complexity. Furthermore, comprehending and modifying goal models become too difficult beyond a certain level. One of the most intensive i^* models created for a real-life application includes approximately 525 links and 350 elements [150]. Given this information, our artificially generated experiment cases have sufficiently enough number of model elements. Figure 5.8 reports a linear trend for the result of this experiment. The size of the models are shown on the x-axis whereas the y-axis reports the reasoning time in milliseconds.

Scalability with respect to problem complexity. In order to understand how the tool behaves as the number of inter-dependencies increase, we generated 400 test

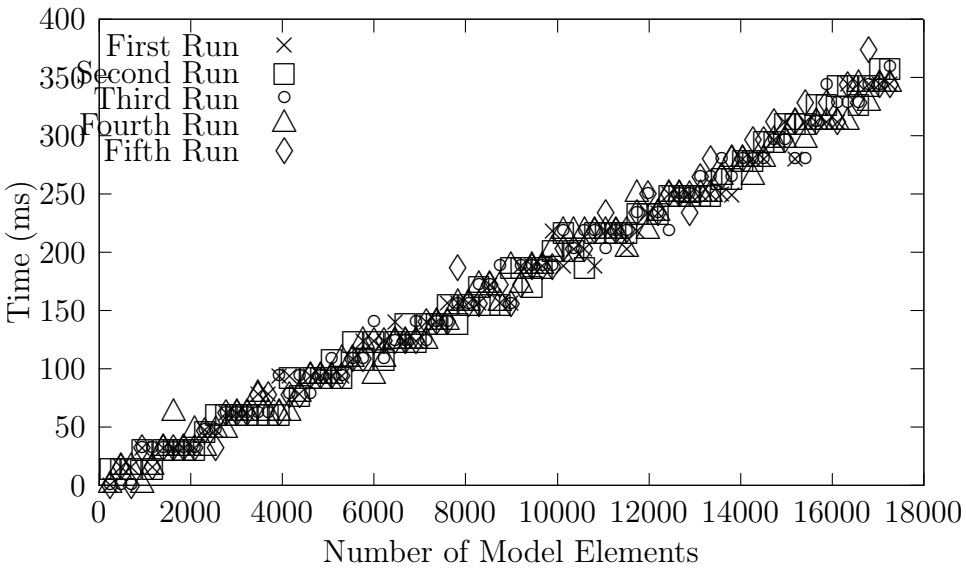


Figure 5.8: Scalability wrt problem size

cases by increasing the number of directed inter-dependencies in a fixed size model. Our initial model had 250 goal nodes, 118 refinement nodes, 16 precedence, and 16 exclusions. For each run, an increasing number of random directed inter-dependencies are placed between randomly selected goal nodes from the original model. Figure 5.8 presents the result of this experiment. The number of added inter-dependencies are shown on the x-axis whereas the y-axis reports the reasoning time in milliseconds. There is a smooth increase in reasoning time until approximately 150 added inter-dependencies, the reasoning time increases rapidly after this threshold.

Scalability with respect to alternatives. Multiple refinements of a goal introduce alternative solutions for satisfying that goal (/i/* and Tropos use the term OR decomposition.). In order to test the scalability of the tool against the number of alternative solutions in a model, we created five model variations with fixed number of goals and inter-dependencies, 11918 and 1402, respectively. In the first model, all 5609 refinement elements had two incoming edges, and a parent goal had only one incoming refinement edge. This setup corresponds to having all AND decompositions in an i or Tropos model. For the second variation, we have started with the first model, but for the 25% refinement elements, we have created another refinement element pointing to the same parent goal, and directed one of the two incoming edges of the original refinement element to the newly created one, in other words, we have converted 25% of the AND decompositions to OR decompositions. For the rest of the variations, we have increased the conversion rate to 50%, 75%, and finally 100% and run the reasoner releases times for each model. Figure 5.10 presents the results of this experiment. The reasoner finds a solution for the first two models run under a second, for the second two models around three seconds and for the fifth model around five seconds. These results confirm that as the number of alternative solutions increases in a problem, so does the reasoning time. Figure 5.11 presents the

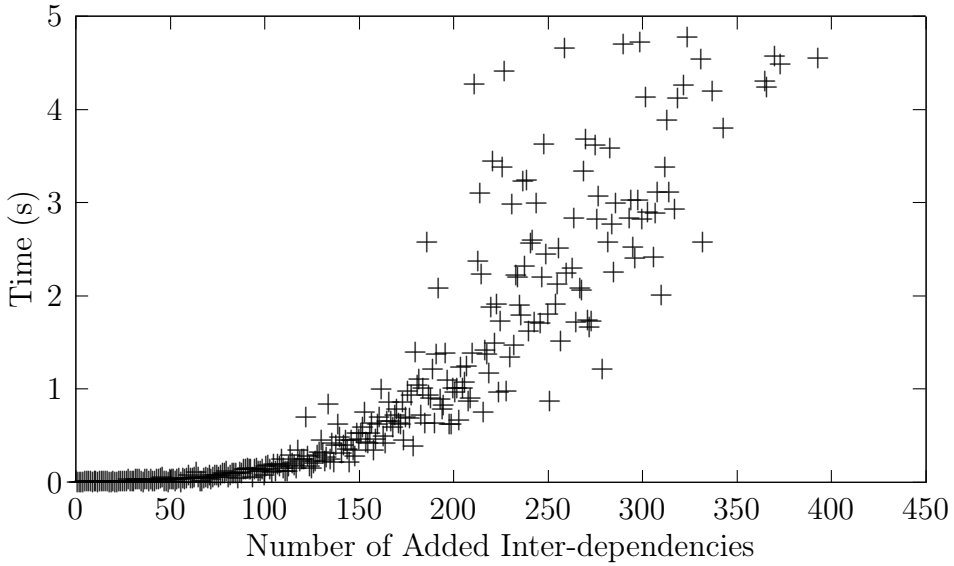


Figure 5.9: Scalability wrt problem size

results for the same experiment, where y axis is the average time passed to find a solution and x axis indicates the percentage of OR refinements in the model.

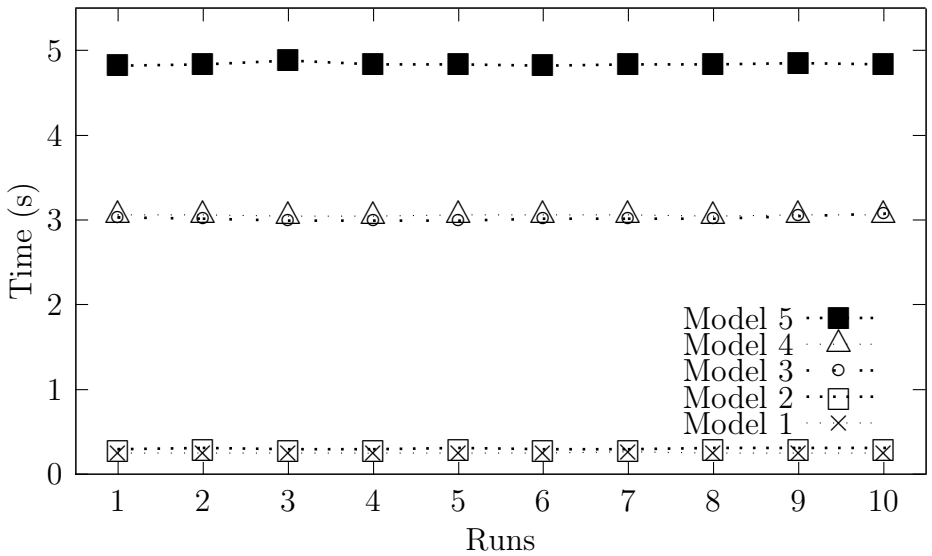


Figure 5.10: Scalability wrt problem size

Discussion. We focus on evaluating the scalability of our proposed approach with respect to the size of the model, complexity of the model in terms of inter-dependencies that affect the selection of the optimal solution, and the number of alternative solutions. We use artificially created models for our experiments that have higher size and complexity than reported real-life examples. The results confirm that our tool is scalable for all three

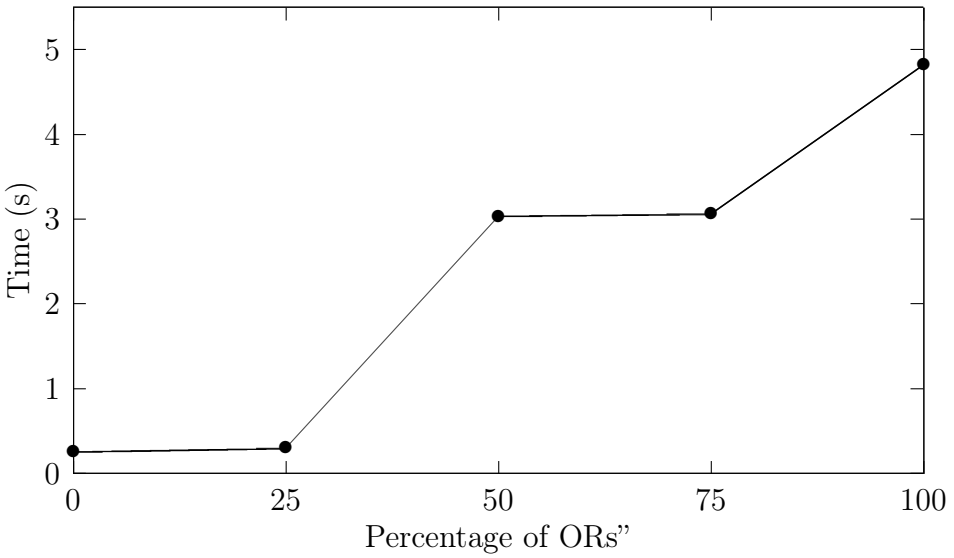


Figure 5.11: Scalability wrt alternatives

cases even for artificially generated big models. However, as the connectivity within a model via directed inter-dependencies increases, the scalability of the tool decreases. Even in this case, the tool returns the optimal solution for a model of 250 goals and 200 inter-dependencies. The inter-dependency to goal ratio is much higher than the real-life cases reported by [148], which is around 20%.

