ACQUIRING AND REASONING ABOUT VARIABILITY IN GOAL MODELS

by

Sotirios Liaskos

A thesis submitted in conformity with the requirements for the degree of Philosophy of Science
Graduate Department of Computer Science
University of Toronto

Copyright © 2008 by Sotirios Liaskos
Abstract

Acquiring and Reasoning about Variability in Goal Models

Sotirios Liaskos
Philosophy of Science
Graduate Department of Computer Science
University of Toronto
2008

One of the most essential parts of any software requirements analysis effort is the exploration of alternative ways by which stakeholder problems can be solved. Systematic modeling and analysis of requirements variability allows better decision making during the early requirements phase and substantiates design choices pertaining to the configurability aspect of the system-to-be. This thesis proposes the use of goal models for capturing and reasoning about requirements variability. The goal models we adopt consist of AND/OR decompositions of stakeholder goals and express alternative ways by which stakeholders may wish to achieve them. By capturing goal variability using such models, we propose a shift of focus from variability of the software design, to variability of the problem that the design is intended to solve. This way, we ensure that every important variation of the problem is identified and analyzed before variations of the solution are specified.

The thesis exploits opportunities that arise from this new viewpoint. Firstly, a variability-intensive goal decomposition process is proposed. The process is based on associating each high-level goal to a set of variability concerns that must be addressed through decomposition. We introduce a universal categorization of such concerns and also show how domain-specific variability concerns can be identified by annotating domain corpora. Concern-driven decomposition offers a structured way of thinking about problem variability, while systematizing its identification process. Further, an expressive LTL-based preference language is introduced to support leverage of large spaces of goal alternatives. The language allows the expression of preferences over behavioral and qualitative properties of solutions and a reasoning tool allows
the identification of alternatives that satisfy these preferences. This way, individual stakeholders can get the solution that exactly fits their needs in a particular situation, through simply specifying desired high-level characteristics of these solutions. Finally, a framework for connecting alternatives at the goal level to alternative configurations of common desktop applications is presented. The framework shows how a vast number of configurations of a software application can be evaluated and ranked with respect to a small number of quality goals that are more intuitive to and comprehensible by end users.
Acknowledgements

This doctoral thesis presents the results of a five-year research effort that took place in the Department of Computer Science, at the University of Toronto. During that time, I had the opportunity to meet and collaborate with many wonderful people, without the support of whom this thesis would have never been possible. I would like to mention as many of them as I can here knowing that, alas, it is very probable I forget some of them.

The thesis is a natural continuation of my master’s project, which was an initial exploration of the relationship between goal variability and user’s skills and preferences. That was made possible through a tight collaboration with bowen hui (lower case deliberate), fellow PhD student. My collaboration with her signified my first exposure on how research happens and how scientific papers are written; she has been an invaluable mentor for me.

Soon afterwards, I joined a team to study the effect of goal modeling to design. Dr. Yijun Yu, now Assistant Professor at the Open University, UK, post-doctoral collaborator at that time, Alexei Lapouchnian, fellow PhD student, and Prof. Julio Cesar Sampaio do Prado Leite, Associate Professor at the PUC Rio, Brazil, visiting the University of Toronto at that time. Fellow PhD student, Wendy Liu, would participate at times. Our long and sometimes heated discussions had a great influence on my thinking about goal and design variability.

At the same period, my colleagues and I formed a reading group to study and discuss papers in the general area of software engineering. We called it “gadg” (it has always been obscure as to what the term stands for). Some of the frequent participants at that time were Alexei Lapouchnian, Rick Salay, Neil Ernst, Wendy Liu, Mehrdad Sabetzadeh and Shiva Nejati, Yiqiao Wang, Jennifer Horkoff, all fellow PhD students, Dr. Yijun Yu, Dr. Markus Strohmaier, visiting scholar at that time, now Assistant Professor at the Graz University of Technology, Austria, and many others. These meetings helped me a lot to form a better idea of our area and the foundations of the things we were researching. Apart from “gadg”, the discussion with most of the colleagues above would be an everyday ritual in the aisles and our little cubicles in the Bahen Center for Information Technology, and would be maintained off-line through mutual review of each other’s written work. In that field, Mehrdad Sabetzadeh has always been a very accurate and insightful commenter on our draft papers.

Much of the empirical evaluation reported in this thesis was possible thanks to the Precarn/IRIS project titled “Intelligent system requirements for cognitively impaired individuals”. In the context of this project, I was given the opportunity to acquire requirements from a variety of health professionals through a series of meetings and interviews that took place in Halifax.
I am grateful to Paige King, M.D. and PhD researcher at the Division of Geriatric Medicine of Dalhousie University, Halifax, for helping me arrange those valuable meetings and for being an important information source herself. I am also thankful to Dr. David Evans, the coordinator of the project, who greatly facilitated the whole process.

At the University of Toronto, I had the opportunity to interact with extremely knowledgeable and capable professors who shaped the direction of my research, sometimes with minimal interaction. For example, Prof. Graeme Hirst and Prof. Ravi Balakrishnan deeply influenced the direction of my thought, through a single e-mail and an ad-hoc half-hour discussion. Amongst those who attended my work more closely, Prof. Marshia Chechik has been an important source of inspiration and useful criticism several times. I am also grateful to her for offering me detailed and insightful comments on earlier version of the thesis. Prof. Bashar Nuseibeh, from the Open University, UK, acting as my external thesis appraiser, offered precious feedback on various aspects of the work. Prof. Sheila McIlraith has been a tremendous source of inspiration and support for major part of this thesis. Her impressive mastery of her field and her focus on questioning all my assumptions helped me a lot to make this thesis stronger.

I was, furthermore, honored to have Prof. Eric Yu and Prof. Steve Easterbrook as the two formal members of my PhD advisory committee. Their involvement was far from just “formal”, and it does not concern only this thesis, the publications and their invaluable advising and commenting on them. Steve and Eric lead an exemplar research and learning environment at the University of Toronto, whose influence has been tremendous in my effort to understand what research is about, and how it can be meaningfully conducted in our field.

The thesis was supervised by Prof. John Mylopoulos. Very few people have influenced me as deeply as he did, at all levels. Infinite patience, absolute freedom and trust, steady and unconditional support and encouragement and, of course, a continuous flow of amazing ideas and inspiration is what I experienced seven years of working with him. He is a model supervisor. Like so many others, I am simply proud to have met and worked with Dr. Mylopoulos. I hope the result comes up to his expectations.

This doctoral project has been funded by the National Sciences and Engineering Research Council (NSERC) of Canada, Bell University Laboratories and the Precarn/IRIS project “Intelligent system requirements for cognitively impaired individuals”.

v
# Contents

1 Introduction

1.1 Overview .................................................. 1
1.2 From Software Requirements to Stakeholder Goals ............ 2
1.3 Goal Variability ............................................ 4
   1.3.1 Goals and Alternatives ............................... 4
   1.3.2 Using Goal Variability for Designing Software .......... 6
   1.3.3 One-of development .................................... 6
   1.3.4 Goal Variability and Software Customization ............. 7
1.4 A Framework for Acquiring and Reasoning about Goal Alternatives .. 8
   1.4.1 Acquiring Alternatives ............................... 9
   1.4.2 Selecting Alternatives through Preference Specification .... 9
   1.4.3 From Goal Alternatives to Software Configurations ....... 10
   1.4.4 Structure of Thesis ................................... 11

2 Requirements Variability: State of Research and Practice ........ 12

2.1 The Literature Landscape .................................... 12
2.2 Origins of Variability ...................................... 13
   2.2.1 Traditional Reuse-centered Domain Analysis .............. 14
   2.2.2 Requirements-Driven Acquisition of Variability .......... 16
2.3 Variability Modeling ....................................... 18
   2.3.1 Variability Modeling and Object Oriented Analysis ....... 18
   2.3.2 Feature Modeling ...................................... 20
   2.3.3 Variability and other Requirements Modeling Approaches .... 21
   2.3.4 Variability and Problem Frames .......................... 23
   2.3.5 The Variation Point model .............................. 23
2.4 Variability Binding and Product Configuration .................. 24
2.4.1 Feature-based product derivation ........................................... 24
2.4.2 DSLs ................................................................................. 25
2.4.3 Alternatives Analysis in Requirements ................................. 25
2.4.4 Product Configuration Technologies ................................... 26
2.5 Conclusion: Open Research Questions .................................. 26

3 Goal Models and Goal Variability ............................................. 28
3.1 Overview ............................................................................. 28
3.2 Goals in Requirements Engineering ...................................... 28
3.3 A Basic Goal Modeling Language ........................................ 30
  3.3.1 Fundamental Concepts ..................................................... 30
  3.3.2 Hard-goal subgraph ......................................................... 32
  3.3.3 Soft-goal subgraph ......................................................... 32
3.4 Label Propagation ................................................................. 37
3.5 Goal Alternatives ................................................................. 38
3.6 Summary ............................................................................. 40

4 Acquiring Goal Variability ....................................................... 41
4.1 Overview ............................................................................. 41
4.2 Goal Variability: a Closer Look .......................................... 42
  4.2.1 On the Notion of OR-decomposition of Goals .................... 42
  4.2.2 Origins and Meaning of Variability in Requirements .......... 42
  4.2.3 Non-intentional Variability ............................................... 44
4.3 Motivating Example and Research Approach ...................... 45
4.4 The Goal Modeling Language .............................................. 46
4.5 Variability Concerns for Goal Decomposition ..................... 48
4.6 Constructing Problem-specific Variability Frames ................ 51
4.7 Domain Variability ............................................................. 53
4.8 Variability-Intensive Decomposition and Analysis ............... 54
4.9 In Practice ......................................................................... 61
4.10 Summary .......................................................................... 62

5 Goal Alternatives and Preference Specification ..................... 64
5.1 Overview ............................................................................. 64
5.2 Fitness Criteria for Goal Models .......................................... 65
Chapter 1

Introduction

1.1 Overview

This thesis introduces a framework for identifying and reasoning about variability of stakeholder goals during requirements analysis. The framework consists of a language for constructing high-variability goal models and a set of methods that allow building such models, analyzing them, as well as connecting them with the customization infrastructures of existing software systems. These are intended to support the requirements engineering process by allowing requirements analysts to identify and explore variations of stakeholder problems, to identify alternative ways by which these problems can be solved, as well as to understand which alternative is most suitable given certain high-level criteria. This, in turn, can support the definition and exploration of alternative designs and customizations that can potentially address the problem-at-hand.

In this first chapter we introduce the notion of goals and discuss how they are placed in the current state of knowledge in Requirements Engineering (Section 1.2). We informally introduce the fundamental elements of goal models, show how variability emerges by developing such models and illustrate why this can be useful in several contexts (Section 1.3). We then provide an overview of our framework and describe its basic elements (Section 1.4). A description of how our detailed presentation of our contributions is organized in the chapters of this thesis closes this chapter (Section 1.4.4).
1.2 From Software Requirements to Stakeholder Goals

It is widely accepted in the Software Engineering research community that requirements engineering plays a critical role to the overall success of software development endeavors ([92, 23, 115]). The requirements engineering process aims at discovering what the purpose of the system(s)-to-be is, by understanding the needs and preferences of stakeholders and representing them in a form that allows several kinds of subsequent analysis ([92]). Thus, research in requirements engineering encompasses a wide variety of topics pertaining to the understanding of processes for identifying and analyzing stakeholder needs, including elicitation, negotiation, modeling, specification, verification and documentation. The subject matter in all these processes is the very concept of requirement. Hence, significant research effort has been dedicated towards understanding what requirements really are and how they evolve throughout the corresponding engineering process.

The traditional understanding of requirements was that they are a high-level description of features of software systems. These features are functions, behaviors, interfaces, data and other characteristics that the software exhibits. Thus, requirements were understood as a set of “The system shall ...” statements describing these characteristics of the system-to-be. By identifying such statements of how the system is to be used, users are able to describe what they need without reference to design technicalities, while designers have the necessary set of constraints that their design needs to satisfy.

Soon, however, the need to express what precedes and necessitates these specifications emerged. Why “shall the system do [this]” and not “[that]” or nothing at all? Are users and other non technically proficient stakeholders really in the position to describe or even comprehend a given description of the system-to-be? Or should one focus on what they really want to be true in their world rather than their (potentially flawed) vision of a technical artifact? Today, it is indeed widely accepted that software features emerge at rather later stages of the analysis process, while requirements are instead a concept that exists before, beyond and independent of how the system-to-be is defined.

Critical to this evolution of our thinking about the meaning of requirements has been the idea of separation between problem and solution. The former constitutes the desire of stakeholders to depart from the present state of the world, which is for them problematic or suboptimal, towards an envisioned one, in which the problematic circumstances are no longer present. The solution, on the other hand, is a description of the software system that will enable and operate in the envisioned state. Obviously, in order to guarantee that the provided solution
will indeed allow stakeholders to reach the state of the world they want to reach, it is necessary to fully understand what that state is. In the Requirements Engineering research community, a popular depiction of this separation has been Michael Jackson’s model of two distinct yet overlapping domains: the machine domain and the domain of the machine’s environment ([64]). While the former includes all the phenomena that are internal to the machine, the latter are phenomena that exist in the machine’s environment, in a way that it may or may not be able to sense or influence them. Requirements reside in the environment, and, specifically, they are “conditions over [its] states and the events” ([64]). Thus, the purpose of the software development effort is the construction of a machine that interacts with its environment in such a way, so that the desired conditions are met. Hence, the requirements engineering process begins with the identification of these conditions, which is what we called the problem.

Nevertheless, the identification and representation of the relevant aspects and constituents of the problem as well their in between relationships is itself a non-trivial process. How can one go about producing a problem statement? What if there are several problem statements? We obviously need to understand how alternative problem descriptions relate to each other, and in light of that, explore how acquisition of one problem description potentially leads to the discovery of another. In this context, the notion of goal has successfully served the need for eliciting and representing high-level descriptions of the problem and how they relate, contribute to or conflict with each other. Goals are states of the world that one or more stakeholders want to be reached and/or maintained. While this definition greatly resembles Jackson’s notion of requirements, the power of using goals to understand and analyze problems lies in our ability to express relationships among them, through the construction of goal models ([88, 31, 4, 126]). Goal models allow representations of how different statements about the desired state of the world relate to each other. There are several ways by which such statements can relate. For example they may belong to different yet interdependent stakeholders, represent different aspects of the same problem or represent different problems related in some aspect. Goal models offer a rich set of relationship types to represent wide range of such interdependency types.

Consider, for instance, the problem of scheduling meetings, which we wish to analyze for purposes of, for example, building a system that supports scheduling such. Part of the meeting scheduling process is to Invite Participants to a Meeting. This is a goal because it describes a state of the world the meeting initiator wants to accomplish. Now, consider also the goal Have Invitations Sent by Secretary. Comparing the two goals, we observe that the latter would constitute (one of) the means to achieve the former.
Thus, the two goals are potentially associated with a means-ends relationship ([126]), in that the latter goal solves the former (i.e. shows how it can be satisfied), while the former explains the latter (i.e. shows why it was introduced).

As a second example of a relationship type between goals, consider again the goal Have Invitations Sent by Secretary and the goal Secretary Not Be Busy. Fulfillment of the former influences to some degree the satisfaction of the other. Indeed, that the secretary has to send invitations for the meeting implies that he will be busier than he would be if he did not have to do so. If, instead, the first goal were Have Invitations Sent by System, there would be no influence to the satisfaction or denial of the goal Secretary Not Be Busy. Thus, a contribution relationship amongst goals occurs when the fulfillment or denial of one goal influences (contributes to) the fulfillment or denial of some other goal.

Goal models allow visual representation of such means-ends and contribution relationships leading to several interesting types of goal graphs, which, in turn, allow analysts and stakeholders to better comprehend the constituents and structure of the problem at hand. While there are many more different types of such relationships in the literature, the means-ends as well as the contribution relationship have an important position in our framework. In particular, we use goal models constructed based on such relationships in order to explore and evaluate different ways by which problems can be transformed into solutions. We present the fundamentals of this idea in the next section.

1.3 Goal Variability

1.3.1 Goals and Alternatives

We saw above that in goal models we can represent the fact that the fulfillment of one high-level goal is realized by the fulfillment of a goal of lower level, by saying that the latter is a means to achieve the former. It is interesting to observe that, in fact, each high-level goal may be fulfilled by several alternative sub-goals. For example the goal Invite Participants to a Meeting can be fulfilled by the goal Have Invitations Sent by Secretary, but also by goals Have Invitations Sent by Meeting Initiator, Have Secretary Call Invitees, or Have Initiator Call Invitees. All these four constitute alternative ways by which the parent goal can be satisfied. In turn, there may be several different ways to Have Invitations Sent by Secretary; for example electronic or land mail may be two alternative options. But this extension doubles the num-
ber of alternative combinations of subgoals that can fulfill the higher level goal *Invite Participants to a Meeting*. Obviously, as more and more goals are iteratively analyzed into sets of alternative sub-goals that can fulfill them, the alternative solutions for problem expressed in the top-level goals grow exponentially. The term *goal variability* refers exactly to the fact that there may be more than one, and, in practice, a large space of alternative ways by which top level goals can be fulfilled.

The emergence of variability in goal models is natural, in that we rarely construct such models to refer only to a particular solution to the problem-at-hand. Instead a goal model is useful if it identifies a space of alternatives for fulfilling a root-level goal. As an example, consider, again, the goal *Invite Participants to a Meeting*. In practice, for this goal to be completely specified we also need to understand more things about the agent who wants to fulfill it and the special circumstances she is in. For example, she may want to invite her colleague from her home, her office or from the airplane. She may desire, at the same time, to maintain maximum privacy or, alternatively, maximum efficiency while her colleague is invited. She may or may not have a physical of cognitive disability preventing her from communicating in certain forms. The truth is that we may not know these details, because our goal model is intended to apply under a variety of circumstances. Thus, a high-level goal such as *Invite Participants to a Meeting*, necessarily introduces a family of *problem variants*, by leaving several details and parameters unspecified.

At the same time, for every given goal, different ways to fulfill it (alternative means-ends) can almost always be introduced and, moreover, debated subject to their characteristics. These characteristics can be understood as contributions of alternative goals to the satisfaction of other goals. In our meeting invitation example, involvement of a secretary adds a note of formality and distance (hence, contributes towards the satisfaction of the goal *Maintain Formality*), but burdens the secretary (hence, contributes against the satisfaction of the goal *Secretary Not Be Busy*). A personal phone call is always efficient but may be disruptive (implying respective contributions to goals such as *Maintain Efficiency* and *Avoid Disruptions*). Note that such characteristics may not always be such contributions to other goals but conditions under which alternative goals can be fulfilled. For example a personal phone call is impossible on an airplane or to a person with hearing disorder. Nevertheless, in either case, not knowing the individual circumstances surrounding the invitation, that is the actual problem variant, we maintain both alternatives in our model, which grows, this way, in the number of alternatives it supports. We also keep track of the characteristics of each variant to facilitate our comparison when more details are available (e.g. as to whether
the particular participant has a hearing impairment, or whether the particular meeting initiator is ready to sacrifice efficiency in order to avoid disruptions of the invitees).