5.6 Chapter Summary

Our contribution is three-fold in this paper. First, we propose an expressive goal-oriented framework to represent the next release problems. Second, we formalize frameworks by mapping NRP into SMT/OMT formulas. Third, we have implemented a prototype tool that supports the modeling process, automatically maps NRP models into SMT/OMT formulas, apply SMT/OMT reasoning utilizing a back-end solver, and visualize the results in two different ways. Our modeling framework is more expressive than the existing approaches to the NRP, supporting NRP relations and preferences. Moreover, the scalability of our tool has been established experimentally.

Our visual notation suffers from a common problem of visualizing goal models. Goal models are intended to be created and read by humans, however, they become unwieldy to understand as they grow. Limited screen space prevents users to visualize the whole model at once, as the nodes and labels shrink when the user zooms out. One solution proposed to overcome this problem is to visualize the model in a more modular fashion, either using layers assigned for different model elements or expanding and collapsing parts of the models. We leave this feature of the tool as future work. Another limitation of our approach is that goal modeling is a human-centric activity. It is laborious to build goal

models and identify inter-dependencies among goals.

Chapter 6

Risk Analysis in Sociotechnical Systems

Risk analysis should be an integral part of any development project for complex sociotechnical systems intended to operate within an uncertain environment. Yet studies show that risks are highly underestimated in IT projects leading repeatedly to disastrous financial losses, delays, or total failures [14]. Risk analysis involves risk assessment and management, the activities for the first comprise risk identification and evaluation whereas the latter requires the selection of treatments to prevent or at least ameliorate to the implications of risks [151].

Existing approaches to risks identify situations that may lead to risks, assess the impact of risks on the system, and introduce treatments to mitigate the impact. The treatments may refine existing design, or even change the initial requirements [152]. However changing the requirements in later stages of software development adds additional cost, thus integrating risk analysis with requirements analysis has financial benefits in addition to leading to more robust designs [153].

Goal models have been used to capture and hierarchically structure requirements for sociotechnical systems. Higher level goals capture stakeholder needs whereas lower level goals capture strategies for fulfilling higher-level ones. The structure of goal models supports systematic analysis techniques to determine solutions to root-level goals [147]. Goal models can also be used to reason about security and trust for a system-to-be [154]–[156].

Considering the advantages of goal-oriented requirements engineering and integrating risk analysis with requirements analysis, Asnar et al. [25] proposes a goal-risk analysis framework. Goal-risk models capture stakeholder requirements, risks, and treatments, and support their analysis to find the optimal solutions with respect to cost. This approach focuses on avoiding risk while minimizing cost.

Alternative ways of managing risk include taking risks in exchange of possible greater benefits, accepting risk, and preparing a contingency plan. For the former, the solutions are optimized with respect to a utility function of stakeholder goals, for the latter, treatments to impact of risks should be modeled and analyzed. Cost is a factor that cannot be ignored for both cases. So a thorough risk analysis requires multi-objective optimization and various types of analysis to identify alternative solutions. In many cases, the analysts

consider trade–offs between cost, risk aversion, utilities, and so on.

We adopt Asnar's framework, and extend it with constrained goal model [64] in order to support multiple types of risks analyses for different risk management strategies. Our extension exploits Satisfiability Modulo Theories/Optimization Modulo Theories (SMT/OMT) solvers to support efficient reasoning and discovery of solutions for risk analysis problems. We define multiple objective functions such as maximum goal reward, minimum risk factor, maximum risk prevention, minimum cost, and minimum damage on risk models and discover globally optimal solutions with respect to multiple objectives.

6.1 Risk Modeling

We follow the three layered approach of Asnar et al. [25] for modeling risk in goal models, and we keep the names for the sake of convention: asset layer, event layer, and treatment layer. These layers are shown in green, red, and blue, respectively, in our meta-mode presented in Figure 6.1. Details of the meta-model are explained below.

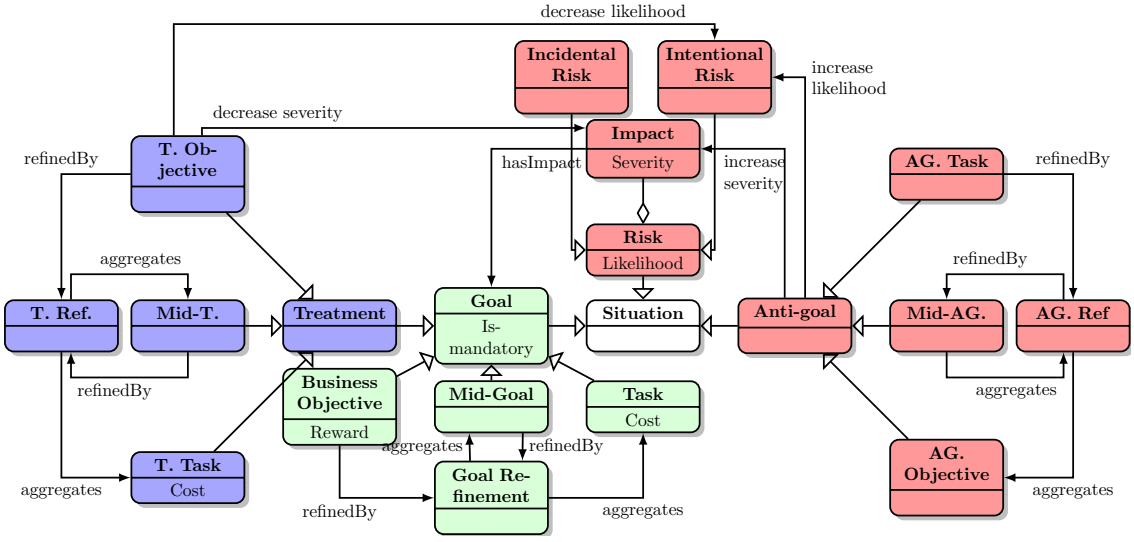


Figure 6.1: Meta–model for risk modeling in goal models

A *situation* represents a partial state of the world [157]. In our goal modeling framework, situations are used as binary propositions that either hold *true* or *false*. *Goals* are desired situations. Stakeholder goals are represented in the asset layer. Goals without any parents (top goals) are *business objectives* that have associated *rewards* with them. The reward of a business objective is the utility gained from the achievement of the objective. Once the business objectives are identified, they are refined into more concrete child goals. *Refinement* nodes aggregate child goals and connect them with a parent goal. A parent goal may have multiple refinements, and it is satisfied when at least one of its refinement nodes is satisfied. A refinement node is satisfied when all child goals associated with the refinement goals are satisfied. Finally, the leaf goals with no incoming edges from refine-

ment nodes are *tasks* that have associated *costs*. If a goal is labelled *mandatory*, it must be satisfied by any proposed solution. Otherwise, a goal is *preferred* and is to be satisfied by a solution if it is not in conflict with any elements of that solution.

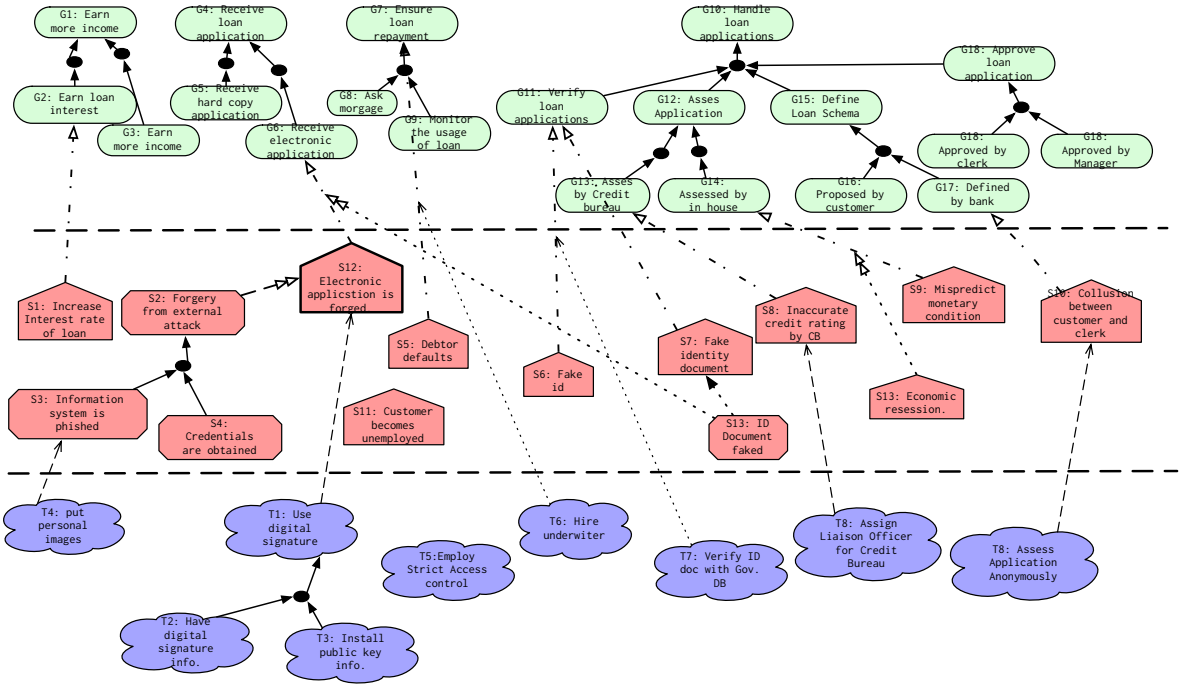


Figure 6.2: Illustrative example: Loan Origination Process (LOP)

To illustrate our modeling framework, we model the Loan Origination Process (LOP) [25] in Figure 6.2. The process is initiated by a loan application and ends with a decision. The bank's ultimate motive is, of course, to earn income, and this is to be achieved by handling loan applications, and ensuring loan repayment. The topmost layer is the asset layer, goals are depicted as oval nodes, while refinement nodes are shown as black-filled circles. Child nodes are connected to refinement nodes, which then relate them to parent nodes. One example is G7: Ensure loan repayment, which is refined in to G8: Ask mortgage and G9:Monitor usage of loan. We omit numerical values from the model presented for visual simplicity. If the reward of all top goals are the same, the attribute values can be omitted. Similarly, all leaf goals (tasks) have a cost, which can be filled with relative (for example, within a scale of 1–5) or absolute values (230 for G9).

Risks, *anti-goals*, and other situations that lead to risks are modeled in the event layer. A situation models a partial state of the world [157]. An anti-goal is an undesired situation of the system being modeled, yet it is desired by another system, which may be malicious, such as S2: Forgery from external attack presented in an octagon node in Figure 6.2. Similar to goal refinement, an *anti-goal refinement* links a set of child anti-goals to their parent anti-goal. A parent anti-goal may have multiple refinements as alternative ways of being achieved. The parent anti-goal is achieved when at least on of its refinements is achieved. A refinement is achieved when all child anti-goals connected

to the refinement are achieved. For example, in Figure 6.2 S2: Forgery from external attack is refined into S3: Information system is phished and S4: Credentials are obtained. A *risk* is a situation that harms one or more goals of the system being modeled with a possibility of loss. Each risk has an associated *likelihood* as the probability of the risk to happen. An *intentional risk* rises due to an anti-goal. In our illustrative example the risk S12: Electronic application is forged exists due to the anti-goal S2: Forgery from external attack. The directed edge with dotted line and black arrowhead from the anti-goal to the risk indicates an *increase likelihood* relation. Increase likelihood relation is only allowed between and anti-goal an a risk. *Incidental risks* are caused by external factors that can't be controlled, such as S13: Economic recession. *Impact* captures the effect of a risk on a goal. A risk may have different impacts on different goals. During modeling, the *severity* of the impact can be modeled in terms of absolute values such as monetary loss, or in terms of a relative scale can be used such as the five value scale used by CORAS [123]. In our illustrative example, S12: Electronic application is forged has an impact on the goal G6: Receive electronic application. In order to reduce the number of model elements, we omit the impact node between a risk and a goal, and use a directed dashed edge with a white arrowhead to represent *has-impact* relation. An anti-goal may increase the severity of an impact on a goal through *increase severity* relation. In LOP example S13: ID Document faked anti-goal increases the impact of S12 on G6.

Treatment layer includes *treatments* which are goals intended to mitigate the impact and/or reduce the probability of the risk. *Treatment refinement* links child treatments to parent treatment. Similar to other types of refinements, the parent treatment may have multiple refinements, each indicating an alternative way of achieving the parent. *Treatment objectives* are top treatments, and *treatment tasks* are the leaf treatments and they have a treatment cost attribute. Treatments mitigate the severity of an impact via *decrease severity* to recover after the occurrence of a risk. On the other hand treatments may also aim for prevention, decreasing the likelihood of the risk through *decrease likelihood* relation. In the example presented in Figure 6.2 the treatment layer is the bottom layer in which treatment are presented in hexagonal nodes. T1:Use digital signature treatment decreases the likelihood of S12: Electronic application is forged (prevention). T6:ire underwriters reduces the severity of the impact of S5: Debtor defaults on G7: Ensure loan repayment.

6.2 Evaluation of the Visual Notation

We follow the guidelines presented by Moody [16] for 'good' graphical notation. First, we use the horizontal position (planar variable) to distinguish layers, thick dashed lines clearly separates the three layers. The topmost layer is the asset layer, where stadium shaped nodes are goals, black filled dots are goal refinements and directed straight edges with black arrows are aggregation links of child goals to refinement nodes, and refines relations from the refinement nodes to parent goals. The straight edge is overloaded, yet the usage is inline with the conventional use, successfully conveys the semantics, yet decreases the

visual clutter. Another symbol overload in this layer is the representation of goals where we do not differentiate the symbols for business objectives, middle goals, and tasks (top, middle, and leaf goals, respectively). Again, in this situation the symbol overload does not cause ambiguity for it is easy to differentiate the concepts by checking the incoming and outgoing edges. Among these three constructs two of them have numerical attributes. We propose placing the values in square brackets after the text when modeling on paper or a board, and ability to toggle the display of the values when modeling with a software tool to keep visual simplicity.

The middle layer is the event layer. We keep the convention of using house-shaped nodes for representing risk. A semantically immediate solution would be to use fire shaped nodes for risks, however this reduces the text area available for the description, and might introduce difficulty when using analog tools. Instead we opt for red color to signify danger. Even though incidental and intentional risks can be differentiated by perceptual configuration due to the incoming edges from anti-goals, we increase the thickness of the border of incidental risks (use as a retinal variable) to emphasize the distinction. Anti-goals are represented as parallelograms in [43], we prefer using octagons for the text are can be used more effectively. For anti-goal refinement we use the same visual elements as the goal refinements to preserve graph economy by keeping the number of graphical notations low. Dashed edges with white double triangle arrowheads are used to represent the increase likelihood relation (from anti-goals to risk nodes), double triangle arrowhead signifies an increase and increases the visual distance from other edge types together with the line type and arrowhead color. Increase severity relation is represented by dotted edges with white double triangle arrowheads. Although the visual distance is one (line type) between the representation of this relation and the former one, target constructs are different (the target of the former is a risk node, whereas the latter aims to has impact edge), helping perceptual configuration. Has impact relation is represented by dashed and dotted line with a white triangle arrowhead, for the white the arrowhead is a construct from the event layer, yet this relation does not signify an increase so a single triangle is chosen. According to the principle of dual coding, complementing the graphical notation with text annotations convey information more effectively, yet using edge labels creates usual clutter, we propose using tool-tips for edge labels for software tool support.

The bottom layer is the treatment layer. We use cloud shape nodes for the conventional octagonal nodes used in [25] is from the same shape family (polygons) as nodes used in the event layer therefore they are too similar. We use the color blue to fill the nodes to smooth out the danger implied by the nodes in the event layer. There are two inter-dependencies whose sources are treatments, one is decrease likelihood whose target is a risk node and the other is decrease severity whose target is an impact edge. We use open triangle arrow heads to emphasize the decrease, vary the line types to increase the visual distance. Figure 6.3 summarizes our visual notation.

Three layered approach also helps managing the complexity of the models, enabling modularization, thus focusing on specific layers both in analog and digital tools. Considering all edge and node types, our notation exceeds the cognitive limit of six categories, yet this is an open problem present in other graphical notations [15].

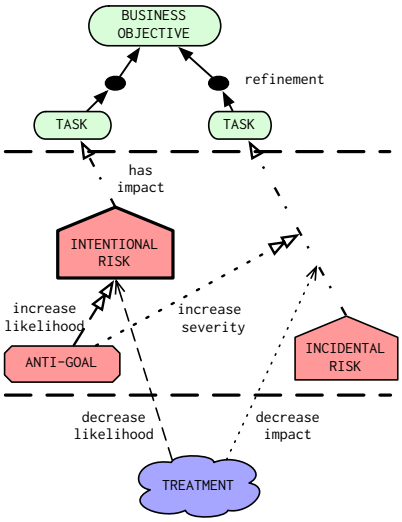


Figure 6.3: Legend for the visual notation

6.3 Risk Analysis

Risk models are instrumental to comprehend, capture, and convey goals of a system, risks that may harm these goals, anti-goals that may create these risks or increase the severity of their impact on system goals, and treatments that prevent or mitigate the risks. The structure of the models enables systematic analysis to extract information that is not trivial for the analyst. Risk analysts are interested in multiple questions to asses and manage risks. Risk assessment includes setting likelihood and impact of each risk, which is done during modeling, and calculating the risk factor as the product of these two values. Once the risk assessment is done, the analyst has several strategies to manage risk. The first strategy is to avoid risk, that is, selecting a solution with the minimal risk factor. The second strategy is to prevent risk, that involves utilizing treatments to decrease the likelihood of the risk (or prevent the risk from happening). The third strategy is to mitigate the impact of the risk through treatments. Apart from the risk management strategies, the analyst may be interested in the reward of the system-to-be, giving a higher priority to utility rather than risks. The budget for the project is a constraint of which the analyst should keep track, so the analyst may opt for minimizing the cost, trading cost for risk factor. Below we provide a list of questions that help risk analyst to asses risk and select a strategy for risk management.

Q1. Which solution has the maximum goal rewards?

Q2. Which solution has the minimum task costs?

Q3. Which solution has the minimum risk factor?

Q4. Which solution has the minimum risk likelihood?

Q5. Which solution has the maximum recovery?

Q6. Which solution has the maximum prevention?

Q7. Which solution has the minimum treatment cost?

SMT/OMT reasoning is a powerful scalable approach that combines multi-objective optimization and satisfiability. We transform our models into SMT-LIB language [149] to pass the models as input to the external solver OptiMathSAT [24].

$$((\wedge_{j=1}^{n} G_j) \leftrightarrow R) \wedge (R \rightarrow G_{parent}) \tag{6.1a}$$

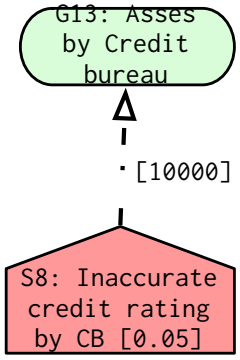
$$\text{Task Cost} = \sum_{G} ite(G_i, cost_{G_i}, 0) \tag{6.1b}$$

$$\text{Reward} = \sum_{G} ite(G_i, reward_{G_i}, 0) \tag{6.1c}$$

Formulas (6.1a) to (6.1c) presents the propositional encoding of the asset layer. Formula (6.1a) states that a refinement node R is achieved if and only if all sub-goals that are connected to the refinement node is satisfied $(((\wedge_{j=1}^{n} G_j) \leftrightarrow R))$, and the satisfaction of the refinement R implies the satisfaction of the parent goal G. Formulas (6.1b) and (6.1c) are numeric pseudo-boolean functions that define the Task Cost and Reward functions, respectively. $ite(G_i, cost_{G_i}, 0)$ denotes an if-$t$hen-$e$lse term, which is evaluated as cost of the task G_i, if G_i is achieved, 0 otherwise.