Therefore, a goal model implies alternative ways by which individual stakeholders under arbitrary environmental circumstances can possibly achieve the top level goals. If the individual stakeholder and her characteristics, as well as the particular circumstances are known, the top level goal is better defined, and the alternatives that better serve the refined goal can be chosen. We argue below that this property of goal models is very useful for both understanding the space of possibilities of one-of software development projects, and, more importantly for understanding the variability of a problem domain for the purpose of designing customizable software.

1.3.2 Using Goal Variability for Designing Software

1.3.3 One-of development

Goal variability, and its natural emergence during the development of goal models for groups of stakeholders and circumstances, has an obvious usefulness in facilitating the requirements analysis process in one-of software development projects. Indeed, in realistic software development projects, there is rarely a unique set of features of the system-to-be that is capable of fulfilling the business objectives of the stakeholders. Instead, a number of alternative solutions are usually present for further debating amongst analysts and stakeholders as to which one is more suitable for their business and for what reason. In the meeting scheduling example, the appearance of alternative ways for fulfilling the goal Invite Participants to a Meeting, is a natural result of the process of understanding how the goal can potentially be fulfilled and may originate from several sources such as the stakeholders’ experience from studying similar problems or from different points of view of different stakeholders. Good decision making in the requirements analysis process requires that all these options are known and evaluated before the preferred one is chosen. The choice depends again on understanding the specific circumstances surrounding the goal Invite Participants to a Meeting in the particular problem instance, including the high-level preferences of the particular user or organization. This way, both the decision process is systematized and rationalized, and the decision result comes with an explicit justification through the representation of how it relates to key decision criteria, which are also represented through goals.
1.3.4 Goal Variability and Software Customization

While exploration of alternative designs is an integral part of the requirements analysis for one-of software development, the same need for alternatives identification, justification and comparison exists when building generic software that is capable of addressing this wide range of alternatives by being appropriately customized. In this case goal variability also plays a significant role in understanding what are the alternative customizations that need to be supported and which of these are suitable for which situation.

The process of understanding what alternative customizations can be supported by a software system has been extensively studied in the area of software product-lines. In product lines, the main concern is the construction of generic software assets that can be reused as many times as possible to solve similar, but not necessarily identical, problems. The generic software assets can be extended or configured in a way that the resultant software system fits well to the individual characteristics of each of the problems. A process called domain analysis is performed in order to identify what features of the family of products are expected to be common in every instance of the product line, and what features are expected to vary by, for example, being optional or alternative and mutually exclusive with other features.

Recall, however, that requirements exist before and beyond software features. As the features themselves are a result of understanding the requirements that need to be met, so does feature variability, which must be a result of our understanding of requirements variability. Thus, the ways by which goal attainment can vary, which is a representation of requirements variability, must obviously play a role in understanding what features should be supported and how they should vary. Returning to our Meeting Scheduling example, all ways for inviting participants we mentioned above (phone, e-mail or land mail), should be possible in different instantiations of a software product-line for supporting the scheduling of meetings, through the existence of alternative software features. Therefore, the goal model is used as a tool for identifying these possibilities and justifying them in terms of how they meet the stakeholders’ top level goals.

Furthermore, our characterization of each alternative at the goal level can be used to select which feature fits to which problem instance. We saw, for example, that for inviting participants the phone call is efficient, but the written invitation is more formal. Through this observation, however, we have reduced the feature selection into a selection of which quality between efficiency and formality is more important. This is even more interesting when several other features of the envisioned meeting scheduler influence efficiency or formality. Specification
of the preferences of individual stakeholder instances under given circumstances over these qualities can be used to infer what software features (not just goal alternatives) are most appropriate for them. This way, in an attempt to derive the appropriate product for a given problem instance, a potentially large set of decisions over behavioral details of the system are reduced to a specification of priorities over high-level qualities.

In conclusion, analysis of alternatives at the goal level provides justification of candidate or alternative features of the system-to-be and allows high-level selection of those that best fit the characteristics of the problem at hand. In this thesis, we present systematic ways for identifying goal alternatives, specifying criteria for choosing alternatives that best fit to a particular situation, and associating them with software configurations. In the next section, we will look at the general ideas as well as the specific contributions of this thesis in more detail.

1.4 A Framework for Acquiring and Reasoning about Goal Alternatives

We showed above that goal models provide a useful tool for identifying alternatives and for selecting the most desired ones to match the characteristics of individual problems. So far, however, the goal-oriented requirements engineering literature has not provided systematic ways for doing so. On one hand, there has been limited study on how the process of identifying more alternatives when developing goal models can be facilitated towards both understanding the nature of variability that is introduced and allowing the discovery of more goal alternatives that would otherwise remain hidden in the domain. On the other hand, while the role of high-level qualities in selecting alternatives at the lower level has been quantified, an expressive language for allowing stakeholders to precisely describe their priorities over such high-level qualities is yet to be introduced. Finally, while there is a claim that goal alternatives imply alternative designs or configurations of generic designs, an approach to interpret the former into the latter is still missing.

In this thesis, we present a systematic framework for acquiring and reasoning about variability in goal models as well as for associating goal variability with the configurability aspect of existing software systems. The framework consists of a method for guiding the goal decomposition process in order to allow easier and more complete identification of goal alternatives, as well as a formal language that allows expression of preferences over desired degrees of satisfaction of high-level qualities of desired solutions. In addition, we show how these tools can
potentially be useful for customizing software systems, by introducing a method for connecting
goal alternatives with alternative configurations of existing software systems. In this Section
we look at each of these contributions in detail.

1.4.1 Acquiring Alternatives

The first concern in developing generic high-variability goal models is the origin of this vari-
ability, that is how analysts come up with alternative means-ends relationships for goals. In
our study, the notion of the means-ends relationship itself is questioned. We find that there is
a more elaborate way to characterize decompositions of goals into alternative subgoals, that is
OR-decompositions, in which means-ends is only one of the possible categories. For instance
the OR-decomposition of the goal Invite Participants to a Meeting into the
goal Have Invitations Sent by Secretary and the goal Have Invitations
Sent by Meeting Initiator, can be seen as an “agentive” decomposition rather than
a means-ends one, in that it emerges by considering different agents for achieving the goal with
their activities. Thus, we collect such types of motivations for introducing OR-decompositions,
which we call variability concerns, and use them as drivers for introducing variability in the
goal models. In addition, we provide a way for constructing lists of such variability concerns
that are specific and tailored to particular domains, through examination of corpora of texts
describing those domains.

This part of the framework offers a way to think about goal variability, that enables system-
atic identification of alternatives and arguably allows the discovery of alternatives that would
otherwise remain hidden. In addition, we offer an initial exploration as to how the circum-
stances surrounding goal fulfillment that are purely non-intentional, such as properties of the
environment or user skills, can be modeled in a way that shows how they influence the selection
of alternatives.

1.4.2 Selecting Alternatives through Preference Specification

The preference specification part of our framework shows how one can reason about the vari-
ability that is identified using the above methods. The purpose of such a reasoning exercise
is to select alternatives given specific criteria that correspond to the problem instance at hand.
These criteria are specified in terms of stakeholder preferences over their quality goals as well
as properties of the environment that initially hold true. Preferences, in particular, are built
using formulae that express desired degrees of satisfaction or denial of quality goals and/or
behavioral characteristics of the preferred alternatives. Linear temporal logic (LTL) is used to express such desires. Then these desires are ranked in a way that reflects stakeholders’ understanding of their relative preference. A tool based on a preference-oriented planning system searches through the space of alternatives and returns the ones that best match the given preference specification.

This formal language and the corresponding reasoning tool aims at filling the gap between the high-level desires and priorities of individual stakeholders and low level details of what constitutes a solution that is most suitable for them. This way, stakeholders are not forced to interpret the former into the latter, a task which they may not have the expertise to perform. In addition, even if stakeholders know how to translate their preferences into solutions, given the potentially vast size of the space of alternatives, selecting the one that really suits them can be particularly burdensome without automation and supporting tools.

1.4.3 From Goal Alternatives to Software Configurations

The last part of the thesis constitutes an exploration of the relationship between the goal alternatives and software variation points, aiming at understanding the potential of applying our approach for configuring customizable systems. We introduce a method for inferring goal models from configuration screens of existing systems, for the purpose of using the inferred models to perform the configuration tasks. We use an e-mail client in our exploratory case study. We find that a significant part of the configuration options could be mapped to user goals, implying that goal models, together with the preference specification component we introduce, can potentially be used as configuration tools.

Our motivation for exploring the relationship between goal variability and software variability is based on the observation that if we were able to map the former into the latter, software configuration would be a matter of high-level preference specification. Despite the strength of the assumption that current configuration screens of desktop applications constitute results of a sensible requirements analysis process, our exploration shows that this connection between goals and configuration options is still possible to a significant degree. This result paves the way for more research both on building tools for requirements-driven leverage of existing configuration interfaces and, perhaps more fundamentally, on developing techniques for requirements-driven design of the configurability aspect of software systems, and, of course, the respective binding interfaces.
1.4.4 Structure of Thesis

The thesis is structured as follows. In Chapter 2 we have a close look at the literature within the scope of our effort; the chapters that follow repeat, summarize and discuss parts of this chapter in order to better show how they connect to the introduced research. Chapter 3 provides a more detailed account of goal modeling, and presents the core elements of the goal modeling language we are going to use throughout the thesis. Each of the subsequent chapters assumes its own subset, extension or adaptation of the core language presented in Chapter 3. Chapters 4, 5 and 6 provide the pieces of our contributions as respectively overviewed in Sections 1.4.1, 1.4.2 and 1.4.3 above. Finally, Section 7 provides elements of research work that we believe would be useful for consolidating and extending the work presented in this thesis.
Chapter 2

Requirements Variability: State of Research and Practice

2.1 The Literature Landscape

Variability has long been a core research subject in the product-lines community. In product-lines the motivation behind investigating variability is to come up with its dual, that is commonality, whose identification, in turn, paves the way for the definition of reusable core assets. The process for identifying commonality and variability is part of a larger set of activities for understanding and analyzing sets of systems that share a similar purpose. This process is called domain analysis ([90, 97]), where the term “domain” implies a set of such systems.

Due to the economical potential of software product-lines, variability from that point of view has been studied from even before the ’90s. For researchers in Requirements Engineering, however, variability had rarely been a primary subject of study, until recently. Instead, in requirements engineering research variability appears as a natural, unavoidable part of the problem that needs to be tackled. For example, in scenario-driven RE, inevitable variations on scenarios forced researchers to introduce the appropriate techniques to model and manage them. Analyzing, for example, how requirements variants are common or different, where they come from, whether the set of variants is complete subject to some criterion, or how variants should be selected has not been part of the problem. When variability is a primary issue in Requirements Engineering, the discussion is almost exclusively on modeling, and, as we will see, the approach taken is largely solution oriented, that is centered on software features.

Meanwhile, a community of Artificial Intelligence researchers has been studying ways by which the configuration of products can be facilitated through the application of knowledge
representation and reasoning techniques. Products here are not necessarily software products and how the configuration representations are acquired, or what the value of such representations as a communication tools are, does not fall within the scope of the investigation. However, this line of work offers invaluable insights as to how variability binding can be supported by automated reasoning techniques.

In this chapter, we will take a thorough look at the bibliography that studies software variability, focusing on the above areas. There are three aspects of variability that we will focus on: variability acquisition (most often referred to as variability identification), variability modeling and variability binding. These aspects correspond to the three main contributions of this thesis: a method for acquiring requirements variability, a goal-based language for modeling it, and a method for selecting requirements’ variants of interest, potentially for configuring software.

For our discussion on variability acquisition, we consider work done in the software product line and software reuse community (2.2.1) as well as work done in the requirements engineering community (2.2.2) which includes both efforts that clearly and explicitly have requirements variability as their topic, and efforts that only implicitly or even unconsciously deal with the issue. Then, in Section 2.3, we take a closer look on how variability is accommodated in requirements models and documents. A large body of literature deals with variability representation in use cases and other object oriented diagrams (2.3.1) whereas feature models (2.3.2) constitute a popular way for representing admissible combinations of features. Variability in other requirements artifacts such as requirement specification documents or behavioral models (2.3.3) as well as meta-models for organizing variation points (2.3.5) is also discussed. In Section 2.4, we look at work on variability binding. Our focus inevitably falls in the configuration community, which offers a set of interesting approaches for handling large and complex design spaces. We also look at work on product derivation in product-lines and scenario management in requirements engineering.

### 2.2 Origins of Variability

The variability of a system can be understood in terms of a set of alternative configurations that the system can acquire. The question posed in this section is what leads to the definition of this set. Why is a variation point or a binding thereof included or excluded from the set? What is the variability acquisition process that provides a systematic way to answer the previous question?

In product-lines the approach that is followed is domain analysis, a systematic process for understanding commonality and variability among existing systems of a domain. This
contrasts to requirements-based approaches, which focus exclusively on the variability that exists in the requirements specification as it emerges during the elicitation process, without necessarily looking at, for example, existing assets or expert opinion on similar systems. In both cases, however, the discussion revolves around the system-to-be and its features.

### 2.2.1 Traditional Reuse-centered Domain Analysis

There are several approaches to domain analysis [71, 72, 46, 1, 114, 90, 109, 11, 125, 97]. Most of them are considered to be part of a general *domain engineering* framework for developing product-lines. Among other things, most domain analysis methods imply a framework for acquiring commonality among the products that participate in the domain, and, at the same time, a way to define what participates in the domain and why.

In Feature Oriented Domain Analysis (FODA - [71]) as well as its successor Feature-Oriented Reuse Method (FORM - [72]) it is explicit that the “domain” is the application domain, that is a “set of current and future applications”. Context analysis, that is the definition of the scope is based on (1) the degree of commonality (i.e. reuse potential) found, (2) the availability of information and expertise, (3) the expected usages of the domain products and (4) the project resources and constraints. Thus, if the set of products does not turn out to have an adequate degree of commonality, the set is redefined; similar is the fate of the domain, if the set contains products that the particular team cannot develop. Information for defining the scope and determining commonality and variability is found in textbooks and standards, by studying existing applications or by asking domain experts.

FODA has been a very popular baseline for work on product family variability analysis, in that it introduces the notion of feature modeling, that is the process of generating feature-based variability models. Thus, in [29], Czarnecki and Eisenecker provide a comprehensive analysis of how features and models of them can be identified. According to the authors, all stakeholders, domain experts, existing systems or OO models (pre-existing or being constructed) can serve the identification of features. In the proposed domain engineering method called DEMRAL, the generation of feature models is influenced by the modular, aspectual and subjective decomposition that is taking place at the same time.

The result of feature modeling has been found to be useful in other product-line frameworks as well, though without necessarily playing the most important role. Thus, in FeatuRSEB ([52]), use-case analysis may be the basic (model-driven) way for discovering variability, but feature models also express “how ‘good’ systems should be built in the future” ([52]), as they
reflect the experience from past projects.

In Domain Analysis and Reuse Environment (DARE - [46]), domain variability information comes mainly from domain experts and examination of existing systems. The same is true for FAST ([125]). In [124], where commonality analysis in FAST is described, the main source of variability information comes from structured meetings with domain experts.

In [10], where Bassett introduces the frame technology for software reuse, an approach to domain analysis based on the definition of archetypes and deltas is introduced. An archetype "exemplifies a category" whereas a delta "gives rise to other category members" ([10]). A frame is a formal description of the archetype and (possibly) its deltas. Although this representational framework must have implications on the way variability is identified, a method for defining the appropriate archetypes and deltas is not given by Bessett. As we will see, Jarzabek et al. ([67]) are greatly influenced by Bessett's archetype-based approach to variability.

PuLSE follows an even more organization/asset-centered approach ([11]). The authors believe that their view of a "domain" as a strictly organization-centered notion, is easier to scope. Existing, future and potential future applications are "mapped out" on a table where the rows are distinct characteristics and the columns are the applications; the content of the table shows which characteristics belong to which applications, whether the competitors have these characteristics, as well as benefit and cost values. The table is used for decision making in terms of which products are the worthiest for development. Thus both scope and variability are defined with respect to existing assets and their potential extensions.

Finally, in Organization Domain Modeling (ODM - [109]) the distinction between variability in "problem space" versus variability in "solution space" is not found to be important. Thus, methods for capturing and modeling domain requirements may or may not be useful but not as a source from which the design is generated. There is a formal approach for scoping the domain, by considering candidate systems and characterizing their potential membership to the domain (exemplars, counter-exemplars, borderline-exemplars). In place of requirements, a process called "contextualization" of domain data refers to understanding and documenting the organizational environment which, according to ODM, constitutes a domain in practice.

To sum up, in all the above efforts the focus of the domain analysis is a set of similar systems and not necessarily similar problems. This is demonstrated by the primary role that is given to the examination of existing systems and artifacts that describe them. As it will become apparent later, even the characteristics of the models that are used to represent the result of this process disclose the strong solution-oriented character of domain analysis.
2.2.2 Requirements-Driven Acquisition of Variability

Traditional domain analysis methods provide a “holistic” approach to variability acquisition in a sense that variability originates from all sorts of sources certainly including existing systems. However, there is a) work that follows the same holistic approach but puts unusual weight on requirements analysis as well as b) “pure” requirements engineering research work that is not consciously targeting families of systems but, nevertheless, deals with the problem of variability acquisition.

Thus, as in reuse-centered approaches, the Domain Specific Software Architecture (DSSA) framework [114], bases the identification of variability and commonality on domain experts, although it is mentioned that customer input may play a significant role, too. A set of scenarios are identified and result in a reference requirements specification document that contains variation points from where the concrete requirements can emerge (more details on the representation are in Section 2.3). The notion of “requirements” here, reflects the traditional idea of requirements being software features. This is also the case in [56], where a feature model is constructed as a result of a requirements analysis process. In this chapter, we generally use the term requirements specification or, simply, specification, to refer to the traditional understanding of requirements as being features of the system-to-be.

In [38] a requirements specification-centered approach is also considered. The author proposes that commonality should be expressed in a Product-Line Requirements Specification document. Such document contains specification that are common to all members, as well as statements that show how specifications may vary. The author does not offer a particular process for deriving such a document, but supports its usefulness in terms of understanding and communicating the family-wide characteristics and decisions, as well as producing the individual member specifications for validation and verification.