The transformation to SMT-LIB is trivial for Formula (6.1a). In order to represent the if-then-else term for the cost and reward functions, soft–assertions are used with the syntax as the following, `(assert-soft (not G3) :weight 5 :id task.cost)`. This soft–assertion introduces a cost when `G3` holds `true`. For all task costs the same id (`:id cost`) should be used. In order to find the total task cost of a solution, we declare a task cost function, which is the sum of the cost attributes of all tasks that holds true within a solution. This is achieved *i.* declaring a real function `(declare-fun taskcost () Real)`. *ii.* asserting the id of the task costs as the value of this function `(assert (= reward (- task.cost 0)))` (Formula (6.1b)). In order to find the solution with the minimum task cost `(minimize taskcost)` command is used. Similarly, to find the solution with the maximum reward, we define a reward function that returns the reward of the business objectives that are assigned to be `true` in a given solution, achieved by the following two statements: `(declare-fun reward () Real) (assert (= reward (- bo.reward 0)))`. The optimization command is stated as `(maximize reward)`.

The previous paragraph explains how to formalize our models and provide the input to the reasoner to answer Q1 and Q2, that are general concerns when reasoning on goal models. The third question aims to help decision making in presence of risks. A risk factor is the product of the likelihood and the severity of the impact of a risk. A lower risk factor indicates a safer decision. Consider two tasks G13: Assessed by credit Bureau and G14: Assessed in house from the illustrative example which are connected to the same



(a) Risk impacting a goal

$$risk.factor_{G_{13}} = 0.05 * 10000 \tag{6.2a}$$

$$rf_{G_{13},S_8} \leftrightarrow (G_{13} \wedge S_8) \tag{6.2b}$$

$$\text{Risk Factor} \tag{6.2c}$$

$$= \sum_{G} ite(rf_{G_i}, risk.factorG_i, 0) \tag{6.2d}$$

(b) Propositional Encoding

Figure 6.4: Risk impact on a goal

parent node through different refinement nodes, so they are alternatives to each other. G13 is under the risk of S8: Inaccurate credit rating by Credit Bureau with a likelihood of 5% which may lead to a loss of €10000 whereas G14 is also under the risk of S9:Mispredict monetary condition with a likelihood of 10% and a possible loss of €8000. Out of these two alternative tasks G13 is safer due to its lower risk factor (€500 versus €800). Selecting a solution with the minimal risk factor is a result of adopting risk avoidance strategy for risk management.

Figure 6.4a is a sample from the illustrated example presented in the previous section, Figure 6.4b presents the propositional encoding of the Risk Factor objective function and how risk factor of an individual task is calculated. Once the risk factor is calculated for each impact on a task, the numerical value is presented in the SMT-LIB language similar to the presentation of task cost, however here we introduce an additional model element, which is set to true when both the task and the risk holds true in the model. The reason is that the analyst may choose the ignore some risks if she believes that they will not occur at run-time. So the calculated risk factor values are aggregated in the Risk Factor function only if the task is included in the solution where the risk is considered in the model by the analyst, as shown by the second line of Figure 6.4b. In SMT-LIB, an individual risk factor is represented as `(assert-soft (not RF-G13-S8) :weight 500 :id task.risk.factor)`, and those individual values are aggregated by `(declare-fun riskfactor () Real) (assert (= riskfactor (- task.risk.factor 0)))`. Finally, the optimization scheme is set to `minimize riskfactor` to receive the solution that has the lowest risk factor.

Q4 searches for the solution that has the minimum chance of encountering a risk, regardless of the impact of the risk. Formula (6.3a) presents the associated risk likelihood of G_{13}. Similar to risk factor encoding, we introduce an additional element ($rl_{G_{13}}$) to ensure that only likelihood of risks that are considered by the analyst are included (Formula (6.3b)). The soft-assertion `(assert-soft (not rl-G13) :weight 0.05 :id task.risk.likelihood)` is the translation of the if-then-else pseudo boolean function used in Formula (6.3c). The objective function returns the sum of the values for tasks

that are included in the solution (tasks are set to be true). Here our assumption is that each impact edge comes from independent risks, therefore we sum the probabilities. More complex representation of risks in the event layer requires a change in the definition of this objective function in the future.

$$risk.likelihood_{G_{13}} = 0.05 \tag{6.3a}$$

$$rl_{G_{13},S_8} \leftrightarrow (G_{13} \wedge S_8) \tag{6.3b}$$

$$\text{Risk Likelihood} = \sum_{G} ite(G_i, risk.likelihood_{G_i}, 0) \tag{6.3c}$$

Q5 aims to find a solution including not only the stakeholder goals but also treatments to mitigate the impacts on the risks on goals. The mitigation of a treatment on an impact is represented by a decrease severity inter-dependency, and the numerical attribute indicates how much the treatment recovers the impact of the risk on a task. For the analysis, the important point is to achieve treatment objects that recovers tasks that are also included in the solution. From the illustrative example, it does not make sense to achieve the treatment 'verify ID documents with the government bodies' (T7) that decreases the severity of the impact to € 5000 just because there is a risk of fake documents (S6) unless the goal of verifying loan applications (G11), which is affected by the risk, is part of the solution. Therefore we introduce a new element for each decrease severity relation, which is set to be true when the target treatment and the source risk are true, and the task that is under that risk is included in the solution (Formula (6.4a)). For all decrease severity inter-dependencies, the total recovery is aggregated by the pseudo-boolean function presented in Formula (6.4c). In order to avoid misleading solutions due to the numerical values, it is important to put a constraint during modeling a mitigation on an impact cannot be higher than the severity of the impact of the risk.

$$tr - risk - goal_{T_7,S_8,G_{13}} \leftrightarrow (G_{13} \wedge S_8 \wedge T_7) \tag{6.4a}$$

$$rec_{T_7,S_8} = 5000 \tag{6.4b}$$

$$\text{Recovery} = \sum_{T,G,S} ite(tr - risk - goal_{T_i,S_j,G_k}, rec_{T_i,S_j}, 0) \tag{6.4c}$$

Total or partial prevention of a risk (decreasing the likelihood) decreases *i.* the risk factor of affected goals. The purpose of asking Q6 is to discover solutions including goals and treatments that has the minimum total risk factor. The main difference between Q3 and Q6 is the former does not consider treatments so the focus is to find a solution within the asset layer with the lowest risk factor. On the other hand Q6 considers treatments which lead to different solutions within the asset layer, for example among two alternative solutions with risk factor 3 and 5 the solution for Q3 is the first alternative, however if there is a treatment that reduces the second risk factor to 2, the solution for Q6 is the second alternative. In order to answer Q6, we first need to calculate the likelihood of the risk for each combination of treatments that decreases its likelihood. Then, for each new

likelihood, a new risk factor is calculated for each task that is affected by that risk. Once pre-processing calculates these values, we transform the model, similar to the approached presented in Figure 6.4. The main difference is that, for each calculated value, a new element is created, which is set to true when all considered treatments, risk, and the task is true, and other treatments that prevents that risk but not considered are false. The pseudo-boolean function for the prevented risk factor aggregates the new risk factor values for the new elements that holds true.

Algorithm 1 creates necessary SMT/OMT formulas. Before running the reasoner, we need to declare all possible likelihoods of a risk resulting from combining different treatments that reduce its likelihood. Lines 3 to 5 in in Algorithm 1 calculates the likelihood after prevention by all possible combinations of treatments that are related to a given risk.

```
1  initialization;
2  foreach task t of asset layer do
3  |  foreach impact i of risk r on t do
4  |  |  M = {Treatments decreases likelihood of r} foreach N ⊂ M do
5  |  |  |  calculate new likelihood of r ;
6  |  |  |  calculate risk factor after prevention;
7  |  |  |  declare a new element n;
8  |  |  |  n ↔ (t ∧ r ∧ {⋀ treatments ∈ N} ∧ ¬{⋁ treatments ∈ M \ N});
9  |  |  |  assert risk factor after prevention for n;
10 |  |  end
11 |  end
12 end
```

Algorithm 1: Minimizing total risk factor with prevention

Q7 focuses on the objective of minimizing the cost of treatments selected as part of the solution. The transformation from the models to SMT/OMT clauses are quite similar to the transformation of the asset layer. Formula (6.5a) presents the cost function for treatments.

$$\text{Treatment Cost} = \sum_{T} ite(T_i, cost_{T_i}, 0) \tag{6.5a}$$

Custom objective functions. Risk analysis plays an important role in project management for projects of all sizes from IT projects to governmental decisions. There are two main strategies to follow for risk management, the first strategy is to avoid risk, selecting solutions that have low risk factor. An analyst who is interested in risk avoidance tries to answer Q3. The second strategy is to accept risk, but either prevent it from happening (Q6) or mitigate its impact (Q5). Cost is an important factor for deciding the risk management strategy. In some cases risk prevention is cheaper than recovery, for example

preventive health measures are usually cheaper, that's the reason behind governments' investments in public vaccination campaigns to prevent an epidemic so that they would not need to face with the cost of treating infected masses. In some other cases, the opposite is true. Instead of keeping several backup servers all the time to prevent any service disruption, a bank may prefer to pay for a recovery service in case of a down-time. Safety critical systems give the highest priority to prevent risks, so they ask Q4 and Q6 first, and may give cost a lower priority. The analyst may also give the utility gained from the achievement of goals a high priority. A start-up may choose to roll a product to the market even though such move is associated with a high risk factor, yet the reward of their business objectives is too tempting. A thorough risk analysis requires combining these analyses, such as finding the solution with the minimal total cost of treatments and goals that has the minimal risk factor with maximum risk recovery. Depending on the objectives of the analysts several queries may be constructed.

Using SMT/OMT reasoning the analyst may set the optimization scheme for the solution in three ways. When there is a clear ordering of objectives, the analyst may opt for lexicographic optimization. The reasoner orders the solution according to the order of objectives, so the optimum solutions with respect to the first objective is returned first. If there are multiple solutions with the same objective value, the solutions are then ordered with respect to the second objective and so on. An example order of objectives is the following.

```
(minimize taskcost)
(minimize task.risk.factor.prevented)
(maximize reward)
```

The analyst may also construct a single multi-objective optimization function such as \$0.6 * task.cost + 0.4 * treatment.cost \$ and minimize or maximize that function. The third option is to combine these techniques together, having multiple objective functions ordered clearly.

Additional constraints. Constrained goal models and SMT/OMT reasoning allow applying additional constraints on solutions. For example, it is possible to set a constraint on the total budget of the solution, where the total budget is the sum of task and treatment costs. The reasoner returns the solution with maximal recovery whose total cost is less than € 100000 in the following example. Another example is for the consideration of risks, the analyst may choose to ignore risks whose likelihood is under a certain threshold, or allow risk factor up to a certain value. Additional constraints can be asserted as both hard and soft constraints in SMT-LIB.

```
(assert < totalCost 100000)
(maximize recovery)
```

Security risk analysis. Anti-goals [43] and obstacle analysis [158] are used to analyze risk that is caused by malicious actors. existing reasoning techniques can be applied in the event layer to determine whether a risk holds or not. In this chapter our focus is on

finding optimal solutions with respect to loss, cost, and rewards.

Opportunity analysis. An opportunity is a probabilistic situation that has a positive impact on stakeholder goals. Opportunity identification and analysis is conducted during new product development [159], our approach can be used to discover optimal solutions with respect to gain. Opportunity factor is calculated similar to risk factor, and it is desired to be maximized.

Scalability. We set up an experiment to test the scalability of our approach, running the experiment run on a Windows 64 bit machine with Intel(R) Core(TM) i7-3770 CPU 3.40Ghz and 8GB of RAM, and collected the reasoning time reported by OptiMathSAT to find the solution. For the experiments, we use OptiMathSAT version 1.3.10. Starting from an initial input model of 22 elements, we kept replicating the input model and connecting to the same parent node. The solver returned the results for the largest model with 17275 model elements uder 400 miliseconds showing a linear trend. Our results are parallel to those reported in [24], [64].

6.4 Chapter Summary

This chapter presents a multi-objective goal-oriented risk modeling and analysis framework by extending Asnar et al. [25] with constrained goal models [64]. We provide a meta-model enhanced with inter-dependencies in the model elements that determines the optimality of a solution. Our visual notation follows the design principles summarized in [16]. We transform goal–oriented risk models to SMT/OMT to discover optimal solution with respect to multiple objectives. Our proof-of-concept tool automatically maps the models into SMT/OMT formulas and retrieves solutions from the back–end reasoner. We investigate a pre-defined set of questions for analyzing risk yet our approach is flexible so that the analyst may define her own objective function and search for the optimal solution with respect to this custom objective function.

Limitations. SMT/OMT–based reasoning can handle linear arithmetic over rationals, it is a limitation for our framework. We overcome this limitation by pre-processing our models before transformation to SMT-LIB formulas, yet using combinations of treatments increases the number of formulas. Our graphical models suffer from the complexity problem of goal models. Building large goal models is an error-prone manual activity where the modeller loses grasp of what her model says as it gets bigger. We use layers to decrease the number of visible elements when the focus is on a single layer.

Chapter 7

Discussion, Conclusions, and Future Work

7.1 Fulfillment of Success Criteria

In this section we re-visit success criteria defined in Section 1.4 and state whether they are satisfied. We also discuss which contribution satisfies the criteria.

Adressing RQ_1, 'What is an effective process for modeling requirements and interactions of sociotechnical systems?'

$SC_{1.1}$ *Conduct modeling requirements and interactions in few steps.* This success criteria is fulfilled by a self-evaluation study and a case study.

- Protos approach defines and formalizes finite number of refinement rules to be applied during sociotechnical systems design.
- We have performed a self-evaluation study in the automobile insurance domain.
- We have conducted a case study with PhD students at the University of Trento in London Ambulance System.

$SC_{1.2}$ *Applicability to different domains.* We satisfied this criteria by applying the design process in different domains.

- Evaluation activities involve automobile insurance domain and London Ambulance system domain.

$SC_{1.3}$ *Formal semantics.* We provide formal semantics of the language in Chapter 3.

Addressing RQ_2 'How to explore designs of sociotechnical systems?'

$SC_{2.1}$ *Capture requirements and interactions.* We propose an i^* based requirements modeling language that captures social interactions as commitments.

$SC_{2.2}$ *Automatically create plans as a solution to the design problem.* We encode goal models in PDDL and use an external planner to generate plans.

Addressing RQ_3 'How to manage the iterative evolution of sociotechnical systems?'

$SC_{3.1}$ *Capture requirements and synergies for next iteration.* We provide a modeling language that captures relations and synergies among requirements. The relations and synergies are based on an industrial survey.

$SC_{3.2}$ *Support automated analysis for finding optimal solutions.* We encode goal models in SMT/OMT formulas so we benefit from OptiMathSat solver.

$SC_{3.3}$ *Provide solutions in acceptable time.* Our scalability analysis show that our approach scales with respect to the number of model elements, degree of model connectivity and number of alternative solutions.

Addressing RQ_4 'What is an effective method for risk analysis for sociotechnical systems'

$SC_{4.1}$ *Capture risk and synergies in requirements models.* We provide a conceptual model to capture requirements, risks, treatments, and the synergies between these model elements.

$SC_{4.2}$ *Support automated analysis for finding optimal solutions.* We encode goal models in SMT/OMT formulas so we benefit from OptiMathSat solver.

7.2 Conclusions

Design and evolution are two open challenges for sociotechnical systems [160]. Autonomy of systems, existence of local and global goals, possibly conflicting with each other, hazy boundaries between social and technical systems, and continual change are characteristics of sociotechnical system which also define problem characteristics.

We have proposed languages, analysis techniques, and methods to support sociotechnical system design and evolution. Key features of our contributions are *i.* Protos approach for sociotechnical system design, *ii. DEST* modeling language and a planning based exploration technique for discovering alternative design solutions, *iii.* a modeling language and a mapping to SMT/OMT formulas to apply SMT/OMT reasoning to the next release problem, *iv.* a modeling language, mapping to SMT/OMT formulas, and demonstration of several risk management strategies.

Protos: Protos is a requirements engineering approach for designing sociotechnical system. We re-formulate the requirements engineering problem stated by Zave and Jackson [17], and place specifications of social interaction (protocols) and multiple systems in the problem statement, as opposed to specifications for a machine as stated previously.

Protos design process relies on continual refinement of first class concepts: requirements can be refined into other requirements, systems (teams) can be refined into other systems, social refinements create and further refine social interactions, captured as commitments. The modeling process continues until all requirements are dealt with: either

satisfied through refinements, or a system takes final responsibility to satisfy them. The modeling language, concepts and refinement rules, are formalized.

Exploring design space: We propose a formalized, visual modeling language based on *i** to capture requirements of sociotechnical system. The meta-model includes social commitments to capture social interactions. In *i** framework social interactions are captured by dependencies, however dependencies are from one actor from another, and do not state whether the dependee accepts the dependency or whether there are any conditions for the dependum to be satisfied. In this sense, *DEST* provides a richer representation of social interactions. The meta-model also allows modeling temporal constraints between goals.

There could be several solutions within a model, so we explore these alternative solutions using AI planning. We define actions to AND and OR decompose goals, create and discharge commitments and encode these actions in PDDL language. An external planner is used to generate plans, a sequence of actions.

Handling iterative requirements evolution: Towards handling evolution of requirements in sociotechnical systems, we start from the well-established next release problem. All of the existing approaches are search–based software engineering methods, unable to guarantee the optimal results, mostly ignoring dependencies, relations and synergies between requirements.

We formulate the problem for goal models and propose a modeling language based on concepts and relations identified through an industrial survey. We transform goal models into SMT/OMT formulas, and allow flexible objective function specification. We use an external reasoner (OptiMathSat [24]) to discover the globally optimal solution.

Next Release Tool is a standalone application developed to support modeling and analysis activities for the next release problem. The tools provides a graphical editor and automatically converts goal models into SMT/OMT formulas. It feeds the formulas to OptiMathSat, retrieves and highlights the results on the graphical model.

Risk analysis: Uncertainty is a reality for different types of systems whether it indicates attacks from malicious agents, or unknown decisions of another system that affect others. For risk assessment, we propose a modeling language that captures synergies between goals, risks, and treatments. These concepts are modeled in three different layers, and relations and synergies among them are captured in the goal–risk models.

Unlike existing approaches in goal–driven risk management literature our focus is multi-objective risk analysis. We transform goal–risk models to SMT/OMT formulas ans rely on OptiMathSat for discovering optimal results. A system may adopt one or more risk management policies. We demonstrate how each policy transforms into an optimization schema as a guideline to analysts.

7.3 Ongoing and Future Work

Several future directions remain open for our research.

Empirical evaluation: One of the limitation of our work is that proposed languages and reasoning methods are not evaluated in real life case studies. In the short term our

aim is to evaluate our approach on the next release problem in two different settings. One study is based on data gathered from feature request system of an open source project where we build goal models based on the available data and discover the optimal solution and compare this solution to historical data on decisions during software development. The other study is planned to be run with a company where experts build or supervise building goal models and evaluate the solutions. Our aim is also to collect expert opinion about our language used to represent the next release problem.

Enhancing meta-models: Our research objective is to support design and evolution of sociotechnical systems yet the meta-model of final two components of our research do not capture individual systems an their goals. Although it is still possible to use these approaches for risk analysis and iterative evolution of sociotechnical systems, our short term objective is to enhance these meta-models to better capture characteristics of sociotechnical systems.