In [76], although the scope of the product family is defined also by investigating existing solutions and by asking domain and market experts, after the domain has been defined requirements play a primary role. High-level requirements (from where variability may emerge) are classified subject to the aspect of the family they refer to. The identified aspects are then turned into variation points that introduce alternative ways by which the aspects can be dealt. Instead of aspects, [112] considers dimensions over which variability can be defined in an hierarchical way (regarding variability dependencies) - the authors imply that such dimensions can be orthogonal to a sufficient degree.

The aforementioned methods are strongly solution-oriented as most the times they are mo-
tivated by the need to identify commonality and variability specifically for product-lines. For example, in [38], the example domain of application is a flight control system where variability comes from different aircrafts in which it can be installed. Similar observations can be made about the other methods we presented above. In requirements engineering research, variability has also been (implicitly or explicitly) studied in the contexts of goal oriented and scenario oriented requirements analysis, without explicit reference to product lines.

The goal oriented school proposes the notion of recursive decomposition of problems (high-level goals) into subproblems, until a solution is reached. Both the formal composite system centered flavor of goal analysis, KAOS ([31]), and the quality/social systems centered one, i*-based Tropos ([126, 22]), use the notion of OR-decomposition of goals. In [87] the particular property of goal models is discussed: each subgoal of an OR-decomposed goal is simply an alternative way to fulfill the parent goal; the difference is that each subgoal implies different degrees of fulfillment of high-level system-wide qualities. This property has been exploited in [62]. However, neither KAOS nor Tropos provide a way for systematically discovering the variability that appears in the generated models.

The scenario-oriented school inevitably introduces variability in an attempt to provide classes of similar scenarios. In [110], a use-case is a collection of scenarios where several events (i.e. start or end points of actions) may occur in a variety of orders (alternative events are also introduced). Constraints are then employed to restrict the space of alternative scenarios to a subset of interest. In [2], families of scenarios contain variation points that correspond to decisions that change the way a scenario progresses.

The two approaches, scenarios and goals, are bridged by Rolland et al. who introduce a formal approach for pairing scenarios with goals ([103, 102]): a pair of a goal and a scenario yields the requirements chunk. The requirements chunk is refined following several rules; an important assumption is that the scenario-part of the chunk consists of goals, and that the goal in the goal-part is achieved by the scenario in the scenario-part. Alternative refinements can be introduced by considering alternatives for either the goal-part or the scenario-part of the chunk. Thus for Rolland et al. variability is discovered: a) through changing the parameters of a goal, b) through keeping the parameters of the goal fixed and changing the scenario (“workflow”) by which the goal is achieved.

This work by Rolland et al. serves as a significant inspiration to our work. In particular we elaborate on the use of goal parameters for decomposing goal, propose a way to discover such parameters, and introduce a different decomposition method that is geared towards developing high-variability goal models, rather than identifying scenarios.
2.3 Variability Modeling

Having seen how requirements variability is acquired either in the context of domain analysis for software product lines and in the context of requirements engineering, we discuss next how requirements variability can be represented. First, we discuss efforts on incorporating variability in use-cases and other diagrams that are used during object oriented analysis. Then we describe the notion of features and feature models. We then look at other types of models and representations that can be produced through the RE process; some of them directly relate to the methods we described earlier. Finally, we discuss efforts on defining variation point meta-models.

2.3.1 Variability Modeling and Object Oriented Analysis

The Reuse-driven Software Engineering Business (RSEB) framework [66] is one of the base-lines on the object oriented approach to variability representation. In the RSEB, a construct called variation point is introduced in order to signify that a particular element can be reused in several alternative ways. These ways are enumerated and documented appropriately; the variation point appears as a dot in the appropriate element and is considered to be a special type of annotation.

Perhaps most relevant to our discussion here is the use of variation points in use-cases, as these are the elements that are considered as requirements models in the OO culture. The variation point that appears in a use case is materialized in several ways: alternative \texttt{<<extensions>> (subject to varying users, types referred, functionality etc)}, parameters placed at the textual description of the use cases (and can even be invocations of scripts instead of simple variables) or alternative specializations through \texttt{<<uses>> relationships}.

Variability in use-cases has also been a concern in [68, 28, 15, 50, 16] and [53]. Thus, in [68], the generic use-case diagram may contain use-cases and actors that are stereotyped as variants as well as (sub-)sets of use cases that are stereotyped as optional. The textual description of the use case is adapted accordingly: particular steps of the success scenario contain questions, alternative answers to which correspond to alternative variability bindings. Below the question, the options are separated using XML-like tags \texttt{<variant-ALTn: [answer] > [Use case content]}</variant> or simply \texttt{<variant-OPT> [Use case content] </variant>}. A very similar approach is followed in [122], whereas in FODAcom ([119]) parameterization and extension points are used. The former are very similar to RSEBs variation points,
whereas the latter can be used to represent variability in, for example, actors that are involved in a particular use case. In [15], tags are introduced in the use case description that act as placeholders. The possible values of each placeholder are given in a separate section below the success scenario. A very similar technique is proposed in [28] where the term “Variations” is used to describe that special section. In [50], \texttt{<<kernel>>} and \texttt{<<optional>>} use cases are distinguished, whereas in [16] “essential” use cases (essential in a sense that they tend to be technology-free, idealized and abstract) are again reused through \texttt{<<extends>>} and \texttt{<<uses>>} relationships, as well as through parameters.

In this line of work, Halmans and Pohl provide a notably elaborate view of the problem of modeling variability in use-cases ([53]). The authors distinguish between technical and essential variability. The latter is the customer’s viewpoint on variability and contains a number of subcategories such as: functionality (function and behavior), system environment, business process integration, quality and information/data. They investigate whether and how each of these types of variability can be accommodated in use-cases. The result is that variability in quality, business process integration and information variability cannot be represented using use-cases. For the remaining two types of variability, though, the authors propose the appropriate extensions. Textually, variability in use case preconditions, and provision of alternative success scenarios can be used to represent functional variability. Diagrammatically, a special element called variation point is used. This variation point is different from RSEB’s one, in that it is an element and not an annotation. It introduces a branching factor to \texttt{<<includes>>} relationships. The “output” of a variation point can be mandatory, optional or alternative use cases or actors. The “input” is the origin of the \texttt{<<includes>>} relationships, i.e. another use case, or undefined if the “output” is actors.

An alternative approach to variability in use cases is given by Jarzabek et al. in [67] where the notion of flexible use cases is introduced. These are “archetypical” ([10]) use cases (“default use cases”), which are written to a frame format (yielding to f-default use cases) that can further be customized through scripts based on Bassett’s frame technology.

Apart from use cases, other types of OO diagrams, such as static structures, need to embody variability. The literature (e.g. [71, 66, 3, 50]) recognizes three basic ways to do so: specialization (is-a based variability), aggregation (part-of based variability) and parameterization.

Gamma’s pattern system ([37]) has been found helpful in terms of determining how inheritance, aggregation etc. can be used to represent certain types of variability ([66, 74, 113] - [113] discusses variability in web services). In [74] the single and “multiple Adapter” patterns are used to represent OR-ed and XOR-ed alternatives whereas the “Option” pattern is used...
for optional alternatives. In [66] the “Mediator” pattern is used for hiding variability in communication among alternative classes, the “Decorator” and “Composite” patterns are used to implement (OR-ed) use-case extensions and the “Observer” in cases where e.g. subscription to events from alternative classes is needed. In [113] the “Strategy” pattern is also considered for expressing functional variability. In addition, in [29] there is an extensive discussion on how feature models (see Section 2.3.2) can control variability in OO diagrams. For example, single inheritance can roughly be mapped to a feature with alternative subfeatures, while Multiple and (preferably) parameterized inheritance can be mapped to OR-ed subfeatures. These efforts, however, are variability implementation techniques for object-oriented designs rather than techniques for modeling variability at the problem level.

2.3.2 Feature Modeling

The result of commonality and variability analysis in traditional domain analysis and product-line engineering frameworks, particularly those influenced by FODA, is a model that shows how the distinct features of the domain can be combined to give a domain member.

For FODA ([71]) a feature is “a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems” whereas for ODM ([109]) it is a “significant differentiating capability across domain systems”. These imply that features are meaningful when talking about characteristics of software systems. However, modern approaches to feature modeling try to lift such ontological restrictions. Thus, for Czarnecki and Eisenecker ([29]), a feature is “an important property of a concept instance [which] allow[s] us to express the commonalities and differences between concept instances”. This is not far from Bassett’s archetypes and deltas ([10]).

The feature model shows how features can be combined together to provide a concrete product. The approach to feature modeling that is given in [29] appears to enjoy wide acceptance in the literature, so we use that as a baseline for our presentation here. Thus, according to [29], a feature model is a tree. Each node represents a feature. The root node represents the concept. If a feature has children these are subfeatures of that feature. The subfeatures can be mandatory, optional, alternative or OR-features, with respect to the parent feature or their sibling subfeatures.

A (valid) featureset is a set of features that satisfies the feature model. If a feature that has subfeatures is considered for the construction of a featureset, the characterizations of the subfeatures as mandatory, optional etc., imply particular rules on which of the subfeatures must
also be considered for the same featureset. Thus, mandatory subfeatures must be included wherever the parent feature is included, whereas from a set of alternative subfeatures exactly one must be included.

Czarnecki and Eisenecker claim that the semantics of the feature/subfeatures relationship are and must remain undefined so that the configurability aspect is clearly separated from other aspects. Thus, for example feature models cannot have structural semantics. A feature model should be seen only as a visual way to represent a set of constraints that need to be satisfied when creating subsets of features.

Feature models play a dominant role in the product family literature, where several issues are raised regarding their use. For example, in [56, 101] the problem of representing feature models in UML is investigated. In [21] annotating low level features with telecommunication QoS attributes is discussed. Integration of Feature Modeling with OO analysis has also been a popular investigation theme. FeatuRSEB, [52] and FODAcom [119] set the baselines for such a process whereas [50, 120] propose methods for integrating feature models with UML diagrams. We will not get into details on the modeling technicalities that such integration implies.

Apart from the hierarchical feature model, another approach for mapping the product family is given in [112]. An n-dimensional system is constructed, where each dimension represents a variation dimension (e.g. behavior, platform etc. - notions similar to aspects of high level discussed in [76]). The axis’ represent alternative variability bindings of the implied variation points, in a way that each member of the product family occupies a point in the constructed n-dimensional space.

### 2.3.3 Variability and other Requirements Modeling Approaches

We now turn our focus to work on representing variability in other requirements views such as functional and behavioral. In FODA [71], Kang et al. claim that there are three ways to represent variability in a functional/behavioral model: 1) break the model into multiple ones, 2) create one parameterized model that incorporates variability and 3) create a generic model and instantiate it accordingly. (1) appears to be the choice if the other two don’t work and (3) seems to introduce a scalability issue.

Option (2) is illustrated in [71] as well as in [50] through statecharts. Indeed, one can have OR-decomposition of states into sub-states and transitions to each of these substates guarded by special “variability” clauses. Other efforts for modeling behavioral variability that have
been introduced, employ the use of variable label transition systems ([86]), modal transition systems ([44]) or message sequence charts with variation points ([111]). In [2], the “application variation model” is a mixture of a feature model with a UML activity diagram: each feature is connected with an activity, and a special graph that consists of activities but has the semantics of a feature model is introduced. [101], instead, populates activity diagrams with additional decision elements and, similarly, in PuLSE ([11]), generic storyboard models (i.e. workflow models) contain decision nodes which are solved depending on the domain member under consideration. Jarzabek et al. ([67]) customize such workflows by exporting them in textual format and automatically editing them through customization scripts.

In [38] a semi-formal hierarchical representation of variability decisions in the product-family specification document is annotated appropriately so that customizable fragments of an SCR ([57]) specification (called CoRE classes) can be produced. To express variability, the tabular representation may: a) include tags/placeholders (e.g. to direct the effect of an event to a certain class), contain variation variables (mixed with the actual specification ones) or have conditions on whether particular portions of the specification can be included or not. To ease that transition from the annotated document to the customizable tabular representation a set of heuristics is proposed.

Variability can also be expressed in the traditional requirements specification documents. In DSSA ([114]), each requirement has a title which is suffixed appropriately: \(-\text{OPT}\) if is is optional, \(-\text{ALT}n\) if it is alternative, where \(n\) is the nth alternative. Non-functional, UI, design and implementation requirements may include variability suffixes as well. In FODAcom ([119]), the specifications that are derived from the use cases, are templates that may contain extension and parameter tags. Extension tags are accompanied by additional alternative requirements specifications (written in informal text) that extend the behavior of those of the main specifications that are appropriately tagged, while parameter tags simply come with a set of alternative values.

Finally, as we saw in earlier sections, variability in goal graphs is modeled through OR-decompositions of goals ([87]). However, following the \(i^*\) ontology ([126]), variability can also be established through contribution links, though implicitly. Thus, when a soft-goal is a target of more than one “makes” contribution links, this may imply that there is a choice among the goals that are origins of these links. The same decomposition tree approach, accompanied by the appropriate documentation, is followed in the scenario-goal coupling work in [103].
2.3.4 Variability and Problem Frames

The relationship between requirements variability and Problem Frames ([65]) has recently been the subject of investigation in the research community. Problem Frames is a conceptual framework for requirements analysis aimed at clearly distinguishing a software problem from its context. Thus, Problem Frames are directly based on Jackson’s proposal of viewing software ("machine phenomena") as one of the two overlapping domains, the other being the context. Problem Frames employ context diagrams to represent the relationship between the machine and its context and problem diagrams to analyze the relationship between indicative domain properties (i.e. what is there) and optative requirements (i.e. what we wish to be there).

Classsen et al. ([26]) attempts an exploration of the use of these diagrams in the presence of optional requirements. While this work shows how plain feature models don’t necessarily disclose several consequences that variability has in the whole requirements specification, it does not offer a concrete approach for modeling and reasoning about large sets of problem variants. The same comment applies to work by Salifu et al. ([106]), on the use of Problem Frames for analyzing requirements for monitoring and switching problems. Again, Problem Frames are found to be useful for exposing, modeling and understanding the complexity of the problem, but a systematic approach to exploring and dealing with alternatives is not given.

2.3.5 The Variation Point model

Research work has been conducted on the definition of meta-models of variability. This work is motivated by the need to document all variation points that exist in assets in a uniform way, possibly in the form of a separate model, the variation model. A feature model whose features have been bridged with the variation points in the OO diagrams of the assets (i.e. the FeatuRSEB approach) is an example of a variation model.

However researchers have tried to envision more “informative” variation models than bare feature models. Thus, in [7], a variation point contains many alternative variants each of which refers to one or more assets; in addition, the variation point itself has a rationale. In [105] each component element contains variability which is described through: an element in one of the system models (describing an asset), a set of variants, a resolution (binding) mechanism, as well as a decision rule. In [13], Becker presents an even more elaborate meta-model, which, among other things, captures variant dependencies, distinguishes between static and dynamic variation points (depending on when they are resolved) as well as between static and generic assets (the latter need to be instantiated and may contain static variation points). Finally, in [122],
without really introducing a meta-model, the authors show how the same variation point can be found in a requirements description as well as in OO diagrams controlling object inheritance, parameterization and adaptation of callback mechanisms.

2.4 Variability Binding and Product Configuration

In the previous sections we looked at several proposals for acquisition and modeling of requirements variability. We will now look at research work on variability binding, a term that refers to the process of selecting variants of interest out of the set of variants that is implied by the a variability-enabled requirements model. As we will see, most of the variability modeling approaches we discussed above offers limited support for selecting variants of interest. Most common is manual binding of variation points, potentially through reference to a guiding structure. In this section, we will prefer to focus on work that offers more formal, automated, or otherwise elaborate ways for deciding how to bind variation points. This includes DSLs, product configuration technologies as well as simpler yet formal decision support systems that can potentially facilitate the variant selection process.

2.4.1 Feature-based product derivation

Feature-oriented approaches to variability modeling, (as well as, in fact, most of the work presented in the previous sections) assumes that binding is a process of examining each variation point and assigning its binding value either without any support, or through the use of, for example, an assistive model or guide. Such an example is FeaturSEB ([52]) where selection of alternatives in the feature model can me made by for example consulting the use cases. Another example is [30] where a (manual) technique for gradual resolution of variation points in feature models is proposed.

One can argue that the very nature of feature models is such, that reasoning about their variability for the purpose of selecting featuresets that meet certain criteria is rather difficult, at least without adding extra elements to the basic notation. As we saw, feature models are understood as having the exclusive role of representing the configurability aspect of high-variability systems ([29]). Thus, in order to be capable of representing many and diverse variability aspects in a single view, feature models deliberately say little about the semantics of variability. For example, in the same feature model, behavioral, structural or quality features may co-exist. This, however, comes at the cost of limiting the reasoning and analysis opportunities over those
models, which would support the process of selecting alternatives.

Efforts to overcome this difficulty are based on attaching selection criteria to features. In [128], features of the feature models become nodes of a Bayesian network whose nodes represent qualities of the system-to-be. This way different featuresets imply different probabilities as to whether certain qualities will or will not be achieved. Well known algorithms for reasoning about Bayesian networks can apparently be applied to perform the reverse process: given a desired probability threshold for a certain quality, one can calculate featuresets that achieve it. This idea of attaching such decision models in variation points is also used in other non feature-based techniques. Popular seem to be several types of decision matrices that, roughly, construct a set of selection criteria and then assign a score as to how each of features meets each of the criteria (e.g. in ([11, 2, 21, 9] and to some extend [107] where product maps are used). In a totally different context, but using a very similar approach, the use of multi-attribute decision theoretic approaches has also been explored ([75]).

2.4.2 DSLs

Domain Specific Languages (DSLs) constitute an interesting approach to variability binding, and are particularly popular in the software product-lines community. In DSLs ([27, 14]), binding of variability is performed by constructing scripts in a language whose terms are specific to a particular domain. DSLs are designed in accordance with a domain analysis process to allow for high level descriptions of solutions for particular problem instances. Yet, there may not be an explicit variability model that a DSL is binding. Due to the fact that they are tailored to a specific domain, DSLs are supposedly easy to use by experts of the domain, and not necessarily computer or software experts. Typically, however, DSLs are solution-oriented and are rarely related to knowledge found in the problem domain. Even when DSLs employ problem-specific concepts, these are restricted to structured and formal processes or functions found in the problem domain, e.g. financial operations in ([20]).

2.4.3 Alternatives Analysis in Requirements

In requirements engineering there has been limited work on supporting the selection of requirements alternatives identified during the corresponding analysis. In the context of a scenario generation approach from generic use-cases proposed in [110], a constraint language is considered for filtering out undesired scenarios. The language is based on a logic called ALBERT-CORE which the foundation of the ALBERT II specification language [18]. In ad-
dition, design rationale diagrams are used to assess alternative designs to satisfy the generic use cases, subject to certain quality criteria. While the proposed language offers a significant degree of expressiveness (though does not seem as expressive as LTL, which we will use), the overall comment on this work is that it focuses on the operational details of the solution leaving their origin and justification to stakeholder goals out of the scope.