Improving tool support: The Next Release Tool is the most mature tool we have for our research. There is no implementation for Protos, and proof-of-concept implementations for exploring alternatives in sociotechnical systems design space and analyzing risk. For now our focus is on the Next Release Tool as it will be used for the empirical evaluations, but we plan to develop a more mature tool for risk analysis. In the long run, our objective is to implement a tool that allows distributed modeling and supports participatory design process.

Framework integration: Several components with respect to design and evolution of sociotechnical systems compose our framework, yet these components are not integrated with each other. Similar but different modeling languages are used depending on the problem we address so an ontology or mapping could be devised to integrate these languages so that output of a component could be directly used as input of another. For that reason integrated tool support is also important.

7.4 Future Lines of Research

Aside from our research agenda, our research open new research directions.

Serious games: Serious games and gamification have been applied to different domains to increase user engagement, organizational productivity, ease of use and usefulness of systems. Game-design elements and game principles can be applied to our framework for the benefits mentioned. For example, Protos methodology can be gamified to increase participation of the stakeholders to the modeling process, to encourage participants for more in depth and high-quality refinements, or to encourage building social commitments using badges, and points. Landmarks could be set for the modeling process as levels to be passed by the stakeholders.

Multi-criteria decision making: Our approach concerning risk analysis and the next release problem could be generalized to aid decision making in a more general context, or could be turned into a customizable framework, where end users specify the synergies exist in their domain and their effects, their constraints on solutions, and benefit from

automated SMT/OMT reasoning fro making their final decisions.

Crowd sourcing: We provide reasoning techniques that rely on quantitative or qualitative information. Although we refer to several existing techniques to extract these informations, how to get these values are beyond the scope of this thesis. The reasoning mechanisms could be combined with crowd-sourced models where the crowd provides synergies and intensity values to collectively help building models and automated reasoning provides the optimal solutions. One possible application domain is public administration, where the general public is provided with partial models and asked for their input. Citizen concerns could be directly presented on models, also as an exercise of participatory design, and then optimal solution is found by the reasoner.

Understanding human judgment: Humans take decision under certain assumptions and constraints based on multiple, possibly conflicting objectives. Now that we have powerful tools for capturing decision domains and discovering optimal solutions, it is possible to compare human judgment with the optimal solutions found by automated reasoners. Such research could answer questions like 'How well do human experts specify objective functions?', 'How do humans prioritize objectives?', 'How does expert judgment work compared to automated reasoning?'.

Bibliography

[1] F. E. Emery and E. L. Trist, "The causal texture of organizational environments", *Human relations*, vol. 18, no. 1, pp. 21–32, 1965.

[2] E. L. Trist and K. W. Bamforth, "Some social and psychological consequences of the longxwall method of coal-getting: An examination of the psychological situation and defences of a work group in relation to the social structure and technological content of the work system", *Human Relations*, vol. 4, no. 1, pp. 3–38, 1951.

[3] G. Ropohl, "Philosophy of socio-technical systems", *Techné: Research in Philosophy and Technology*, vol. 4, no. 3, pp. 186–194, 1999.

[4] L. V. Bertalanffy, *General System Theory: Foundations, Development, Applications*. George Braziller Inc., 1969.

[5] G. Fairtlough, *The three ways of getting things done*. Triarchy Press Limited, 2007.

[6] M. Satyanarayanan, "Pervasive computing: Vision and challenges", *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 10–17, 2001.

[7] K. Lyytinen and Y. Yoo, "Ubiquitous computing", *Communications of the ACM*, vol. 45, no. 12, pp. 63–96, 2002.

[8] A. Mowshowitz, "Virtual organization: A vision of management in the information age", *The Information Society*, vol. 10, no. 4, pp. 267–288, 1994.

[9] M. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions", in *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*, 2003, pp. 3–12.

[10] I. Foster, "The anatomy of the grid: Enabling scalable virtual organizations", *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[11] M. Henning, "Api design matters", *Queue*, vol. 5, no. 4, pp. 24–36, 2007.

[12] I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, and R. Paige, "Large-scale complex it systems", *Communications of the ACM*, vol. 55, no. 7, p. 71, 2012.

[13] E. Yu, "Towards modelling and reasoning support for early-phase requirements engineering", in *3rd IEEE International Symposium on Requirements Engineering*, Jan. 1997, pp. 236–235.

[14] B. Flyvbjerg and A. Budzier, "Why your it project may be riskier than you think", *Harvard Business Review*, vol. 89, no. 9, pp. 601–603, 2011.

[15] D. L. Moody, P. Heymans, and R. Matulevicius, "Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax", in *RE*, Aug. 2009, pp. 171–180.

[16] D. Moody, "The physics of notations: Toward a scientific basis for constructing visual notations in software engineering", *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[17] P. Zave and M. Jackson, "Four dark corners of requirements engineering", *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 1, pp. 1–30, 1997.

[18] M. P. Singh, "An ontology for commitments in multiagent systems", *Artificial Intelligence and Law*, vol. 7, no. 1, pp. 97–113, 1999.

[19] S. Browne and M. Kellett, "Insurance (motor damage claims) scenario", *Document Identifier D*, vol. 1, 1999.

[20] J. Kramer and A. L. Wolf, "Succeedings of the 8th international workshop on software specification and design", *SIGSOFT Softw. Eng. Notes*, vol. 21, no. 5, pp. 21–35, 1996.

[21] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language", 1998.

[22] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The next release problem", *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[23] Y. Zhang, M. Harman, and S. A. Mansouri, "The multi-objective next release problem", in *GECCO*, 2007, pp. 1129–1137.

[24] R. Sebastiani and P. Trentin, "Optimathsat: A tool for optimization modulo theories", in *Computer Aided Verification*, Springer Science + Business Media, 2015, pp. 447–454.

[25] Y. Asnar, P. Giorgini, and J. Mylopoulos, "Goal-driven risk assessment in requirements engineering", *Requirements Engineering*, vol. 16, no. 2, pp. 101–116, 2010.

[26] D. Ross and K. Schoman, "Structured analysis for requirements definition", *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 6–15, 1977.

[27] P. Zave, "Classification of research efforts in requirements engineering", *ACM Computing Surveys (CSUR)*, vol. 29, no. 4, pp. 315–321, 1997.

[28] B. Nuseibeh and S. Easterbrook, "Requirements engineering", in *Proceedings of the conference on The future of Software engineering - ICSE '00*, - 2000, nil.

[29] A. van Lamsweerde, "Requirements engineering in the year 00", in *Proceedings of the 22nd international conference on Software engineering - ICSE '00*, - 2000, nil.

[30] A. van Lamsweerde, "Goal-oriented requirements engineering: A guided tour", in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[31] A. I. Antón, W. M. McCracken, and C. Potts, "Goal decomposition and scenario analysis in business process reengineering", in *Advanced Information Systems Engineering*, ser. Advanced Information Systems Engineering. Springer Science + Business Media, 1994, pp. 94–104.

[32] A. Lapouchnian, "Goal-oriented requirements engineering: An overview of the current research", University of Toronto, Tech. Rep., 2005, `http://modald.googlecode.com/svn/masterthesis/Papers/Lapouchnian-Depth.pdf`.

[33] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach", *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[34] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5.

[35] H. A. Simon, *The sciences of the artificial*. MIT press, 1996.

[36] I. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling", in *International Requirements Engineering Conference*, 2010, pp. 115–124.

[37] J. Horkoff, F. B. Aydemir, F. Li, T. Li, and J. Mylopoulos, "Evaluating modeling languages: An example from the requirements domain", in *ER*, 2014, pp. 260–274.

[38] F.-L. Li, J. Horkoff, J. Mylopoulos, R. S. S. Guizzardi, G. Guizzardi, A. Borgida, and L. Liu, "Non-functional requirements as qualities, with a spice of ontology", in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, Aug. 2014, nil.

[39] F.-L. Li, J. Horkoff, A. Borgida, G. Guizzardi, L. Liu, and J. Mylopoulos, "From stakeholder requirements to formal specifications through refinement", in *Requirements Engineering: Foundation for Software Quality*, ser. Requirements Engineering: Foundation for Software Quality. Springer Science + Business Media, 2015, pp. 164–180.

[40] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition", *Science of Computer Programming*, vol. 20, no. 1-2, pp. 3–50, 1993.

[41] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering", *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 978–1005, 2000.

[42] A. Van Lamsweerde, S. Brohez, R. De Landtsheer, and D. Janssens, "From system goals to intruder anti-goals: Attack generation and resolution for security requirements engineering", in *In Proc. of RHAS'03*, 2003.

[43] A. Van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models", in *Proceedings of the 26th International Conference on Software Engineering*, IEEE Computer Society, 2004, pp. 148–157.

[44] A. Anton and C. Potts, "The use of goals to surface requirements for evolving systems", in *Proceedings of the 20th International Conference on Software Engineering*, Apr. 1998, pp. 157–166.

[45] C. Rolland, C. Souveyet, and C. Achour, "Guiding goal modeling using scenarios", *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1055–1071, 1998.

[46] E. S. K. Yu and J. Mylopoulos, "Understanding " Why " in Software Process Modelling, Analysis, and Design", 1994, pp. 159–168.

[47] F. Massacci, J. Mylopoulos, and N. Zannone, "Security requirements engineering: The si* modeling language and the secure tropos methodology", in *Advances in Intelligent Information Systems*, ser. Advances in Intelligent Information Systems. Springer Science + Business Media, 2010, pp. 147–174.

[48] D. Amyot and G. Mussbacher, "Urn: Towards a new standard for the visual description of requirements", in *Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science. Springer Science + Business Media, 2003, pp. 21–37.

[49] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology", *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[50] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, "Evaluating goal models within the goal-oriented requirement language", *International Journal of Intelligent Systems*, vol. 25, no. 8, pp. 841–877, 2010.

[51] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Formal reasoning techniques for goal models", in *Journal on Data Semantics I*, ser. Journal on Data Semantics I. Springer Science + Business Media, 2003, pp. 1–20.