In [77], instead, system goals are the primary concern. The paper introduces a way to assess partial satisfaction of goals through the use of objective functions on measurements of physical properties of the domain. This way, that alternative designs imply different values for various objective functions, means that preferences over the relative importance of these functions can lead to the selection of the corresponding design. However no specific language is given for expressing such priorities. The same idea of evaluating alternatives, though more informally, is discussed in [87] and, later, in [62]. Both [87] and [62] are direct predecessors of our work and we will discuss them in detail in the next section.

### 2.4.4 Product Configuration Technologies

Work on product configuration (see [104] for a survey), which is a community within AI, explores ways to search through large and complex configuration spaces in order to identify variants that meet certain constraints. Most product configuration technologies appear to follow a similar approach: a (generic) product model with a great number of degrees of freedom ([100]) is accompanied by an infrastructure for describing individual requirements and constraints, while an inference engine searches for configurations that satisfy both the generic and individual model (e.g. [40, 54]). For example, preferences over predefined low-level decision points ([70]) and evaluation based on impact of decisions to high level qualities of the result ([5, 100]) have been proposed. In [5] measuring impact to high-level qualities, serves the purpose of abstracting the configuration details from the user. These efforts, however, rarely refer to software, let alone stakeholder requirements. Also none of these proposals supports the definition of constraints over temporal characteristics of admissible configurations; something we introduce in our preference specification language.

### 2.5 Conclusion: Open Research Questions

We discussed a number of research efforts on acquiring and modeling variability as well as for supporting the selection of alternatives of interest in requirements engineering. Our general
conclusion is that the current state of knowledge is missing a) an approach to variability ac-
quision that is both systematic and focusing not on the solution but on the problem and b) an
expressive language for describing high-level criteria for automatically selecting alternatives of
interest, from a potentially large set of such. Such criteria should reflect the high-level quality
and behavioral desires of stakeholders, without reference to system details.
Variability acquisition has been approached almost exclusively in the area of Domain Anal-
ysis for product lines. As we saw in Section 2.2 the current proposals stay at the level of just
pointing to the sources of variability information (e.g. existing systems, domain experts, sev-
eral system decompositions) or at best, providing advice for structuring the elicitation process
(e.g. brainstorming meetings). No existing variability identification approach combines:

1. a focus on problems rather than solutions, which we argued why it is essential earlier,
2. a concrete procedure for systematic identification of problem variants, making thereby
   (problem) variability discovery less of an art and more of an engineering effort,
3. reliance on well-studied modeling techniques for requirements analysis that are both
   scalable and amenable to reasoning.

In terms of alternatives selection, there is even less progress in either Product Lines or Re-
quirements Engineering as we saw in Section 2.4. However, the emergence of technologies
such as DSLs and product configuration techniques alone is indicative of the necessity of rais-
ing the level of abstraction when leveraging domains with high degrees of variability. While
software and other tangible products have been the subject of such research, the intangible
“logical” products of requirements analysis (goals in our case) have not, although they exhibit
the same variability and complexity.
In this thesis, we attempt to fill this gap by introducing a framework for identifying and
reasoning about requirements variability and, in addition, we show how this framework can
potentially be useful for configuring elements of the software system. In the next chapter, we
will start presenting this framework in detail by discussing its modeling foundation.
Chapter 3

Goal Models and Goal Variability

3.1 Overview

In this chapter we present the basics of the goal modeling language we are going to use throughout the thesis. We describe the visual elements and the ways by which they can be linked to each other to form goal models. We also present the semantics of those links with respect to the satisfaction of goals and provide a sketch of how this is going to be useful for our objective of evaluating alternatives. To fit its particular purposes, each of the subsequent chapters includes a different set of extensions and/or restrictions and a different choice of a formal language for defining the semantics of the language.

The chapter is organized as follows. In Section 3.2 we briefly describe how goal modeling has evolved in Requirements Engineering research and locate the modeling formalisms that are closer to the one we will use in this thesis. In Section 3.3 we present the goal language and in section 3.4 we present the basic satisfaction evaluation algorithm. Finally in Section 3.5 we provide a rough description of how the satisfaction evaluation algorithm is used for assessing the quality of alternatives implied by the goal model.

3.2 Goals in Requirements Engineering

Goal models are visual formalisms that have been found to be useful for identifying requirements of software systems by focusing on understanding the intentions of the involved stakeholders ([88, 31, 4, 127, 126]). A major motivation for using goal models in the context of a requirements engineering process is that they offer a systematic way to derive low-level operational details of the system-to-be from the high-level desires of stakeholders, through a contin-
uous decomposition process. This decomposition process has been greatly inspired by work on Artificial Intelligence, particularly on problem solving using hierarchical decomposition structures ([91]). In this process, the high-level goals, which represent the problem the stakeholders pose, are recursively decomposed into goals of lower level, which gradually change from being parts of the problem into being parts of the solution. Thus, at the bottom level of a complete goal model the activities comprising the behavior of the system-to-be emerge. Moreover, top-down decomposition is complemented by a bottom-up model development process, in which the objective is to explain why certain low-level stakeholder desires and system behaviors are posed/requested, through introducing goals of higher level. This way, complete requirements acquisition can be hypothesized, in that, ideally, all system behaviors that fulfill the top-level goals are assumed to have been discovered by the end of the process.

Numerous research efforts have been dedicated towards constructing languages for expressing goals and their decomposition. In [31], Dardenne et al. set the foundations for a formal framework for modeling and analyzing system goals, known as KAOS. In KAOS, development of goal decomposition structures constitutes the main part of the modeling activity, and is complemented with a variety of associated non-intentional elements, such as agents, events and actions. At the bottom of the decomposition structure, low-level goals are associated with operationalizations, which are sets of actions to be performed for achieving the goals. The semi-formal visual notation can be formalized using Linear Temporal Logic, which allows the application of several reasoning procedures for checking, for instance, the consistency of the resultant model. The basic KAOS framework has been subsequently enriched in several ways, through, for example, the introduction of (AND-)decomposition guiding patterns ([32]) or techniques for conflict and obstacle analysis ([117, 116]).

While KAOS focuses on functional goals, non-functional quality goals were soon found to be important in goal-oriented requirements engineering. Such goals represent vague qualities of the system to be such as Privacy, Security or Usability. The NFR framework, introduced in [88, 24], offers a way to decompose such goals in order to find precise operationalizations by which they can be achieved. However, when compared to their functional counterparts, quality goals imply different modeling challenges, in that they do not allow precise definition of their satisfaction criteria. We are interested in satisficing such goals, not completely satisfying them. Thus, instead of “crisp” decompositions, quality goals require relationship constructs that allow qualitative analysis of stakeholders’ understanding about their satisfaction. These ideas were soon incarnated into $i^*$ ([127, 126]), a popular semi-formal goal modeling language. In $i^*$, decomposition structures of functional goals are based on means-ends relationships, while for
quality goals, *contribution* links between them are used to show how the satisfaction of one influences the satisfaction of the other. Moreover, an elaborate way to support qualitative reasoning about how satisfaction of one goal propagates into the other has been introduced in [49] and [108].

The goal models we use in this thesis are based on a restricted version of i* ([126]) in combination with the modeling and analysis framework introduced by Giorgini et al. ([49]). They consist of a goal decomposition structure loosely coupled with a goal graph for assessing the quality of alternative solutions of the decomposition. In the next sections, we present our goal modeling language and, then, we show how we will use the analysis framework presented in [49] to serve our purposes.

### 3.3 A Basic Goal Modeling Language

#### 3.3.1 Fundamental Concepts

The fundamental elements of goal models are goals. Goals describe abstract states of the world that the stakeholders want to reach. For example, Schedule Meeting, Collect Constraints or Minimize Human Effort are all goals\(^1\). There are two types of goals we will consider, *hard-goals* and *soft-goals*. Their difference is that soft-goals, as opposed to hard-goals, are goals for which there is no clear-cut criterion that can be used for deciding whether they are satisfied or not. For example, the goals Schedule Meeting and Collect Constraints are hard-goals because we can come up with specific criteria to test their satisfaction. Instead, Minimize Human Effort is a soft-goal because we cannot easily agree on such criteria: arguably, what constitutes human effort and when it is minimal is subject to individual opinion and interpretation of the terms. Obviously, hard-goals and soft-goals are suitable for expressing functional and quality requirements respectively.

In addition to hard-goals and soft-goals, *tasks* are also used in goal models. As opposed to a goal, which constitutes a state to be reached but leave open the question how it is going to be reached, a task directly points to the action that needs to be performed. For ex-

---

\(^1\)The “orthodox” way to express goals is by describing the desired state. Compare, for example, Schedule Meeting with Have Meeting Scheduled. The former represents rather an abstract activity to be performed while the latter describes the actual state to be reached. As we will argue in the next chapter, while the latter seems to correspond to the notion of goals more correctly, for our purposes both forms can be used interchangeably, as long as no information is introduced describing how the goal is to be performed (e.g. who, when, where, with what means). In fact, most goal analysts use the activity-based form.
ample Have Participants Notified is a goal in that many details as to how it will be achieved are left open. Instead Secretary to E-mail Cancellation to All Participants can be modeled as a task, as there little room of debate as to how this will be attained. In general, however, the boundaries between the two notions are not crisp and greatly depend on the objective of the particular modeling exercise.

The goal models we will consider in this thesis are graphs constructed using hardgoals, softgoals and tasks as nodes and a variety of relationships as edges. In Figure 3.1 such a goal model is depicted. Goals, soft-goals and tasks are depicted through oval, cloud-shaped and hexagonal elements respectively. The goal graph consists of two subgraphs: the hard-goal subgraph and the soft-goal subgraph which we discuss below.

Figure 3.1: A goal model
3.3.2 Hard-goal subgraph

Hard-goals form the hard-goal subgraph, which is a hierarchical structure of AND-decompositions and OR-decompositions. When we AND-decompose a goal into subgoals, we represent the fact that all subgoals need to be fulfilled in order for the parent goal to be considered fulfilled, too. Thus, in Figure 3.1, the goal Schedule Meeting cannot be attained unless both the goals Collect Constraints and Choose Meeting Time are somehow attained. The OR-decomposition, instead, is introduced when we want to show that the satisfaction of one of the children suffices for the satisfaction of the parent goal. For example, either of the goals System to Collect Constraints or Secretary to Collect Constraints suffices for the fulfillment of the goal Collect Constraints, thus we introduce an OR-decomposition. Leaf level nodes of a complete hard-goal subgraph are tasks and tasks can only be leaf level nodes of the hard-goals graph (thus, the graph of Figure 3.1 shows only part of the complete model).

Each solution of the AND/OR goal tree is a set of leaf level tasks that suffice for the fulfillment of the root goal according to the AND/OR structure. We call every such set of tasks an alternative, and the corresponding subtree alternative subtree. Notice that the number of alternatives that a hard-goal tree implies grows (in the worst case) exponentially with the number of OR-decompositions that appear in the tree.

3.3.3 Soft-goal subgraph

Soft-goals form their own sub-graph, the soft-goal subgraph, through the use of weighted contribution links. Recall that soft-goals do not have a clear cut criterion to be used for deciding whether they are satisfied or not. We can, however, assign them degrees by which we believe they are satisfied and denied, based on relevant evidence. Thus, to each soft-goal \( l \) we associate two variables: \( valS(l) \) which represents the degree by which we believe the soft-goal is satisfied and \( valD(l) \) which represents the degree by which we believe it is denied. The contribution links show how the satisfaction and denial value of one goal can influence our knowledge of the satisfaction or denial of other goals. Thus, through the use of contribution links we show how satisfaction/denial of one goal propagates to other goals of our model.

Contribution links are drawn between soft-goals, and also from hard-goals and tasks of the hard-goal sub-graph to soft-goals. Thus, every task and hard-goal that is the origin of a contribution link is considered to also be part of the soft-goals graph. Consequently, these hard-goals and tasks are also associated with a \( valS(\cdot) \) variable (albeit not a \( valD(\cdot) \) one),
which, as we will see later, has a limited domain of values compared to the one of soft-goals. We do not associate a \( valD(\cdot) \) value to hard-goals and tasks, as the notion of hard-goal or task denial is not intuitive for such elements.

In general, the domain of satisfaction and denial variables as well as the type of contribution links depend on the representation granularity we wish to achieve. The Giorgini et al. modeling and evaluation framework ([49]), which we follow in this thesis, offers two representation alternatives: a qualitative and a quantitative one. Below, we detail the specific modeling rules for each as well as their intuitive semantics.

**Qualitative Modeling Framework**

In qualitative modeling of soft-goal satisfaction propagation, the variable \( valS(\cdot) \) (respectively \( valD(\cdot) \)) take values in the domain \( \{F, P, N\} \), which mean Full satisfaction (resp. denial), Partial satisfaction (resp. denial) or No evidence of satisfaction (resp. denial) at all, respectively. It is assumed that these three values are totally ordered: \( F > P > N \). Thus, if \( valS(\text{Minimize Human Effort}) \) equals \( P \), this means that the goal Minimize Human Effort is partially satisfied. If \( valD(\text{Enhance Participatory Spirit}) \) equals \( F \), this means that the respective soft-goal is known to be fully denied. For hard-goals and tasks, for which partial satisfaction/performance is not defined, the domain is restricted to the values \( F \) and \( N \).

<table>
<thead>
<tr>
<th>Weak Contributions</th>
<th>Strong Contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \xrightarrow{+s} l_2 )</td>
<td>( l_1 \xrightarrow{++s} l_2 )</td>
</tr>
<tr>
<td>( l_1 \xrightarrow{-s} l_2 )</td>
<td>( l_1 \xrightarrow{-s} l_2 )</td>
</tr>
<tr>
<td>( l_1 \xrightarrow{+D} l_2 )</td>
<td>( l_1 \xrightarrow{++D} l_2 )</td>
</tr>
<tr>
<td>( l_1 \xrightarrow{-D} l_2 )</td>
<td>( l_1 \xrightarrow{-D} l_2 )</td>
</tr>
</tbody>
</table>

Table 3.1: Qualitative Propagation Links

Furthermore, there are eight types of contribution links between two soft-goals \( l_1 \) and \( l_2 \), seen in Table 3.1. In the table, the subscripts \( S \) and \( D \), represent whether it is the satisfaction or the denial of \( l_1 \) that is influencing \( l_2 \), respectively. Moreover, the sign of the propagation shows whether the link implies contribution of the corresponding variable to the same or the opposite variable of \( l_2 \), depending on whether it is positive ++ or negative −−, respectively. Finally, the number of signs, one (+−) versus two (+++−−), show weak and strong in-
fluence, respectively. Thus, in Figure 3.1, Minimize Human Effort $\xrightarrow{+s}$ Quality of Scheduling Process, means that achieving minimization of human effort in the process of scheduling a meeting weakly helps to achieve quality of scheduling process. On the other hand, Collect From Invitees Software Calendars $\xrightarrow{−s}$ Accuracy of Constraints means that pulling constraints from invitees software calendars strongly hurts the goal to collect accurate constraints. Absence of the subscript $S$ or $D$ implies that both possibilities are in effect. Thus, $l_1 \xrightarrow{+} l_2$ implies both $l_1 \xrightarrow{+s} l_2$ and $l_1 \xrightarrow{+D} l_2$.

The satisfaction and denial value of a soft-goal depends on the satisfaction and denial values of all goals that contribute to that goal through a link. More specifically, given a contribution link from goal $l_1$ to goal $l_2$, $valS(l_2)$ and $valD(l_2)$ are determined by the corresponding $valS(l_1)$ and $valD(l_1)$ values as well as the type of the contribution link, as shown in Table 3.2. We will later discuss the case of multiple contribution links targeting the same soft-goal.

<table>
<thead>
<tr>
<th>Contribution</th>
<th>$valS(l_2)$</th>
<th>$valD(l_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1 \xrightarrow{+s} l_2$</td>
<td>$\min{valS(l_1), P}$</td>
<td>N</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{++s} l_2$</td>
<td>$valS(l_1)$</td>
<td>N</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{−s} l_2$</td>
<td>N</td>
<td>$\min{valS(l_1), P}$</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{−−s} l_2$</td>
<td>N</td>
<td>$valS(l_1)$</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{+D} l_2$</td>
<td>N</td>
<td>$\min{valD(l_1), P}$</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{++D} l_2$</td>
<td>N</td>
<td>$valD(l_1)$</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{−D} l_2$</td>
<td>$\min{valD(l_1), P}$</td>
<td>N</td>
</tr>
<tr>
<td>$l_1 \xrightarrow{−−D} l_2$</td>
<td>$valD(l_1)$</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 3.2: Qualitative Contribution Links

Quantitative Modeling Framework

The quantitative framework allows more fine-grained analysis of satisfaction/denial propagation by using real numbers instead of labels $N$, $P$ and $F$. Thus, the domain of the variables $valS(\cdot)$ and $valD(\cdot)$ is the set of real numbers in the interval $[0, 1]$. Again, however, specifically for hard-goals and tasks, the domain is restricted to the values 0 and 1.

The contribution links we use when modeling for quantitative analysis can be seen in Table 3.3. Intuitively, $l_1 \xrightarrow{ws} l_2$ (respectively, $l_1 \xrightarrow{ws} l_2$), denotes that the satisfaction (respectively, denial) of $l_2$ is understood to be equal to $l_1$’s satisfaction factored by $w$. Similarly, $l_1 \xrightarrow{wd} l_2$
Contributions of Satisfaction  | Contributions of Denial  
--- | ---  
Contributions to Satisfaction  |  
\( l_1 \xrightarrow{w_{S}} l_2 \)  |  
\( l_1 \xrightarrow{w_{D}} l_2 \)  
Contributions to Denial  |  
\( l_1 \xleftarrow{w_{S}} l_2 \)  |  
\( l_1 \xleftarrow{w_{D}} l_2 \)  

Table 3.3: Quantitative Propagation Links

(respectively, \( l_1 \xrightarrow{w_{D}} l_2 \)), denotes that the denial (respectively, satisfaction) of \( l_2 \) is calculated as a proportion of \( l_1 \)'s denial. Again, the value of \( valS(l_2) \) and \( valD(l_2) \), depending on the respective values of \( l_1 \) and the type of contribution link from \( l_1 \) to \( l_2 \), are decided based on rules which are shown in Table 3.4. Note that, while [49] discusses several possibilities for interpreting quantitative propagation, Table 3.4 reflects the probabilistic approach. More details on other interpretations can be found in [49]. Again, as in the qualitative case, we omit the subscript \( S \) or \( D \) to denote coexistence of contributions of both satisfaction and denial. Thus, \( l_1 \xrightarrow{w} l_2 \) implies both \( l_1 \xrightarrow{w_{S}} l_2 \) and \( l_1 \xrightarrow{w_{D}} l_2 \).