[52] R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Simple and minimum-cost satisfiability for goal models", in *Advanced Information Systems Engineering*, ser. Advanced Information Systems Engineering. Springer Science + Business Media, 2004, pp. 20–35.

[53] Y. Asnar and P. Giorgini, "Modelling risk and identifying countermeasure in organizations", in *Critical Information Infrastructures Security*, ser. Critical Information Infrastructures Security. Springer Science + Business Media, 2006, pp. 55–66.

[54] Y. Wang, S. A. McIlraith, Y. Yu, and J. Mylopoulos, "An automated approach to monitoring and diagnosing requirements", in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering - ASE '07*, - 2007, nil.

[55] P. Giorgini, J. Mylopoulos, and R. Sebastiani, "Goal-oriented requirements analysis and reasoning in the tropos methodology", *Engineering Applications of Artificial Intelligence*, vol. 18, no. 2, pp. 159–171, 2005.

[56] J. Horkoff and E. Yu, "Finding solutions in goal models: An interactive backward reasoning approach", in *Conceptual Modeling - ER 2010*, ser. Conceptual Modeling - ER 2010. Springer Science + Business Media, 2010, pp. 59–75.

[57] E. Letier and A. van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering", *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 6, p. 53, 2004.

[58] V. Bryl, P. Giorgini, and J. Mylopoulos, "Designing cooperative is: Exploring and evaluating alternatives", in *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, ser. On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE. Springer Science + Business Media, 2006, pp. 533–550.

[59] ——, "Supporting requirements analysis in tropos: A planning-based approach", in *Agent Computing and Multi-Agent Systems*, ser. Agent Computing and Multi-Agent Systems. Springer Science + Business Media, 2009, pp. 243–254.

[60] ——, "Designing socio-technical systems: From stakeholder goals to social networks", *Requirements Engineering*, vol. 14, no. 1, pp. 47–70, 2009.

[61] Y. Asnar, V. Bryl, and P. Giorgini, "Using risk analysis to evaluate design alternatives", in *Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science. Springer Science + Business Media, 2007, pp. 140–155.

[62] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Representing and reasoning about preferences in requirements engineering", *Requirements Engineering*, vol. 16, no. 3, pp. 227–249, 2011.

[63] S. Sohrabi, J. A. Baier, and S. A. McIlraith, "HTN planning with preferences", in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 2009, pp. 1790–1797.

[64] M. C. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos, "Multi-objective reasoning with constrained goal models", `http://arxiv.org/abs/1601.07409`, 2016.

[65] J. J. Odell, H. V. D. Parunak, and B. Bauer, "Representing agent interaction protocols in uml", in *Agent-Oriented Software Engineering*, ser. Agent-Oriented Software Engineering. Springer Science + Business Media, 2001, pp. 121–140.

[66] J. R. Searle, *Speech acts: An essay in the philosophy of language*. Cambridge university press, 1969, vol. 626.

[67] F. for Intelligent Physical Agents (FIPA), "Fipa specification part 2: Agent communication language.", FIPA, Tech. Rep., 1997.

[68] M. Singh, "Agent communication languages: Rethinking the principles", *Computer*, vol. 31, no. 12, pp. 40–47, 1998.

[69] S. Kumar, M. J. Huber, and P. R. Cohen, "Representing and executing protocols as joint actions", in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 2 - AAMAS '02*, - 2002, pp. 543–550.

[70] C. Cheong and M. Winikoff, "Hermes: Designing goal-oriented agent interactions", in *Agent-Oriented Software Engineering VI*, ser. Agent-Oriented Software Engineering VI. Springer Science + Business Media, 2006, pp. 16–27.

[71] P. Yolum and M. P. Singh, "Commitment machines", in *Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science. Springer Science + Business Media, 2002, pp. 235–247.

[72] P. Yolum and M. P. Singh, "Flexible protocol specification and execution", in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 2 - AAMAS '02*, - 2002, nil.

[73] M. Shanahan, "The event calculus explained", in *Artificial Intelligence Today*, ser. Artificial Intelligence Today. Springer Science + Business Media, 1999, pp. 409–430.

[74] N. Desai, A. Mallya, A. Chopra, and M. Singh, "Interaction protocols as design abstractions for business processes", *IEEE Transactions on Software Engineering*, vol. 31, no. 12, pp. 1015–1027, 2005.

[75] R. Milner, *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[76] P. R. Telang and M. P. Singh, "Enhancing tropos with commitments", in *Conceptual Modeling: Foundations and Applications*, ser. Conceptual Modeling: Foundations and Applications. Springer Science + Business Media, 2009, pp. 417–435.

[77] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "An architecture for requirements-driven self-reconfiguration", in *Advanced Information Systems Engineering*, ser. Advanced Information Systems Engineering. Springer Science + Business Media, 2009, pp. 246–260.

[78] F. Dalpiaz, A. K. Chopra, P. Giorgini, and J. Mylopoulos, "Adaptation in open systems: Giving interaction its rightful place", in *Conceptual Modeling - ER 2010*, ser. Conceptual Modeling - ER 2010. Springer Science + Business Media, 2010, pp. 31–45.

[79] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "Modeling and reasoning about service-oriented applications via goals and commitments", in *Advanced Information Systems Engineering*, ser. Advanced Information Systems Engineering. Springer Science + Business Media, 2010, pp. 113–128.

[80] ——, "Reasoning about agents and protocols via goals and commitments", in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, ser. AAMAS '10, Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 457–464.

[81] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about datalog (and never dared to ask)", *IEEE Trans. Knowl. Data Eng.*, vol. 1, no. 1, pp. 146–166, 1989.

[82] M. Halpern, "The evolution of the programming system", *Datamation*, vol. 10, no. 7, pp. 51–53, 1964.

[83] M. M. Lehman, "Programs, cities, students- limits to growth?", in *Programming Methodology*, ser. Programming Methodology. Springer Science + Business Media, 1978, pp. 42–69.

[84] ——, "Programs, life cycles, and laws of software evolution", *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[85] ——, "Software engineering, the software process and their support", *Softw. Eng. J. UK*, vol. 6, no. 5, p. 243, 1991.

[86] ——, "Laws of software evolution revisited", in *Software Process Technology*, ser. Software Process Technology. Springer Science + Business Media, 1996, pp. 108–124.

[87] I. Skoulis, P. Vassiliadis, and A. Zarras, "Open-source databases: Within, outside, or beyond lehman's laws of software evolution?", in *Advanced Information Systems Engineering*, ser. Advanced Information Systems Engineering. Springer Science + Business Media, 2014, pp. 379–393.

[88] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan, "Types of software evolution and software maintenance", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 13, no. 1, pp. 3–30, 2001.

[89] D. Zowghi, A. K. Ghose, and P. Peppas, "A framework for reasoning about requirements evolution", in *Lecture Notes in Computer Science*, ser. Lecture Notes in Computer Science. Springer Science + Business Media, 1996, pp. 157–168.

[90] D. Zowghi and R. Offen, "A logical framework for modeling and reasoning about the evolution of requirements", in *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering*, Jan. 1997, pp. 247–257.

[91] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis, "Telos: Representing knowledge about information systems", *ACM Transactions on Information Systems*, vol. 8, no. 4, pp. 325–362, 1990.

[92] K. S. Barber and T. J. Graser, "Requirements evolution and reuse using the systems engineering process activities (sepa)", *Australasian Journal of Information Systems*, vol. 6, no. 2, 1999.

[93] S. Anderson and M. Felici, "Controlling requirements evolution: An avionics case study", in *Computer Safety, Reliability and Security*, ser. Computer Safety, Reliability and Security. Springer Science + Business Media, 2000, pp. 361–370.

[94] S. Anderson and M. Felici, "Quantitative aspects of requirements evolution", in *Proceedings 26th Annual International Computer Software and Applications*, - 2002, pp. 27–32.

[95] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution", *Information and Software Technology*, vol. 45, no. 14, pp. 993–1009, 2003.

[96] ——, "Erratum to "on the interplay between consistency, completeness, and correctness in requirements evolution"", *Information and Software Technology*, vol. 46, no. 11, pp. 763–779, 2004.

[97] N. A. Ernst, J. Mylopoulos, Y. Yu, and T. Nguyen, "Supporting requirements model evolution throughout the system life-cycle", in *2008 16th IEEE International Requirements Engineering Conference*, Sep. 2008, nil.

[98] N. A. Ernst, A. Borgida, and I. Jureta, "Finding incremental solutions for evolving requirements", in *2011 IEEE 19th International Requirements Engineering Conference*, Aug. 2011, nil.

[99] H. J. Levesque, "Foundations of a functional approach to knowledge representation", *Artificial Intelligence*, vol. 23, no. 2, pp. 155–212, 1984.

[100] R. Ali, F. Dalpiaz, P. Giorgini, and V. E. S. Souza, "Requirements evolution: From assumptions to reality", in *Enterprise, Business-Process and Information Systems Modeling*, ser. Enterprise, Business-Process and Information Systems Modeling. Springer Science + Business Media, 2011, pp. 372–382.

[101] S. Ghaisas and N. Ajmeri, "Knowledge-assisted ontology-based requirements evolution", in *Managing Requirements Knowledge*, ser. Managing Requirements Knowledge. Springer Science + Business Media, 2013, pp. 143–167.

[102] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, 1st. Wiley Publishing, 1998.

[103] G. Ruhe and D. Greer, "Quantitative studies in software release planning under risk and resource constraints", in *International Symposium on Empirical Software Engineering*, 2003, pp. 262–271.

[104] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach", *Inf. Softw. Technol.*, vol. 46, no. 4, pp. 243–253, Mar. 2004.

[105] O. Saliu and G. Ruhe, "Software release planning for evolving systems", *Innovations Syst Softw Eng*, pp. 189–204, 2005.

[106] T. L. Saaty, "How to make a decision: The analytic hierarchy process", *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, 1990.

[107] A. Ngo-The and G. Ruhe, "A systematic approach for solving the wicked problem of software release planning", *Soft Comput*, vol. 12, no. 1, pp. 95–108, 2007.

[108] G. Du and G. Ruhe, "Two machine-learning techniques for mining solutions of the releaseplanner™ decision support system", *Information Sciences*, vol. 259, pp. 474–489, 2014.

[109] Z. Pawlak, *Rough sets: Theoretical aspects of reasoning about data*. Springer Science & Business Media, 2012, vol. 9.

[110] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie, "Dependency networks for inference, collaborative filtering, and data visualization", *The Journal of Machine Learning Research*, vol. 1, pp. 49–75, 2001.

[111] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis, "Search based approaches to component selection and prioritization for the next release problem", in *2006 22nd IEEE International Conference on Software Maintenance*, Sep. 2006, nil.

[112] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[113] J. J. Durillo, Y. Zhang, E. Alba, and A. J. Nebro, "A study of the multi-objective next release problem", in *2009 1st International Symposium on Search Based Software Engineering*, May 2009, nil.

[114] J. J. Durillo, Y. Zhang, E. Alba, M. Harman, and A. J. Nebro, "A study of the bi-objective next release problem", *Empirical Software Engineering*, vol. 16, no. 1, pp. 29–60, 2010.

[115] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "Mocell: A cellular genetic algorithm for multiobjective optimization", *International Journal of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, 2009.

[116] H. Jiang, J. Xuan, and Z. Ren, "Approximate backbone based multilevel algorithm for next release problem", in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, - 2010, nil.

[117] J. Xuan, H. Jiang, Z. Ren, and Z. Luo, "Solving the large scale next release problem with a backbone-based multilevel algorithm", *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1195–1212, 2012.

[118] J. del Sagrado, I. M. del Aguila, and F. J. Orellana, "Ant colony optimization for the next release problem: A comparative study", in *2nd International Symposium on Search Based Software Engineering*, Sep. 2010, nil.

[119] N. Veerapen, G. Ochoa, M. Harman, and E. K. Burke, "An integer linear programming approach to the single and bi-objective next release problem", *Information and Software Technology*, vol. 65, no. nil, pp. 1–13, 2015.

[120] A. Pitangueira, P. Tonella, A. Susi, R. Maciel, and M. Barros, "Risk-aware multi-stakeholder next release planning using multi-objective optimization", in *REFSQ*, 2016.

[121] T. Bedford and R. Cooke, "Probabilistic risk analysis: Foundations and methods", *The Syndicate of the Press of University of Cambridge*, 2001.

[122] J. B. Bowles, "The new sae fmeca standard", in *Reliability and Maintainability Symposium, 1998. Proceedings., Annual*, IEEE, 1998, pp. 48–53.

[123] R. Fredriksen, M. Kristiansen, B. A. Gran, K. Stølen, T. A. Opperud, and T. Dimitrakos, "The coras framework for a model-based risk management process", in *Computer Safety, Reliability and Security*, ser. Computer Safety, Reliability and Security. Springer Science + Business Media, 2002, pp. 94–105.

[124] M. Feather, "Towards a unified approach to the representation of, and reasoning with, probabilistic risk information about software and its system interface", in *15th International Symposium on Software Reliability Engineering*, Nov. 2004, pp. 391–402.

[125] N. Mayer, E. Dubois, and A. Rifaut, "Requirements engineering for improving business/it alignment in security risk management methods", in *Enterprise Interoperability II*, ser. Enterprise Interoperability II. Springer Science + Business Media, 2007, pp. 15–26.

[126] R. Matulevicius, H. Mouratidis, N. Mayer, E. Dubois, and P. Heymans, "Syntactic and semantic extensions to secure tropos to support security risk management", *J. UCS*, vol. 18, no. 6, pp. 816–844, 2012.

[127] H. Mouratidis and P. Giorgini, "Secure tropos: A security-oriented extension of the tropos methodology", *Int. J. Soft. Eng. Knowl. Eng.*, vol. 17, no. 02, pp. 285–309, 2007.

[128] A. Siena, M. Morandini, and A. Susi, "Modelling risks in open source software component selection", in *Conceptual Modeling*, ser. Conceptual Modeling. Springer Science + Business Media, 2014, pp. 335–348.

[129] D. Costal, L. López, M. Morandini, A. Siena, M. C. Annosi, D. Gross, L. Méndez, X. Franch, and A. Susi, "Aligning business goals and risks in oss adoption", in *Conceptual Modeling*, ser. Conceptual Modeling. Springer Science + Business Media, 2015, pp. 35–49.

[130] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models", in *Conceptual Modeling - ER 2002*, ser. Conceptual Modeling - ER 2002. Springer Science + Business Media, 2002, pp. 167–181.

[131] A. van Lamsweerde, "Reasoning about alternative requirements options", in *Conceptual Modeling: Foundations and Applications*, ser. Conceptual Modeling: Foundations and Applications. Springer Science + Business Media, 2009, pp. 380–397.

[132] A. K. Chopra, F. Dalpiaz, F. B. Aydemir, P. Giorgini, J. Mylopoulos, and M. P. Singh, "Protos: Foundations for engineering innovative sociotechnical systems", in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, Aug. 2014, nil.

[133] M. P. Singh, "Semantical considerations on dialectical and practical commitments.", in *AAAI*, vol. 8, 2008, pp. 176–181.

[134] R.-J. Back and J. Wright, *Refinement calculus: a systematic introduction*. Springer Science & Business Media, 2012.

[135] A. J. I. JONES and M. SERGOT, "A formal characterisation of institutionalised power", *Logic Journal of IGPL*, vol. 4, no. 3, pp. 427–443, 1996.

[136] V. E. S. Souza, "An Experiment on the Development of an Adaptive System based on the LAS-CAD—Incomplete Version1", DISI, University of Trento, Tech. Rep., 2012.

[137] A. K. Chopra and M. P. Singh, "Colaba: Collaborative design of cross-organizational processes", in *2011 Workshop on Requirements Engineering for Systems, Services and Systems-of-Systems*, Aug. 2011, pp. 36–43.

[138] M. P. Singh, "Norms as a basis for governing sociotechnical systems", *TIST*, vol. 5, no. 1, pp. 1–23, 2013.

[139] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning.", in *AAAI*, vol. 87, 1987, pp. 677–682.

[140] G. Gans, M. Jarke, S. Kethers, and G. Lakemeyer, "Modeling the impact of trust and distrust in agent networks", in *Proc. of AOIS'01*, 2001, pp. 45–58.

[141] F. B. Aydemir, P. Giorgini, J. Mylopoulos, and F. Dalpiaz, "Exploring alternative designs for sociotechnical systems", in *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, May 2014, pp. 1–12.

[142] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating preferences into goal models for requirements engineering", in *2010 18th IEEE International Requirements Engineering Conference*, Sep. 2010, nil.

[143] D. S. Weld, "Recent advances in {AI} planning", *AI Magazine*, vol. 20, no. 2, p. 93, 1999.

[144] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements", *IEEE Softw.*, vol. 14, no. 5, pp. 67–74, 1997.

[145] K. Regan and C. Boutilier, "Regret-based reward elicitation for markov decision processes", in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, ser. UAI '09, Montreal, Quebec, Canada: AUAI Press, 2009, pp. 444–451.

[146] A. Gerevini and D. Long, "Plan constraints and preferences in pddl3", *The Language of the Fifth International Planning Competition. Tech. Rep. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, vol. 75, 2005.

[147] J. Horkoff and E. Yu, "Analyzing goal models", in *SAC*, 2011.

[148] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. och Dag, "An industrial survey of requirements interdependencies in software product release planning", in *IEEE International Symposium on Requirements Engineering*, 2001, pp. 84–91.

[149] C. Barrett, P. Fontaine, and C. Tinelli, "The SMT-LIB Standard: Version 2.5", Department of Computer Science, The University of Iowa, Tech. Rep., 2015, Available at `www.SMT-LIB.org`.

[150] J. Horkoff, E. Yu, and L. Liu, "Analyzing trust in technology strategies", in *International Conference on Privacy, Security and Trust*, 2006.

[151] A. P. Sage and Y. Y. Haimes, *Risk modeling, assessment, and management*. John Wiley & Sons, 2015.

[152] G. Roy and T. Woodings, "A framework for risk analysis in software engineering", in *Proceedings Seventh Asia-Pacific Software Engeering Conference. APSEC 2000*, Dec. 2000, pp. 441–445.

[153] S. Cornford, M. Feather, V. Heron, and J. Jenkins, "Fusing quantitative requirements analysis with model-based systems engineering", in *14th IEEE International Requirements Engineering Conference (RE'06)*, Sep. 2006, nil.

[154] E. Paja, F. Dalpiaz, and P. Giorgini, "Modelling and reasoning about security requirements in socio-technical systems", *Data & Knowledge Engineering*, vol. 98, no. nil, pp. 123–143, 2015.

[155] L. Liu, E. Yu, and J. Mylopoulos, "Security and privacy requirements analysis within a social setting", in *Journal of Lightwave Technology*, Sep. 2003, pp. 151–161.

[156] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone, "Requirements engineering for trust management: Model, methodology, and reasoning", *International Journal of Information Security*, vol. 5, no. 4, pp. 257–274, 2006.

[157] T. Wetzel, *States of affairs*, `http://plato.stanford.edu/archives/fall2008/entries/states-of-affairs/`, Last accessed Fri Jan 29 13:40 2016, 2003.

[158] L. Duboc, E. Letier, and D. S. Rosenblum, "Systematic elaboration of scalability requirements through goal-obstacle analysis", *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 119–140, 2013.

[159] P. A. Koen, "The fuzzy front end for incremental, platform and breakthrough products and services", *PDMA Handbook*, pp. 81–91, 2004.

[160] P. Feiler, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, and K. Wallnau, "Ultra-large-scale systems the software challenge of the future", Software Engineering Institute, Carnegie Mellon University, Tech. Rep., 2006.