<table>
<thead>
<tr>
<th>Contribution</th>
<th>( valS(l_2) )</th>
<th>( valD(l_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l_1 \xrightarrow{w_{S}} l_2 )</td>
<td>( w \times valS(l_1) )</td>
<td>( w \times valD(l_1) )</td>
</tr>
<tr>
<td>( l_1 \xrightarrow{w_{D}} l_2 )</td>
<td>( w \times valD(l_1) )</td>
<td>( w \times valS(l_1) )</td>
</tr>
</tbody>
</table>

Table 3.4: Quantitative Contribution Links

Figure 3.2 shows a version of the meeting scheduling goal model featuring quantitative contribution links.

**Expressiveness versus Simplicity**

The above presentation aims at illustrating the satisfaction propagation frameworks in their full expressive power. In practice, however, users may desire a notation that is less expressive yet simpler and easier to use. The most effective simplification strategy is to avoid working with two values (i.e. with both satisfaction and denial). Hiding the existence of two separate values can be achieved at two levels.

Firstly, a way to aggregate the satisfaction and denial values can be employed. In the quantitative framework this can be done by subtracting the denial value from the satisfaction
value and considering the result as the general satisfaction value for the soft-goal which is now in the interval $[-1, 1]$. A similar aggregation rule can be constructed for the qualitative framework through the introduction of the appropriate axioms. Note that the essence of the satisfaction and denial propagation remains the same two-variable one. However, the existence of two variables is hidden from the user who is presented with only one general value.

Secondly, at the modeling level, avoidance of exclusive propagation of satisfaction or denial by avoiding use of the corresponding subscripts can help simplify the construction of the soft-goal graph. This can be possible, as in most of the cases propagation can be assumed to be symmetric, that is both satisfaction and denial are propagated by the same degree. For example, in Figure 3.2, the soft-goal Minimize Human Effort appears to influence the soft-goal Quality of Scheduling Process only through its satisfaction, assuming that failure to minimize human effort does not necessarily imply a compromise to the quality of the scheduling process. However, such assumptions that lift symmetry can be considered at later stages of modeling and provided that specific information that necessitates them is given in the domain. For an initial or rough analysis, analysts can begin by assuming symmetric
contributions (i.e. without the S or D subscripts), increasing thereby the comprehensibility of the resulting model.

Following these simplification practices in the qualitative framework, the level of informality and comprehensibility of the soft-goal graph approaches that of $i^*$ ([126]).

### 3.4 Label Propagation

The purpose of introducing the propagation rules of Tables 3.2 and 3.4, apart from providing an intuition of what satisfaction/denial contribution means, is that it allows us to reason about satisfaction or denial of certain soft-goals in our soft-goal subgraph based on evidence that we have about the satisfaction or denial of the others. In [49], Giorgini et al. introduce such an algorithm, called the label propagation (LP) algorithm. Starting from initial satisfaction and denial values for goals that are sources to the graph, the LP algorithm iterates over the propagation rules until convergence for the satisfaction/denial degrees of all goals is reached. At each iteration, when a soft-goal is a target of many contribution links, from all potential satisfaction and denial values (including the existing ones), the maximum one (by absolute value) is selected to be the new value. We sketch this algorithm in Figure 3.4, which has been adapted from [49].

In this thesis, we will use an adapted version of the label propagation algorithm presented in [49], which assumes that the soft-goal graphs do not contain directed cycles. We identify our algorithm as ALP (LP for Acyclic goal models). The additional acyclicity assumption allows us to change the original Label Propagation algorithm in a way that guarantees convergence within one iteration. To achieve this, for each goal node we calculate its depth, that is the maximum path length for reaching the node from any of the sources of the graphs (which are all hard-goals, tasks or CEs in our case). In Figure 3.3 the soft-goal subgraph of the goal graph of Figure 3.1 is shown, where each node is annotated with a number indicating its depth. Thus the soft-goal Quality of Scheduling Process has a depth value of 2.

Hence the ALP algorithm includes three changes, compared to LP. Firstly the label updates are (partially) ordered by maximum path length ascending. Secondly, the update does not (need to) take into account the current label of each node. Therefore, thirdly, only one iteration is needed.

In Figure 3.5, the pseudocode describing the algorithm is given, next to the original one presented in [49]. In the figure, $\text{withdepth}(G, d)$ returns a set of nodes whose depth equals $d$ or $\text{NULL}$ if no nodes of such depth exist. Also, $\text{Label}$ denotes a pair of satisfaction and denial.
values, a LabelSet $C$ is a set of such Labels, $C_g$ denotes the Label in $C$ that is associated with goal $g$, and $candS^j_g$ and $candD^j_g$, are arrays of candidate satisfaction and denial values, respectively, for goal $g$.

### 3.5 Goal Alternatives

The combination of functional goal decompositions with qualitative satisfaction analysis of quality goals can be exploited for the purpose of evaluating alternative requirements for fulfilling stakeholder objectives ([87, 77]). Such evaluation is possible by calculating alternative solutions of the decomposition structure and evaluating the impact each has to the soft-goals. In [62], this idea is applied for personalizing requirements models and a naive way to evaluate the soft-goals that directly relate to the decomposition structure is presented.

We will here offer a more elaborate way to assess how alternatives of the hard-goal graph influence the satisfaction of soft-goals, through actual use of the label propagation algorithm. As we implied above the AND/OR-tree implies a number of sub-trees, fulfillment of which suffices for the fulfillment of the root goal. Given an alternative subtree, we can assign a

---

**Figure 3.3: Maximum Path Length**

---
satisfaction value of $F$ or 1.0, depending on whether we use the qualitative or the quantitative framework respectively, to all those goals or tasks that are both part of the soft-goals subgraph and part of the alternative subtree. Similarly we assign 0.0 or $N$ to the hard-goals and tasks that are not part of the alternative subtree. We can then execute the label propagation algorithm to see how the particular alternative influences the satisfaction and denial of high-level soft-goals. In Figure 3.6 two such alternatives are shown. Each goal or task participating in the alternative and the soft-goals graph, is annotated with its corresponding satisfaction and denial values $\langle \text{val}S(\cdot), \text{val}D(\cdot) \rangle$. Goals or tasks that do not participate in the alternative can be treated as inexistent in the subsequent analysis.

The satisfaction or denial value that soft-goals acquire can be used for comparing alternatives. In Figure 3.6, for example, we can observe that while the first alternative has a slightly smaller degree of denial of the goal Happy Invitee compared to the second one, the latter better helps to Enhance Participatory Spirit amongst potential participants. Hence, if we were asked which of the two alternatives to choose, if enhancing a participatory spirit were a concern of priority, we would most probably choose the second. In Chapter 5 we will present a language for specifying such priorities and a mechanism for choosing the alternative that best satisfies such priorities.
3.6 Summary

In this chapter we presented the goal modeling language based on which we will develop our techniques. It is based on a restricted version of the $i^*$ modeling framework [126] and adopts the semantic and analysis framework presented by Giorgini et al. in ([49]). Goal models in our language consist of a hard-goal subgraph which is an AND/OR decomposition structure, and a soft-goal subgraph which is used for assessing the quality of each alternative in the AND/OR subgraph. The assessment is preformed by establishing contribution links among soft-goals which show how the satisfaction of each influences the satisfaction of others. We described two ways to quantify and calculate satisfaction propagation according to [49] and sketched how this can help us assess the qualities of alternatives.

In the following two chapters, we focus on different parts of the goal models and further extend and elaborate the basic elements and reasoning techniques we provided here. Thus in Chapter 4, we will focus on the hard-goal subgraph, introduce some extensions and variations and propose a way to translate the result in propositional logic. This will allow time-independent bottom-up reasoning about alternatives in the hard-goals graph. In Chapter 5, we will introduce a different version of these extensions which aim at offering a temporal dimension to our goal models. Hence, instead of propositional logic we will use situation calculus to define the semantics of the goal models. In addition, a formal preference specification language will allow top-down selection of alternatives, through the specification of priorities over desired satisfaction of soft-goals.
Chapter 4

Acquiring Goal Variability

4.1 Overview

In this chapter, we take a deep look into the types of variability that can be encoded in goal models and, based on our findings, we propose a variability-intensive process for decomposing and analyzing goals. This process aims at systematizing goal variability acquisition and at attaining completeness in the result, while allowing their representation in a concise manner. In addition, it allows reasoning about alternatives while taking into account the circumstances that hold in the context of attaining a goal.

We organize our presentation in this chapter as follows. In Section 4.2 we take a closer look at the meaning of goal variability and how the literature has approached this question. In Section 4.3 we present our motivating example, and in Section 4.4 we discuss the adaptations we make to the modeling language of Chapter 3 to serve our purposes. Then, in Section 4.5, we introduce the notion of variability concerns and a general categorization thereof, while in Section 4.6 we show how such categorizations that are specific for particular domains can be constructed. We also discuss the role of non-intentional variability in Section 4.7. Finally, we present a concern-driven decomposition process in Section 4.8, describe our experiences in applying it in Section 4.9 and conclude the discussion in Section 4.10.

1The material presented in this chapter has been published as a full paper at the 14th IEEE International Conference on Requirements Engineering (RE’06). See [80] for full citation.
4.2 Goal Variability: a Closer Look

4.2.1 On the Notion of OR-decomposition of Goals

We argued in previous chapters that goals are capable of concisely representing a great number of alternatives in a single goal tree, thanks to the existence of OR-decompositions of goals. These goal decompositions emerge naturally in the process of developing the goal models, by exploring alternative ways by which the parent goal can be satisfied. In our meeting scheduling example of Chapter 3, constraints for scheduling the meeting can be collected either by the System or by the Secretary and thus the goal Collect Constraints is OR-decomposed into two sub-goals. We clearly see, though, that what introduces the need for two alternative sub-goals is the opportunity to consider alternative agents that can perform the activities associated with fulfilling the parent goal (Secretary versus System). As another example in the same problem, there are several competing algorithms for choosing a meeting time. For instance, one is by avoiding conflicts that include significant participants and another is by avoiding all possible conflicts (which may lead to less convenient time for the significant participants). The introduction of alternatives here clearly does not relate to the agents who will perform the calculation but to the instrument by which this calculation is going to be performed; this instrument is an algorithm in our case. Clearly, a better understanding of the meaning of OR-decompositions of goals would shed more light on how these emerge in the decomposition process and could, hence, allow us to construct a systematic account for introducing decompositions in goal models.

4.2.2 Origins and Meaning of Variability in Requirements

We will now argue that the problem of the meaning and origin of requirements variability has not enjoyed appropriate attention in the software engineering literature, at least in a way that would result in interesting opportunities for facilitating the variability identification process. As we saw in Chapter 2, in Product Lines, the result of the largely informal Domain Analysis process ([97, 90]) is most popularly represented through feature models ([71, 29, 72]), which represent admissible combinations of user-visible characteristics of the system-to-be (the features) in a concise hierarchical manner. However, in order to be capable of representing many and diverse types of variability in a single view ([7]), feature models have deliberately relaxed semantics ([29]), which makes them inappropriate for representing the structure, the behavior or other characteristics of the required system. This, in turn, limits the capability of feature
models to be used as tools for variability identification, because it is detached from the views where variability really emerges. For example, a variation point in software behavior, will more likely emerge while thinking of and modeling the behavioral aspect of the system, through for example a statechart or a message sequence diagram, rather than when drawing the feature model; the latter plays the role of elegantly “cataloguing” variability, which is however identified and explained (given precise semantics) elsewhere. Thus, researchers have been proposing the integration of feature modeling with requirements-based variability discovery methods. In [52], for example, use cases are proposed as a main way to capture variability while existing feature models give analysts a hint on how variability of past (good) systems was organized. Another possibility of capturing variability in use cases is also discussed in [53], where variation points are introduced within use case diagrams. We refer the reader to Chapter 2 for more examples of such research efforts.

Few of these approaches, including feature models, provide an account of the meaning of variability. They inform about the existence of variation points, and how they can be possibly bound, but they do not attempt to explain the variation points at a meta-level. Categorizations of variation points that have been introduced refer mostly to the scope in which the variation points will appear or the software view they concern. Perhaps the most elaborate categorization is presented in [53] where distinctions such as functional versus behavioral variability are made. But such distinctions are too rough and independent of the domain that is being modeled, which, as it will become apparent below, limits their usefulness. In addition, as we already argued, these approaches define variability in terms of varying characteristics of the system-to-be, and not in terms of the \textit{causes} of these variations, i.e. the varying characteristics of the problem, the stakeholders and their needs.

In the goal oriented requirements engineering literature, OR-decompositions have not been an explicit subject of study, either. As we saw, in \textit{i*} ([126]), an OR-decomposition is understood as alternative \textit{means} (subgoals) by which a certain \textit{end} (parent goal) can be met; thus OR-decompositions are collections of means-ends links. In GBRAM ([4]), on the other hand, alternative sub-goals appear as alternative responses to questions that arise from the generic type of goals, but this is still far from being an explicit study of OR-decompositions of goals. Finally, in KAOS, despite work on AND-decomposition patterns ([32]) similar patterns for OR-decompositions have not been introduced.

Nevertheless, an important step towards understanding OR-decompositions has been made in [103] and [102], where Rolland et. al. show that OR-decompositions can have more specific semantics. A semi-structured formulation of the goal description, adopted from [96], allows
the analyst to define a number of aspects with respect to which variation may be possible. For example, alternative times, locations or beneficiaries for the fulfillment of a parent goal may each lead to an alternative subgoal. However, there is no discussion on where such variation aspects can be found. The authors go on to propose automatic generation of OR-subgoals by exhausting all possible combinations of alternatives of such aspects and pruning the result though identifying dependencies between these aspects. The dependencies assume that the relevance of a variability aspect depends on values given in other aspects. However, most inapplicable alternatives seem to be filtered out through informal examination of the result. As reported in [102], even after extensive pruning, the discovery procedure may lead to OR-decompositions with as many as 40 subgoals.

### 4.2.3 Non-intentional Variability

An additional element in understanding the origin and meaning of OR-decompositions is the variability of elements in a problem that are not intentional. In fact, the selection of alternative subgoals for fulfilling a parent goal is often determined by external factors. Consider again the Meeting Scheduling example and the goal Collect Constraints. We saw that either the Secretary or the System can collect the constraints of participants. If however the Secretary is non-existent or absent then the choice of the System as the agent to perform the constraint collection is unavoidable. In other words, the choice of a particular sub-goal is not intentional but necessary under the given circumstances. In Chapter 1 we referred to these as the “details and parameters” that need to be specified when applying the generic goal model in a particular problem instance. A particular binding of these parameters restricts the choices that exist at an intentional level (i.e. choices that depend solely on the stakeholders intentions). Conversely, knowledge of possible values of these parameters and the way they restrict intentional choices may lead us to the need for introducing more such choices. In our example, if combined absence of Secretary and System is possible, then a third constraint collection agent must be devised, if we want the goal Collect Constraints to be possible under such circumstances. Thus, identification of non-intentional parameters, how they vary and how they influence selections at the intentional level is an integral part of goal variability identification.
4.3 Motivating Example and Research Approach

Let us now introduce a more detailed example to use throughout the rest of the chapter. The example is from the health care domain. The context of the application is a geriatric assessment unit, where elderly patients with moderate to severe health issues are hospitalized for an amount of time. The objective of the system-to-be is the monitoring of the patients’ movement on the bed and around the unit so that nurses and doctors both maintain awareness of the patients’ health condition and are appropriately alerted when their services are required. The latter is particularly needed for the nurses, who must administer a care plan defined by the doctors, and respond to a variety of events, including cases in which the patient is in danger.

An example of an event that needs such immediate response is when a patient with hypotension is trying to get off the bed by herself. She will probably fall down immediately after she is up. Thus, when an attempt is being made, somebody needs to be notified that the patient tried to get off the bed, and then rush and prevent it. But there are many ways by which this process can vary. For sensing that the patient is trying to get up, for example, one option is to install specially designed sensors that trigger the event automatically. Another option is to have a camera in the patient’s room and a screen at the nursing station allowing manual monitoring and firing of the alarm. When it comes to the notification, there are more alternatives on whom to notify (the nurse that is assigned to the patient or any nurse that is close?), how to notify (using rooms’ speakers or devices wearable by the nurses?), where to notify (among the rooms of the unit?), how intensively (loudly) to notify, etc.

Thus, the goal be notified comes with a list of variability issues (who?, where? etc.), which need to be tackled through decomposing the goal. But on what grounds can one assume a complete list of issues and how can the goal be decomposed to address all of them, while avoiding a combinatorial explosion problem? Further, non-intentional variables, such as how severe the patient’s condition is or where her nurse currently is and what he is doing, obviously play a significant role in selecting the appropriate alternative for each of the above variability issues. But how can one systematically introduce them in the goal model, and then use them to reason about alternatives that are applicable in given circumstances?

The methodological approach that we followed in order to address these questions is based on our exploratory study in the Geriatric Assessment Unit of the Queen Elizabeth II Health Sciences Centre in Halifax, Canada. The example scenario above is taken from this study and slightly adapted to better illustrate the techniques. We conducted a set of interviews of various stakeholders (doctors, nurses and occupational therapists) that worked in the unit. This allowed
us to acquire a clear picture of the domain, the processes that take place in the unit, as well as what the associated health professionals would think would constitute improvements in the current situation. We also performed some limited observation of the nurse activities in the unit.

Having a realistic problem at hand, we started developing goal models focusing on two aspects of the goal modeling process: a) the emergence of OR decompositions in the goal models and b) the role of non-intentional facts in both defining and selecting alternatives. Two hypotheses were set. Firstly, that systematically introducing OR-decompositions in a goal decomposition model based on a “checklist” of categories would be possible and would facilitate the process of discovering alternatives. Secondly, that there is a possible way to model and use non-intentional facts that illustrates how the latter influence both the introduction and the selection of alternatives in the goal model. In both hypotheses the term “possible” describes both the ability of using the elements of the language per se (both the original and the extensions we introduced) as well as the corresponding reasoning techniques and the ability to produce models that are potentially valid. Thus, our research effort was directed towards understanding the particular modeling and reasoning techniques that would allow us confirm these hypotheses based (at this stage) on our own observation and assessment.

4.4 The Goal Modeling Language

The diagrammatic formalism we will use to represent goal models is the one introduced in Chapter 3, with a few restrictions and extensions. Firstly, we will focus only on hard-goals and the hard-goal subgraph, ignoring soft-goals and partial goal satisfaction. This will allow us to perform basic reasoning about alternatives via translating our hard-goal graphs to propositional logic. Secondly, strong contribution links however are used among hard-goals, allowing propagation of either full or no satisfaction at all. Thirdly, a special type of element is used for representing non-intentional domain facts and how they relate to goal satisfaction.

An example of such a model is seen in Figure 4.1. The backbone of the model is an AND/OR decomposition tree, containing goals (the ovals in the figure) can be either satisfied or not satisfied. As previously, when a goal $g$ is AND-decomposed into $g_1, \ldots, g_n$ then $g$ is satisfied iff $g_i$ are satisfied for all $i$. If $g$ is OR-decomposed, it is satisfied iff there exists an $i$ such that $g_i$ is satisfied.

In addition, several types of links are used to represent constraints among goals. Thus, given two goals $g_1$ and $g_2$ the links $g_1 \rightarrow g_2$ and $g_1 \rightarrow g_2$ show that when $g_1$ is satisfied then
$g_2$ is or is not satisfied respectively. The link $g_1 \xrightarrow{++} g_2$ (respectively, $g_1 \xleftarrow{-} g_2$) is equivalent to having both $g_1 \xrightarrow{++} g_2$ ($g_1 \xleftarrow{-} g_2$) and $g_2 \xrightarrow{++} g_1$ ($g_2 \xleftarrow{-} g_1$) at the same time. Roughly, these constraint links correspond to $++$ and $-$ contribution links we saw in Chapter 3 but applied to hard-goals, which are assumed to have a crisp satisfaction value and they do not define a denial value.

Further, the link $g_1 \xrightarrow{pre} g_2$ indicates that $g_2$ cannot be satisfied unless $g_1$ is satisfied. Also, when there is a need to represent more elaborate constraints we can use rectangles in which we can construct condition formulae using more than one goal. We call these elements Condition Elements (CE). From each of such condition elements $c$ we can then draw $c \xrightarrow{++} g$, $c \xrightarrow{-} g$ or $c \xrightarrow{pre} g$ links to one or more goals $g$ of the model.

Notice that soft-goals and the associated soft-goal graph falls beyond our interests in this aspect of our framework, and so does partial label propagation and, consequently, the label propagation algorithm. We instead translate our diagrammatic language to common two-valued propositional logic.

![Figure 4.1: A goal model](image)

We can associate each goal with a propositional literal and represent the satisfaction of the root goal in terms of a propositional formula $G \equiv S_g \land C_g$. $S_g$ represents the AND/OR structure in terms of leaf level literals. Each non-leaf node is recursively replaced by the conjunction or disjunction of its children depending on whether the decomposition is AND or OR, respectively. $C_g$ represents the additional constraint links. Each constraint link in the model results in a conjunct in the formula $C_g$ as follows:
### Link Type

<table>
<thead>
<tr>
<th>Link Type</th>
<th>Conjunct</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>$g_1 \implies g_2$</td>
</tr>
<tr>
<td>$\mathit{pre}$</td>
<td>$g_1 \implies \neg g_2$</td>
</tr>
<tr>
<td>$\iff$</td>
<td>$g_2 \iff g_1$</td>
</tr>
<tr>
<td>$\iff$</td>
<td>$g_2 \iff \neg g_1$</td>
</tr>
</tbody>
</table>

In the first three cases, $g_1$ can be a condition formula instead of a single goal. In all cases, literals representing non-leaf nodes are replaced with clauses that contain only leaf nodes according to the AND/OR structure.

As we have seen, an alternative in a goal model such as the one in Figure 4.1, is a solution of the AND/OR tree, that is, a subgoal choice for each of the OR-nodes, that satisfies the constraints. Finding alternatives is understood as finding truth assignments that satisfy the resultant propositional formula $G$. Thus, in Figure 4.1, configuring the OR-decompositions to the children $\{g, n, f\}$ satisfies the root goal.

As we will see, the formalization we presented at this Section will allow simple time-independent reasoning about goal alternatives, which can be sufficient for some applications, including identification of cases in which enrichment of the goal tree with more alternatives is required. Nevertheless, a more powerful and expressive way to perform such reasoning by also taking into account the time ordering of goal fulfillment will be presented in the next Chapter.

### 4.5 Variability Concerns for Goal Decomposition

Since goals express desired states of affairs, they are normally descriptions of something that needs to be true in the world, for example Message is Sent or Light is ON. However, semi-formal goal decomposition calls for a gradual shift of focus from the desired state of world that a goal describes, to the human or machine activities that can potentially satisfy the goal. Thus, in most of the cases, the above goals will be seen phrased as Send a Message or Turn Light ON. In that spirit, a goal can be understood in terms of a generic activity, and its analysis as the process of specifying this activity better.

Consequently, when a high level goal is phrased, the generic activity it requires is necessarily vague and incomplete. For example consider the goal Send a Message. Who will Send a Message? To whom? When? Where? How fast? What message? Such questions describe concerns that call for alternative responses and, consequently, alternative refinements.
of the goal. A study of the possible types of such concerns can be greatly facilitated by looking at categorizations of semantic roles of sentence elements, as they are studied in Linguistics.

We use Fillmore’s case system ([42]) as a basis for understanding language semantics in a requirements engineering context; though, here we focus on goals. According to Fillmore, a simple sentence consists of a verb and a set of noun phrases. Each noun phrase holds with the sentence a relationship of a particular semantic type. Linguistically, these types correspond to different *cases*. Fillmore proposes that there exists an essential set of such case types that fits in the case system of every known language. Each of these universal case types addresses a particular semantic concern associated with the verb of a sentence. Hence, they can be seen as a set of potential semantic slots that may or must be associated with each verb, and filled whenever the verb is used in a complete sentence. This way, given a verb, a *frame* feature can be defined, which is a set of such semantic slots (*frame elements*) that the verb “evokes”.

For example, the verb “open” is necessarily associated with an *objective* slot (“*what* opens/is opened?”) but may also be associated with an *agentive* slot (to answer “*who* opens?”) and an *instrumental* one (to answer “open *with what?*”).

Considering the verb that describes the generic activity in a goal description, the discovery of alternative goal refinements can be driven by the frame that is associated with that verb and the corresponding elements. In this context, the frame elements can be seen as *variability concerns*, that is, types of questions whose alternative answers result in alternative refinements of the original goal. In the *Send Message* example, the agentive variability concern asks who sends the message. For each potential *response* to the variability concern (for example “the user”, “the machine”, “the administrator”, “the user and her assistant together”) an alternative refinement of the goal needs to be introduced. The collection of all concerns relevant to a goal is the *variability frame* evoked by the goal.

Based on Fillmore’s idea of defining a universal set of frame elements, we can introduce a general set for variability concerns, to be used for the construction of variability frames for goals. The set we constructed includes most of the semantic types Fillmore proposes, but also draws information from adjunct classification schemes that are frequently discussed in grammar books (we used [98] and [61]) as well as from the way goals are formalized in the goal analysis literature. Thus:

**Agentive** (*A*) is the concern of the agent(s) whose activities will bring about the state of affairs implied by the goal description. Responses to the concern are typically actors or combinations of actors found in the domain, including the system(s)-to-be. For example, \{Machine, User alone, User Supported\} \_\_A to choose schedule. Alternative responses to the agentive con-
cern are essentially alternative delegations of a goal to actors (including the system-to-be).

**Dative** ($D$) is the concern of the agent(s) who will be affected by the generic activity implied by the goal. As above, responses to the concern are typically actors or combinations of actors found in the domain, including the system(s)-to-be. Examples are Send a message to \{the admin, the user\}$_D$, Notify \{designated nurse, nurses at nursing station\}$_D$.

**Objective** ($O$) is the concern of the object(s) that is affected by the generic activity implied by the goal. Example are: Send \{an e-mail message, a fax message\}$_O$, Print \{a full report, a summary\}$_O$.

**Factive** ($F$) is the concern of the object(s) or being(s) that is/are resulting from the activity or understood as part of the meaning of the verb. Examples: Format Text \{bold, italic\}$_F$ or Turn light \{on, off\}$_F$.

**Process** ($P$) is the concern that determines the instrument ($P\.ins$) that is involved in the performance of the generic activity implied by the goal, as well as the means ($P\.mea$) and the manner ($P\.man$) by which the activity is performed. The subcategory $P\.mea$ is the concern to which “pure” means-ends variability should be classified. Some examples are: Pay \{by debit, by credit, by cash\}$_{P\.mea}$, Meet new people \{by organizing activities, by participating in activities\}$_{P\.mea}$ or Notify User \{loudly, subtly\}$_{P\.man}$.

**Locational** ($L$) is the concern about the spatial location(s) where the generic activity that is implied by the verb is supposed to take place. Example: Send a message \{in the Car, on a Bus\}$_L$.

**Temporal** ($T$) is the concern about the duration ($T\.dur$) or frequency ($T\.frq$) of the generic activity that is implied by the verb. For example: Check for messages every \{hour, 10 mins\}$_{T\.frq}$, Suspend Notifications for \{2 hours, 10 mins\}$_{T\.dur}$. Temporal location is dealt through the next concern.

**Conditional** ($C$) concerns refer to either alternative conditions under which the goal can be fulfilled ($C\.con$) or alternative triggers of the generic activity associated with the goal ($C\.tri$). For example: Ship product only if \{order has arrived, payment has arrived\}$_{C\.con}$ or Notify user \{when message arrives, in regular intervals\}$_{C\.tri}$.

**Extent** ($E$) variability concerns refer to alternative degrees by which the generic activity can be performed (excluding duration). For example: Display the first \{10,20,10%\}$_E$ \(of\) records.

The set is by no means a template for structuring goal phrases, as in [102, 96], but a catalogue of categories that can help analysts understand the variability aspects of goals.
4.6 Constructing Problem-specific Variability Frames

Using a general set of variability concerns to characterize variability for arbitrary goals may come with certain drawbacks. Firstly, the concerns are not guaranteed to be equally intuitive for every goal for which they are to be used, due to the necessary generality they must demonstrate. Secondly, in order to fit a great number of cases, they are necessarily coarse-grained and may ignore certain variability aspects that arise when examining individual goals in detail. Maintaining catalogues of variability frames that are specific to goals and the problem domains they appear in is a way to cope with these issues.

In the area of semantic frames, the construction of a frame lexicon has proved possible in FrameNet ([8]). FrameNet introduces a lexical database for English that provides the meaning of words in terms of the semantic frame they evoke. The lexicon contains a large set of semantic frames that are to be used for this purpose. Each frame contains its own set of elements that are specific to the frame’s semantics. Hence, if we are given a frame associated with a goal, by simply interpreting the frame elements as variability concerns, we can construct a customized variability frame for a particular goal.

The association of a goal with a semantic frame is again based on the verb of the goal phrase, but other key words of the phrase may play a role as well. In practice, multiple frames may be considered for a single goal. Consider for example a health organization posing the goal Be Aware of Patient’s Condition. By consulting FrameNet, one discovers that there are two frames that are related to it, namely awareness and becoming aware. Both contain elements that are very useful in identifying interesting variability concerns: the cognizer is the person who wants to be aware (e.g. {doctors, nurses, family}), the phenomenon is the situation of which the cognizer(s) wants to become aware (e.g. {a complication, a delirium, a fall}), evidence refers to the particular observations that allow the cognizer to become aware (e.g. {patient not responding, patient wandering in the unit}), while state is the state of the phenomenon when the cognizer become aware of it (e.g. for the case of a patient’s fall: {is about to fall, is currently falling, has fallen}). Although none of these concerns belongs to the universal set we discussed earlier, they arguably do a better job in describing fine-grained variability aspects of the goal originally stated by the stakeholders.

But how can such a frame lexicon be built? The construction of FrameNet is based on the examination of a large corpus of English texts. Once several sentences in which a particular word is found are collected, they are subjected to annotation, which is, roughly, a classification of the phrases that are surrounding the word into frame elements. Associating the word with
a particular frame (which may include devising a new frame or specializing an existing one) also requires human involvement and intuition. Due to the inherent labor-intensiveness of the endeavour, FrameNet is still under construction (2006). However, its development to date shows that given a corpus of attested sentences, it is possible to construct a lexicon of semantic frames.

In the context of requirements analysis, frames that are particular to goals can be constructed the same way. The motivation for doing so is not only the potential incompleteness of general purpose lexicons such as FrameNet, but mainly the construction of frame lexicons that are more informed with respect to particular problem domains. Such specialized frame lexicons can be constructed by referring to documents that describe the processes of the domain of interest or by examining evidence from past projects, particularly artifacts of early elicitation efforts (e.g. interview transcripts, reports etc). The analysts can then identify sentences in which words of interest appear, and annotate the surrounding phrases appropriately. Thus, the resulting frames are based on a corpus that is specific to a domain scoped by the analyst herself.

Consider for example that we want to decompose the goal Schedule a Graduate Meeting in the context of a graduate educational program. Suppose also that we are particularly referring to the meetings that relate with students’ progress in the graduate program (progress meetings, checkpoints etc). In the department of Computer Science at the University of Toronto, we found three documents that describe how this process should be performed, namely, the Graduate Handbook issued by the department, the Graduate Calendar issued by the University-wide umbrella organization of all graduate programs, called the School of Graduate Studies (SGS), as well as a text providing Graduate Supervision Guidelines, again issued by SGS. This collection of texts, which amounts to a total of about 18,000 words, was then subjected to searches of words that relate to the goal in question. We considered the words meet and examination (the latter being an alternative way to refer to several formal meetings) and looked in the context in which they appeared. We found a total of 41 and 116 sentences where these words occurred in their various forms, respectively. By examining the phrases that accompanied the words in each sentence we were able to collect a number of standard semantic elements that define such a meeting.

For instance, in the sentence “The departmental \{thesis\}_M examination is open to \{all students and faculty members of the department\}_N”, discloses two elements that vary in examination meetings, namely, the material that is central to the meeting, denoted with M (here it is a thesis but can also be a progress report, a literature review, etc.) as well as the “openness” of
the meeting in terms of who is allowed to attend it, denoted with \( N \). By working this way with all sentences, we collected 21 variability concerns, some of which are: *Purpose*, *Language*, *Frequency*, *Formality*, *Duration*, *Participants*, *Openness*, *Temporal Location in Year*, *Temporal Location in Graduate Program Timeline*, *Agent Responsible for Organization*, *Agents to be Notified*, and *Meeting Material*. We call our frame *Graduate Meeting*. Each of its elements has a special meaning that is specific to the domain. For example, alternative responses to the concern *Participants* can be \{Core Committee, Core Committee with Student, Extended Committee with Student\}, whereas the concerns *Agents to be Notified* may be \{Nobody, The Grad Office, The SGS\}.

Potential semantic elements that are absent can be derived from frames that appear in FrameNet through frame inheritance. A frame may inherit all elements of another frame and introduce its own elements. Moreover, certain semantic elements may be elaborated, i.e. made more specific. Thus, *Graduate Meeting* inherits elements from FrameNet’s *Congregating*, and hence have frame elements currently absent from the former, such as the *Place* of the graduate meeting, be drawn from the latter. Elaboration is performed by changing the name of an inherited frame element to a more specific one. Thus, the element *Individuals* that is part of *Congregating*, is renamed into *Participants* for *Graduate Meeting*, still leaving space for further elaboration.

### 4.7 Domain Variability

By the term *domain variability* we refer to facts about the domain of discourse that unintentionally vary in the context where the fulfillment of a goal is attempted. For example, a user may want to send a message while being at a particular place, a particular time, doing something specific or being capable of doing certain things such as hearing or speaking in a particular language. Such facts are circumstances under which the goal will need to be achieved and, as we saw, they constitute factors that may influence both the identification of new alternatives when an OR-decomposition is attempted and their selection thereafter. Thus, the goal *Fire a Loud Audio Notification* may presume that there are no people sleeping around the agent to be notified (e.g. in a hospital at night). Conversely, knowledge of the possibility that there might be a case of an agent that needs to receive a notification while being around people who sleep, calls for the identification of additional alternatives that can bypass these constraints.

Our experience showed that domain variability can be effectively identified by focusing
Chapter 4. Acquiring Goal Variability

on three basic entity types in the domain of discourse: *agents* (e.g. Nurse, System-to-Be, Unit Administrator), *locations* (understood here as a synonymous of “contexts”, e.g. Graduate Lounge, Street, Nursing Station) or *objects* (e.g. the Line at the Bank, an Incoming Message, a Driver’s License). Domain variability is then formulated in terms of attributes of each of these agents, locations and objects or relations among them. Of course, the analyst will focus on these attributes and relations that may vary. More specifically:

**Agent Characteristics** refer to varying properties of agents (including the system to be) such as their location, `isAtLocation(Agent, {home, office, bus})`, their skills/capabilities, `hasCapability(Agent, {hear, access_sensitive_data, access_the_internet})`, their current business, `isDoing(Agent, {driving, meeting})`.

**Location Characteristics** are attributes of the location where a goal may need to be satisfied, for example the local time `isTime(Location, {morning, winter, january, friday_evening})`, the levels of noise, `hasNoize(Location, none, low, extreme)`, or the temperature, `hasMinTemperature(Location, {T>30, T<-10})`. Note that mobile locations (e.g. buses, cars) can be treated as stable ones with varying characteristics.

**Object Characteristics** refer to objects of the domain and varying attributes thereof, or simple global facts and parameters that cannot be classified otherwise. Examples of this category are highly domain dependent: `messageSize(Message, {X>50kB, X>1MB})`, `customersInLine(Line, {C>4, C>20})`, `isAtLocation(Printer, {lab, supply room, computer room})` or `totalEnrolments(Course, {C>300, C<20})`.

The relations may refer to either short term circumstances of the agents, locations and objects they involve (e.g. current level of noise somewhere) or long term conditions (e.g. a user’s physical challenge).

4.8 Variability-Intensive Decomposition and Analysis

We now present an example process for variability-intensive reduction of goals and a method for reasoning about the resulting high-variability model. The process is based on the identification of an initial set of variability concerns, followed by a one-concern-at-a-time goal decomposition approach in order to form the AND/OR tree. Meanwhile, domain variables are set as selection conditions for each of the introduced goals allowing reasoning about the role of environmental circumstances in the selection of alternatives. We illustrate the process using the Be Notified example we introduced in Section 2.
Identification of relevant variability concerns and domain variables. Before starting the decomposition of the goal Be Notified the analyst will consider variability concerns that are relevant to that goal. Further, for each relevant variability concern, an initial domain of options is identified. In the example (we are using the general set of concerns): an agentive concern is relevant as to who will generate and send the notification ({human observer, a system}\_A), a dative concern poses the question of who is notified, a process concern calls for alternative notification modes ({open-audio, headphone-audio, vibration}\_P\_ins) and manners ({intensive, normal, subtle}\_P\_man), a locational concern asks where the notification is send ({nursing station, meeting room, room1, ...}\_L), a factitive concern questions the content of the notification ({distinctive sound, a voice message explaining situation}\_F), a conditional-trigger concern deals with alternative options on the condition that needs to be true for the notification to be fired ({trying to get up, sitting on bed}\_C\_tri). This collection of concerns constitutes the variability frame of the goal.

At the same time, interesting relations for representing domain variability are considered. As discussed earlier, these represent characteristics of agents, locations, as well as miscellaneous facts or objects whose characteristics may vary. In our case, agents are different types of nurses in relation to the patient ({The Assigned Nurse, The Closest Nurse}) and the locations are the rooms of the unit ({Patient’s Room, Meeting Room, Nursing Station}). Each nurse can be in a room. The nurse may be busy attending to a patient, doing paperwork, having a break, or she may simply be available. The time can be night, day, or afternoon-nap time (associating time with location is not useful here). Certain other facts may influence the identification/selection of alternatives such as the severity of the patients condition, the degree of belief by which we assume she is trying to get off the bed, the number of nurses available in the unit etc.

Such domain facts are identified with respect to a particular variability concern, otherwise they are irrelevant. Thus, the time of the day (day vs. night) is important for deciding the loudness of a notification, whereas the age of a nurse does not influence such a decision. Figure 4.2 shows some variability concerns associated with the root goal Be Notified as well as the related domain facts.

Concern-driven decomposition. Once an initial set of relevant concerns and the associated domain facts are collected, decomposition of the goal can follow.

Initially, for the root goal, each concern is by default labeled unresolved. When a goal is AND-decomposed, every variability concern relevant to the parent goal is inherited by at least one of the AND-subgoals. Some concerns of the parent goal, however, may be irrelevant for
some sub-goals. Thus, if the Be Notified goal is AND-decomposed into Sense Event and Trigger Notification and Receive Notification the former inherits the agentive (A) and condition-trigger concerns (C.tri), while the latter inherits, among others, the dative (D) and process (P) concerns. All concerns are inherited with their label.

When a goal is OR-decomposed, exactly one variability concern relevant to the goal is addressed, while the rest are automatically inherited. A variability concern is addressed by partitioning its domain and assigning each partition to one of the OR-subgoals. If the partition contains only one domain element, then the concern can be labeled resolved at this subgoal and is not inherited in further decompositions of the goal. If not, the concern is labeled addressed and further inherited by subgoals. Figure 4.3 shows the process in detail. In Figure 4.4, the variability concerns that are relevant to the goal Be Notified appear in a rectangular annotation close to it.

**Multi-faceted OR-Decomposition.** The analyst can organize the order by which variability is addressed in the decomposition model in two ways: vertically or horizontally. The former suggests addressing each concern at a separate level of the decomposition sub-tree. For example, for each response for the agentive concern, a subgoal is introduced, each of which is decomposed with respect to the locational concern, having each of the resulting subgoals decomposed with respect to a process concern, and so on. This way of decomposing the goal results in a rather impractically large goal model, as each leaf tends to represent a unique combination of values of each variability concern.
CONCERN-DRIVEN DECOMPOSITION

1. Let $V(g)$ be the set of variability concerns that are relevant to a goal $g$.
2. Let $\text{res}(c,g)$ be the label of the goal $g$ with respect to variability concern $c$; $\text{res}(c,g)$ takes values from the set \{ RESOLVED, ADDRESSED, UNRESOLVED \}.
3. Let $\text{dom}(c,g)$ be the domain of alternative values the concern $c$ can take in goal $g$ and $|\text{dom}(c,g)|$ its cardinality.

1. Let $g_p$ be the goal to be decomposed.
2. If $g_p$ is a root goal, set $\text{res}(c,g_p) := \text{UNRESOLVED}$ for all $c$ in $V(g_p)$.
3. Decompose the goal into its subgoals $g_1, \ldots, g_n$ as follows:
   - If $g_p$ is AND-decomposed:
     - depending on the decomposition purpose, set $V(g_i)$ to be any subset of $V(g_p)$ – {concerns $c$ such that $\text{res}(c,g_p) = \text{RESOLVED}$} that is relevant to $g_i$. Every concern in $V(g_p)$ must appear in some $V(g_i)$.
     - For all $c$ in $V(g_i)$, set $\text{res}(c,g_i) := \text{res}(c,g_p)$.
   - If $g_p$ is OR-decomposed in response to a variability concern $v$ then:
     - for each $g_i$, set $V(g_i) = V(g_p) – \{c$ such that $\text{res}(c,g_p) = \text{RESOLVED}\}$
     - $\text{dom}(v,g_i)$ is partitioned into $n$ non-empty $\text{dom}(v,g), i = 1 \ldots n$.
     - for each $g_i$, if $|\text{dom}(v,g_i)| = 1$ then set $\text{res}(v,g_i) := \text{RESOLVED}$,
       else $\text{res}(v,g_i) := \text{ADDRESSED}$.
     - for each $g_i$, for every concern $c$ in $V(g_i)$ for which $c \neq v$, set $\text{res}(c,g_i) := \text{res}(c,g_p)$.
4. Run the resolution assessment algorithm.
5. For each subgoal repeat the process from (1) until all variability concerns of the root are resolved.

Figure 4.3: Concern-driven Decomposition

In practice, however, variability concerns are orthogonal with respect to the sets alternatives they introduce. Thus, the set of options for where a notification is to be heard, and the set of options for how loud the notification is heard, do not depend on each other (although a selection in one certainly influences the selection in the other). Horizontal organization of variability in goal models allows variability concerns that demonstrate such orthogonality, to be decomposed in parallel i.e. at the same level of the decomposition tree. This can be achieved by simply AND-decomposing the goal into subgoals for each of which only part of the variability concerns of the parent goal becomes relevant. Such an AND-decomposition deviates from its usual meaning (see for example [32] for an extensive discussion on AND-decomposition types) since it is part of what we would call a multi-faceted OR-decomposition, because it allows the representation of refinements of a goal from multiple points of view. In Figure 4.4, Receive Notification is analyzed through a multi-faceted OR-decomposition. The AND-decomposition is annotated appropriately, to show its special function.

As we will see below, selection dependencies among different facets will very likely exist and are treated by establishing lateral links between sub-goals, via e.g. formulating the selection conditions appropriately.

Variability resolution assessment. At any stage of the decomposition process, we can as-
Figure 4.4: Decomposition for Be Notified

assess which of the concerns that were originally thought as relevant have been actually resolved through the decomposition. This can be done by propagating the concern resolution labels from the leaf level nodes towards the root.

The propagation algorithm can be seen in Figure 4.5. If a goal is either OR- or AND-decomposed, a variability concern related to it is labeled resolved, when the concern is labeled resolved in all the subgoals that inherit it. If there exists at least one subgoal where the concern is labeled unresolved or addressed then the parent goal is appropriately (see Figure 4.5) labeled unresolved or addressed. At the end of the process, each of the variability concerns of the root goal are labeled as resolved, addressed or unresolved. In Figure 4.4, next to each concern, one of the three labels (unresolved, addressed, resolved) is used to describe the resolution assessment at the current (early) stage of the decomposition.

**Domain Facts as Selection Criteria.** Apart from addressing variability concerns, predicates describing domain circumstances are set as selection conditions for alternative OR-
RESOLUTION ASSESSMENT ALGORITHM

INPUT: A concern-driven goal decomposition tree.
OUTPUT: A set of labels indicating the state of variability resolution of the root goal.
1. Consider the set $G$ of all intermediate goals whose children are all leafs.
2. For each such goal $g$ in $G$, let $g_1, \ldots, g_n$ be its children and $V(g)$ the set of variability concerns relevant to $g$.
3. Let $\text{res}(c,g)$ be the label of a goal $g$ with respect to concern $c$, with domain \{RESOLVED, Addressed, UNRESOLVED\}.
4. For each concern $c$ in $V(g)$, update its label as follows:
   - If there exists a $g_i$ such that $\text{res}(c,g_i) =$ UNRESOLVED then $\text{res}(c,g) := \text{UNRESOLVED}$.
   - Else if there exists $g_i$ such that $\text{res}(c,g_i) =$ Addressed then $\text{res}(c,g) := \text{Addressed}$.
   - Else /* i.e. no UNRESOLVED or Addressed labels amongst children */ $\text{res}(c,g) := \text{RESOLVED}$.
5. Prune the leafs of the tree and repeat from (1) until the set $G$ is set to empty.

Figure 4.5: Resolution Assessment

subgoals. The concern that is addressed by the decomposition will indicate the predicate(s) to be used as a selection condition, by consulting the respective dependencies identified in the beginning of the process (Figure 4.2). Using these predicates, the analyst can construct formulae that are set as preconditions for the selection of a sub-goal as part of an alternative solution of the AND/OR subtree.

In practice, we can construct useful selection conditions as propositional clauses in which domain predicates are written as propositions. Such clauses are then set as conditions, exactly as we discussed in Section 2. For example, consider the case of a decomposition where the $Pins$ concern is addressed for the goal Notify Designated Nurse and one of the OR-subgoals for the concern is Receive Open Audio Notification, i.e. an audio signal through some sort of a speaker. Whether this goal can be selected as part of an alternative, depends on the response to the locational concern $L$, as well as the time of the day. In our case, it is OK to send an audio notification through the room speaker, provided that this room is not the patients’ ward at the time when everybody is sleeping there (i.e. at night). More formally, this would be: $\neg \text{’Wards’} \lor \neg \text{isNight}$, where ‘Wards’ and isNight represent a goal in the goal model (see Figure 4.4) and a domain fact, respectively. In Figure 4.6, selection conditions have been added to a few of the OR-subgoals.

Reasoning About Alternatives. Time-independent reasoning about the resulting goal models and the domain circumstances, is essentially a satisfiability (SAT) problem for the corresponding propositional formula. The goal of alternatives’ analysis is the identification of circumstances under which no alternative is appropriate for the fulfillment of a goal. In such
cases the analyst needs to introduce new alternatives that are not constrained by such circumstances. In Figure 4.6, for example, the existence of the subgoal ‘Headphone’ is the result of the requirement to have a notification sent even under circumstances in which ‘Wards’ ∧ isNight is true.

We used Prolog to construct a tool that reads an AND/OR structure with \(\text{pre}\) constraints as well as a set of facts that describe domain circumstances and outputs alternatives that satisfy the constraints. The tool goes through all alternatives of the tree and tests their applicability in the given conditions. Thus, the analyst can collect all domain predicates that have been used in conditions, consider truth assignments for them that reflect realistic circumstances and run the procedure to see which alternatives can be considered. The satisfying alternatives can be further subjected to qualitative analysis and prioritization in the spirit of [49, 62]. Although our reasoning tool for time-independent analysis of alternatives is rather naive in terms of computational efficiency, it preformed reasonably well with our goal models. Specifically, it can test one million of alternatives in about 40 sec on a PC with a 2GHz P-4 CPU and 768Mb RAM; our models demonstrated a maximum of 4 million alternatives. Recent advances in SAT and #SAT-solving (e.g. [36]) give us confidence that most practical goal models can be analyzed without any performance issues.

![Figure 4.6: Selection Conditions](image)

In the next Chapter, we will significantly extend our modeling and reasoning framework and introduce time-dependent analysis in which temporal ordering of goal fulfillment becomes relevant. In that case, as we will see, the computational aspect becomes far more challenging.
4.9 In Practice

The technique for developing high-variability AND/OR models we presented constitutes an attempt for systematizing identification of goal alternatives by exploiting the semantic categories that are reflected by the linguistic structure of goal descriptions. We assume that any expressible variation of stakeholder intention can be associated with such a semantic category, and, therefore, an OR-decomposition in our goal tree. Based on our own observations and assessment, our application of the techniques we discussed in this chapter on the geriatric assessment unit case study introduced strong evidence in support of the hypotheses we had set in the beginning.

We considered our case study on the geriatric assessment unit as a primary tool for performing a preliminary evaluation of the techniques we described in the previous sections. Thus, based on these techniques and, acting purely as goal analysts, we developed and iteratively revised the goal models for our case study to a point that we thought the resultant models were complete and meaningful given our understanding of the domain. Then we performed qualitative analysis of the result and reflected on the modeling process in order to understand the effect of the techniques. Our evidence from this step is suggestive of the feasibility of our approach.

Firstly, we certainly confirmed that concern-driven decomposition of goals is possible. In our nursing study we were able to develop goal models with 124 distinct goals that made use of 36 different domain predicates. The goal models include 31 distinct goal decompositions of which 10 are of type A, 6 are P, 5 are L, and the remaining of type C, F, E and T. The diversity of the types of OR-decompositions that were identified is indicative of the soundness of the concern categorization and the applicability of the corresponding concern-driven decomposition technique. Furthermore, reflecting on the goal decomposition process it is our sense that following a concern-driven decomposition process, as opposed to using traditional OR-decompositions, allows the discovery of variability aspects that would otherwise remain hidden. For example, in Figure 4.4 we hadn’t thought of the possibility that different nurses can respond to notifications from the same patient, until we had to think of the role of the dative concern in the notification process. Thus, while the hypothesis of discovery of otherwise hidden alternatives requires formal experimental validation, our experience suggests that the identification of alternatives has been at least facilitated thanks to the concern driven decomposition.

Secondly, regarding the role of non-intentional variables, in our nursing case study we used
Chapter 4. Acquiring Goal Variability

36 different domain predicates which reveals the relevance of the concept. Through the use of the tool, different configurations of non-intentional variables restricted the space of alternatives in an intuitive way. There were also limited incidences of the tool indicating no alternative for a particular configuration of non-intentional variables, which triggered the introduction of new alternatives. These occurrences, while they proved the concept of context-induced goal variability (i.e. introduction of alternatives due to non-intentional variables), they remained limited in number. Nevertheless, we conjecture that enhancing the tool in order to support automatic generation of test configurations for non-intentional variables might increase the number of context-induced variability. Currently, this process is manual and assumes that the user can guess configurations of non-intentional variables for which no alternative is admissible. Arguably, interesting cases may not be discovered this way.

In terms of identification of variability concerns, our work on the University domain we discussed in Section 4.6 demonstrates that it is possible to use corpus annotation for constructing interesting variability frames. However, it also opens a set of investigation opportunities particularly on understanding the sensitivity of this annotation process given different corpora and different subjects for performing the corresponding task. Such subjects include automated annotation tools. Furthermore the effectiveness of the domain specific variability frames in the variability-intensive decomposition process, compared to generic ones, needs to be examined and quantified in a more formal empirical evaluation effort.

4.10 Summary

In this chapter we presented a technique for constructing goal decomposition trees, in a way that allows more systematic identification of variability that exists in the problem. The technique is based on constructing a set of variability concerns that need to be addressed through the introduction of the appropriate OR-decompositions in the AND/OR tree. We presented a way to construct such sets of variability concerns for a given goal and domain by annotating domain-specific corpora. In addition, we introduced the notion of domain facts, which are non-intentional factors that influence the introduction and selection of alternatives independent of stakeholder desires and preferences. Translation of the goal models into propositional logic allows us to perform lightweight reasoning about which alternatives should be chosen under certain circumstances.

In the next chapter, we will take a closer look at such reasoning opportunities. Our focus will be the assessment of alternatives that the goal model implies, in terms of criteria that need
to be satisfied in a particular problem instance. For the purpose we will add temporal extensions to the modeling formalism allowing representation of alternative sequences (versus unordered sets) of tasks that can fulfill the root goals. We will also consider soft-goals and their partial satisfaction as a way to assess the quality of each sequence. This way, selection of alternatives will be possible by reference to the desired satisfaction of soft-goals.
Chapter 5

Goal Alternatives and Preference Specification

5.1 Overview

In the previous chapter we proposed a method for acquiring goal variability through concern-driven decomposition of goals. The resulting AND/OR trees imply a great number of alternative ways by which stakeholders may wish to achieve their top level goal. In this Chapter we introduce a method for selecting alternatives of interest by specifying criteria that such alternatives need to meet. We propose an expressive formal language to describe such criteria in a form of preference specifications. We first construct formulae that describe things that stakeholders want to be true while fulfilling their goals. These can be desired degrees of satisfaction of soft-goals or temporal properties of the selected alternatives. We then combine these formulae into priority rankings according to the stakeholder preferences. The preference specification and the generic model become the input of a tool that searches the latter for alternatives that best satisfy the former. This way, stakeholder preferences are used to leverage the space of alternatives that is implied by the goal model.

We organize our presentation as follows. Section 5.2 discusses the motivation behind introducing a preference specification framework with an emphasis to its potential usefulness in software customization. In Section 5.3 we complement the discussion of Chapter 2 on related work done in the areas of Requirements Engineering and Software Product Lines. In Section 5.4 we reintroduce the running example of Chapter 4, though from a different aspect, in order to illustrate our criteria specification approach. In Section 5.5 we introduce the adaptations that we are going to make to the goal modeling language of Chapter 3 in order to add a temporal
dimension to the goal models. Then, in Section 5.6 we present the semantics in Situation Calculus. In Section 5.7, we introduce the preference language that we are using and in Section 5.8 we discuss how we elicit preference specifications and how we use them to select alternatives of interest. In Section 5.10 we reflect on our case studies and applications, in Section 5.9 we describe the tool for reasoning about preferences and we conclude in Section 5.11.

5.2 Fitness Criteria for Goal Models

As it became apparent in the previous chapters, goal decomposition models imply many alternatives for goal fulfillment, because at the time in which we develop the models certain details, conditions and circumstances about the particular stakeholder instance and her context remain unknown. Recall for example the goal Invite Participants to a Meeting, which can be fulfilled in a number of different ways depending on the circumstances surrounding the invitation. For example, if the invitee is hearing impaired, certain instruments through which we can invite her, such as telephone or voice mail, must be ruled out. The same is true if the meeting initiator prefers to maintain a certain level of formality, in which case the invitation will be a written one. At the time we develop the model, however, we want it to apply to a large number of such circumstances and personal preferences and therefore try to include in our model as many alternatives as possible. The systematic discovery process we presented in Chapter 4 reflects exactly this intention. The generic model that results from this effort satisfies (i.e. partially describes) a large number of problem variants, and we therefore call it generic.

Full description of the problem requires us to complement the generic model with the details that pertain to particular stakeholder instances that try to fulfill the goal under given circumstances. Once these details are known, certain alternatives are ruled out while others become more relevant. Hence, this additional information describes fitness criteria that need to be satisfied in a given problem instance ([78]). Given the fitness criteria, alternatives are evaluated and selected subject to how well they satisfy the criteria.

This generic model vs. fitness criteria concept constitutes a requirements modeling paradigm particularly suitable for customizable software and product lines. Such software is designed in two stages. Firstly, by exploiting the result of the domain analysis process, reusable core assets are developed centrally, and destined to be adapted to as many problem instances as possible. Secondly, at a local level, the core assets are customized and adapted with respect to the understanding of the problem instance that needs to be tackled. These are two distinct processes and should be served by distinct requirements modeling exercises. In our view, a generic re-
quirements model describes the family of problems that need to be tackled by the customizable software asset, while the fitness criteria further complement the problem description in a way that certain alternatives implied by the generic model become more relevant. So far we have discussed the use of our high-variability goal models as generic requirements models. We now take a look at how fitness criteria for goal alternatives can be represented.

In our framework, fitness criteria are interpreted as preferences of stakeholders over high-level qualitative and behavioral characteristics of desired goal alternatives together with non-intentional facts that characterize the environment. In this chapter, we introduce a formal language which allows analysts to specify such preferences. To do so we introduce an extension of the basic goal modeling language we presented in Chapter 3. This extension allows modeling of temporal characteristics of goal fulfillment, through allowing precedence constraints among goal fulfillment. The preference specification language allows prioritization over behavioral properties (formulated in temporal logic) and overall quality that candidate alternatives must exhibit. Then, an inference engine, based on a combination of preference-oriented ([17]) and hierarchical task network planning (HTN) ([89]), searches for behaviors that best fit to a given preference specification.

In the next section we will complement the critique made in Chapter 2 on current research and practice for specifying instantiation/derivation criteria for generic software or generic models.

### 5.3 Related Work

The notions of preference and priority are not new in requirements engineering. On the contrary, the notion of requirements prioritization originates exactly from the observation that not all requirements have the same importance for all stakeholders. An elementary requirements prioritization approach, for example, is to divide requirements into “must-haves” and “nice-to-haves”, whereby the former are understood as more important, urgent or otherwise of higher priority (e.g. [12]). In addition to this common qualitative approach, more elaborate quantitative prioritization techniques, such as the Analytic Hierarchy Process ([73, 6]) or multi-criteria preference analysis methods ([63]) have also been proposed and successfully used in practice. The use of multi-attribute decision theoretic approaches has also been explored, e.g. in [75].

The modeling and reasoning side of prioritization, however, has not received as much attention in requirements engineering. Instead, researchers have mostly been focusing on modeling requirements variability (e.g. [53, 38, 44, 111]), without including in their scope the problem
of selecting requirements variants according to given stakeholder priorities. Note that “requirements” here refer to features of the system-to-be. For example, in [69], [53] and [67] variability in use cases diagrams and their documentation is incorporated either through introducing appropriate diagrammatic constructs, or by writing scripts that customize the details of default templates ([67]). The binding of variation points is performed manually on a one-by-one basis. As an interesting exception, a scenario generation approach from generic use-cases, proposed in [110], introduces a constraint language to be used for selecting scenario instances. However, that language does not appear to be capable of describing preferences over temporal properties of the expressive power we will consider here and it is generally geared towards solution-oriented use-cases rather than stakeholder goals. The same comment can be made for [34], where a graphical model for representing dependencies amongst preferences, the CP-net, is used for the customization of web pages. Moreover, although the notion of preference is present there, behavioral characteristics of software are not considered.

In product lines, the term product derivation is used to describe the process of instantiating/extending a set of core software assets to construct a particular product instance. Many domain analysis methods include decision models to be used for product derivation (e.g. [11]). Such models are typically based on variation points and hierarchical dependency graphs, such as feature models which we discussed in Chapter 2. The role of feature models is to represent the configurability aspect of the software system, and for this reason they need to have relaxed semantics so that they can cover a large variety of concepts in a single model. Thus, features can be behaviors, functions, structures, or even qualities. This unavoidable generality of features, however, implies that it is very difficult to semantically interpret them in terms of, for example, either behavior or structure of software. This, in turn, reduces the opportunities for using them to perform useful reasoning tasks, other than, for instance, simple combinatorial reasoning. Hence, with plain feature models, derivation of a variant is a process of direct reading and manipulation of the model, as in [30], for example, where a technique for gradual resolution of variation points is proposed. The problem of directly dealing with variation points can be intensified when the feature models are large or when they contain solution-specific terms which are difficult to relate to stakeholder needs, let alone to decide upon them.

Finally, Domain Specific Languages (DSLs - [27, 14]) and Product Configuration technologies ([27, 14]) both introduce ways by which generic assets with high degrees of freedom can be configured at a high-level in order to fit specific constraints. We discussed these in Chapter 2; in summary these proposals are characterized by a focus on the solution rather than the problem, and, in addition, they keep the behavioral aspect of the matter that is being configured
out of their focus.

## 5.4 Motivating Example and Research Approach

Let us now return to the health care example we introduced in Chapter 4, and see how preference specification can facilitate the selection of alternative ways to solve problems. The context of the application was a geriatric assessment unit, where elderly are hospitalized for a period of time. In our example, we analyzed the case where a patient needs to be attended to by a nurse due to an event, which in our example was that the patient was trying to get up even though she is not allowed to due to her health condition. In fact, there is a variety of events that need to be brought to the attention of the nurse. For example the patient may have remained immobile for an extended period of time (which causes sores), or she may have called for the nurse herself to ask a question or to request additional medicine. In all cases, the nurse needs to be notified somehow, either through a broadcasted notification using the speakers of the unit, or through earphones he wears while on duty. Then, the nurse’s reaction needs to be determined. Normally, he has to visit the patient’s room, but if the patient only wants to ask a question or request permission for something, the visit may be replaced by establishing a voice link between patient and nurse. For example, the nurse may be carrying a mobile set with microphone and earphones, or there may be a device at the nursing station which is conveniently located in the unit. The nurses think that this would increase unnecessary disturbance from some patients, but they acknowledge it would also increase their productivity, and save them from extra walking effort.

All these are alternative behavioral designs that need to be evaluated subject to criteria posed by individual stakeholders and context instances. Different geriatric assessment units, or the same unit in different times and situations, may have different priorities over high-level characteristics of the desired solution. For example, in a particular unit the nurses may state that “[they] don’t like the idea of talking to the patient remotely, but if they had to, they would choose to do so at the nursing station.”. The managers of the unit, on the other hand, will use a more high-level language: “we should definitely avoid anything that would make the patient unhappy, but it would also be nice to increase nurses’ productivity somehow.”. How can we translate these statements into a selection of behavioral designs that best satisfy them?

As in the previous chapter, we base our research approach on our study on the nursing domain; the example is extracted and adapted from that study. Again, the organization we investigate is a Geriatric Assessment Unit of the Queen Elizabeth II Health Sciences Centre in
Halifax, Canada. In that unit a number of elderly patients exhibiting a health issue are hospitalized for a limited amount of time until their condition is assessed. We gathered information about the structure and processes of the unit through a series of interviews and observations. Having this information at hand, we formed two hypotheses. Firstly, we would be able to construct temporally extended goal models that, apart from goal structure, show how goals can be ordered in time. Secondly we would be able to form preference formulae that would involve both temporal properties and qualitative impact of desired alternatives. Potentially our preference language would also show potential of merging preferences of different stakeholders. In this exploration, the “ability” of constructing models of goals and preferences, is measured by our intuition (as analysts) on the degree by which the resulting models are sensible and valid given the domain information.

Our modeling techniques were further tried in more examples, this time based on artificial cases, again for the purpose of testing the feasibility and intuitiveness of the result. We discuss the results of this exploration towards the end of the chapter.

5.5 The Goal Modeling Language

In this chapter, we extend the goal modeling language we introduce in Chapter 3 in order to allow representation of the temporal dimension of goal fulfillment. For the purpose we will introduce two special types of links, that help us define allowable orderings of task performance and goal fulfillment. We will also introduce a special type of element that describes conditions under which performance of tasks is possible.

More specifically, our goal model consists of the usual elements:

1. a set of hard-goals $H$,
2. a set of soft-goals $L$,
3. a set of tasks $T$,
4. a set of domain concepts $O$.
5. a set of domain facts $R$, which are relations over domain concepts.

We discussed hard-goals, soft-goals and tasks in Chapter 3. Domain facts express ways by which domain concepts, such as nurse, nursingStation, english, printer, relate to each other at a particular time instance and while actors are performing tasks to fulfill their goals. Examples of domain facts are $\text{isAt(nurse, nursingStation)}$, $\text{isAvailable(nursingStation, printer)}$ and $\text{speaks(patient, english)}$. 
The truth value of domain facts may or may not change due to the performance of tasks. Notice that relational domain facts replace the propositional ones we introduced in Chapter 4.

Using the domain facts together with 0-ary predicates that describe tasks and goals we can construct simple first-order formulae, which we will call condition formulae.

**Definition 4.1 (Condition Formula - CF)** A condition formula $\phi$ is drawn from a set $K$ for which:

1. $H \subset K$, $T \subset K$ and $R \subset K$
2. if $\phi, \phi_1, \phi_2 \in K$ then so do: $\neg \phi$, $\phi_1 \land \phi_2$, $\phi_1 \lor \phi_2$.

A CF is understood in the context of a course of activities that aims at fulfilling a root goal. Predicates that represent tasks (respectively goals/domain facts) are true if and only if the respective task (goal/fact) has been performed (is satisfied/is true) at a given time instance, while the actor is active in order to fulfill the root goal. Thus, $\text{Nurse Notified} \land \text{isAt(nurse, nursingStation)}$ is true if the goal Nurse Notified has been satisfied and the nurse is at the nursing station.

Figure 5.1 shows how the above are represented diagrammatically. In addition to the usual oval, cloud-shaped and hexagonal elements representing goals, soft-goals and tasks, there are two new types of rectangle-shaped elements: the Condition Element (CE), a version of which was also used in Chapter 4, and the Effect Element (EE), each containing a CF and a sole domain fact, respectively.

In the figure, annotations have been added to distinguish between different types of elements, although these are easily distinguished by the type of links by which they connect to the rest of the graph, as it will become apparent below. To ease our presentation we will use $\phi_c$ and $p_e(\vec{o})$ to denote the CF and the domain fact contained in CE $c$, EE $e$, respectively.

Hard-goals and tasks form the hard-goal subgraph which now includes all EEs and some of the CEs. Soft-goals and the rest of the CEs form the (directed and acyclic) soft-goal subgraph. The two subgraphs are connected through contribution links that originate from tasks and goals of the hard-goal graph and target soft-goals of the soft-goal graph. These tasks and goals are considered to be parts of both graphs.

As we saw in Chapter 3 the two sub-graphs have distinct functions in our framework. The hard-goal graph allows us to represent alternative ways by which a root hard-goal can be satisfied (e.g. different ways to have the Nurse Notified), whereas the soft-goal graph allows us to assess how each alternative affects high-level quality goals of the stakeholders (e.g. how different ways to have the Nurse Notified affect the soft-goal Patient’s Privacy).
5.5.1 The hard-goal subgraph

We now present our adapted hard-goal graph in more detail. Its backbone is an AND/OR decomposition tree which consists exclusively of hard-goals and tasks. Leaf level nodes are only tasks, and tasks can only be leaf level nodes of the hard-goal graph. The decomposition tree contains AND- or OR-decompositions. When a goal $g$ is AND-decomposed into goals or tasks $g_1, \ldots, g_n$ then $g$ is satisfied iff $g_i$ are satisfied (performed in the case of tasks) for all $i$. If $g$ is OR-decomposed, then $g$ is satisfied iff there exists an $i$ such that $g_i$ is satisfied (performed if it is a task). Thus, any AND/OR decomposition tree rooted at $g_r$ implies a set of subsets of $T$ that are capable of satisfying the root goal $g_r$. We called these alternatives for $g_r$.

Two additional types of links are associated with the hard-goals graph. The first one is the precedence constraint link that is applied in three ways:

**At the leaf level**, $c \xrightarrow{\text{pre}} t$, where $c$ a CE and $t$ a task, means that $t$ can be preformed only if $\phi_c$ is true at the time when performance of $t$ is attempted.

**Between AND-subgoals**, $g_1 \xrightarrow{\text{pre}} g_2$, means that no task that is part of $g_2$’s subtree can be performed unless a set of tasks that constitutes an alternative for $g_1$ has already been performed.

**On effect elements**, $c \xrightarrow{\text{pre}} e$, where $c$ is a CE and $e$ an EE, means that any effect link (see below) targeting $e$ can be effective only if $\phi_c$ is true at the time the effect is considered.
The second type of links, the effect links $t \xrightarrow{ef} e$ and, respectively $t \xrightarrow{nef} e$, are applied from a task $t$ to an EE $e$ and imply that completion of performance of the former instantly causes the fact $p_e(\vec{d})$ contained in the latter to become true (respectively, false), unless there exists a $c \xrightarrow{pre} e$, where $c$ is a CE whose condition formula $\phi_c$ is false. In that case no changes are made.

The precedence and effect links add a temporal dimension to our goal graphs, in a way that we are not only interested in sets of tasks to satisfy the root goals but also how their execution is ordered. What will follow in Section 5.6 is the provision of semantics for the temporally extended goal models via translating them to a formal language for specifying dynamic domains.

### 5.5.2 The Soft-goal subgraph

Soft-goals form the soft-goal subgraph through the use of weighted contribution links. As described in Chapter 3, every soft-goal $l$ is associated with variables $valS(l)$ and $valD(l)$ each indicating the degree by which it is believed that the soft-goal is satisfied or denied. Goals, tasks and CEs for which there is a contribution link to a soft-goal are also parts of the soft-goal subgraph, and hence each is associated with a $valS(\cdot)$ variable (but with no $valD(\cdot)$ variable).

The contribution values, the types of contribution, depending on whether we are using the qualitative or quantitative modeling framework, as well as the algorithm for calculating the propagation of satisfaction and denial through the graph are given in Chapter 3; we refer the reader to that chapter for more details.

Note that in light of adding a temporal dimension to our goal models, soft-goal satisfaction and denial values are subject to fluctuations while a sequence of tasks is considered in the hard-goals graph. Thus, we are not only interested in the soft-goal satisfaction that results from considering a complete sequence of tasks that fulfills the root goal, as we have done so far, but also in how the degrees change while the sequence is build from the first task until the last.

### 5.6 Goal Semantics in Situation Calculus

We now provide the semantics for the combined goal tree, which includes both the hard-goal subgraph where a family of sequences of possible leaf level tasks is modeled and the hard-goal subgraph which models how the performance of tasks influences the satisfaction of soft-goals. We appeal to the situation calculus to define the semantics of our goal language, which enables us to easily exploit existing algorithms and tools for preference-based planning for the purpose
of evaluating goal-level preferences.

5.6.1 Situation Calculus

The situation calculus is a logical language for specifying and reasoning about dynamical systems [99]. In the situation calculus, the state of the world is expressed in terms of functions and relations (fluents) relativized to a particular situation, e.g., \( f(\vec{x}, s) \). A situation \( s \) is a history of the primitive actions, \( \alpha \in \mathcal{A} \), performed from a distinguished initial situation \( S_0 \).

The function \( do(\alpha, s) \) maps a situation and an action into a new situation thus inducing a tree of situations rooted in \( S_0 \). The predicate \( \text{Poss}(\alpha, s) \) is true if action \( \alpha \) is possible in situation \( s \).

A basic action theory comprises the domain-independent foundational axioms of the situation calculus, successor state axioms, precondition axioms, axioms describing the initial state of the system, unique names axioms for actions and domain closure axioms for actions. \( D \) may also include some state constraints, such as ramification axioms or definitional axioms for fluents. Given a goal formula \( G \), a plan in the situation calculus is a sequence of actions \( \vec{\alpha} = \alpha_1, \alpha_2, \ldots, \alpha_n \) such that for the situation \( s = do(\alpha_n, \ldots, do(\alpha_1, S_0)) \), \( G \) holds in \( s \) and the precondition axioms are satisfied throughout \( \vec{\alpha} \).

The details of \( D \) are described in [99]. In the section that follows, we show how to translate our goal model into a basic action theory, \( D \).

5.6.2 Translating the Goal Model

We define the semantics of our visual goal language via a set of translation rules. Similar translation proposals are introduced in [48] and [121], but for different purposes; a distinguishing feature of our approach is the consideration of soft-goals as part of the translation. We first establish a mapping from the primitives of the goal based graphical language to those of the situation calculus:

**Primitives**

- For every task \( t \) that appears in the goal model introduce an action \( \alpha_t \) and a relational fluent \( \text{performed}(t, s) \) in the situation calculus domain theory.

- For every goal \( g \) introduce an AND/OR formula \( \varphi_g(s) \) of predicates of the type \( \text{performed}(t, s) \). The formula is constructed as follows. Starting from \( g \), each goal is recursively replaced by the conjunction or disjunction of its children, depending on whether \( g \) is...
AND or OR decomposed. If these subgoals are tasks, then the predicate \( performed(t, s) \) is used and the recursion terminates.

- For every domain fact \( p(\vec{o}) \) introduce a relational fluent \( f_p(\vec{x}, s) \), where \( \vec{x} \) are individuals representing domain concepts \( \vec{o} \).

- Use individuals (constants) \( r_t, r_g, r_l \) and \( r_c \) to identify a task \( t \), a goal \( g \), a soft-goal \( l \) and a CE \( c \) that are part of the soft-goals graph. Also, let \( P_T, P_H, P_L \) and \( P_C \) respectively be the set of all such individuals and \( P \) their union. Then define fluents \( v_s(r, s, w) \) and \( v_d(r, s, w) \), where \( r \) is an individual in \( P \), i.e. represents a node in the soft-goal graph. Thus, these fluents represent respectively the satisfaction and denial degree \( w \) of soft-goal graph node \( r \) in situation \( s \). Obviously, the domain of \( w \) depends on the framework of use. Thus:

<table>
<thead>
<tr>
<th>Quantitative:</th>
<th>[0,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualitative:</td>
<td>{N,P,F}</td>
</tr>
</tbody>
</table>

Notice that CE, tasks and hard-goals also have a satisfaction value, albeit with a restriction, as we will see later.

- Introduce the fluent \( \text{link}(r_1, r_2, y, w) \), where \( r_1 \) is in \( P \), \( r_2 \) is in \( P_L \) and \( y \in \{"S+","S-","D+","D-"\} \). The fluent represents the weight \( w \) of the contribution link originating from \( r_1 \) targeting \( r_2 \), while \( y \) denotes the type of the link. Again the domain of \( w \) depends on the framework we are using:

<table>
<thead>
<tr>
<th>Quantitative:</th>
<th>(0,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualitative:</td>
<td>{some, full}</td>
</tr>
</tbody>
</table>

For example, the contribution link \( l_1 \xrightarrow{0.3D+} l_2 \) produces \( \text{link}(r_{l_1}, r_{l_2}, "D+", 0.3) \). On the other hand, the contribution link \( l_1 \xrightarrow{+\!+\!+} l_2 \) produces \( \text{link}(r_{l_1}, r_{l_2}, "D+", \text{full}) \) and \( l_1 \xrightarrow{\text{some}} l_2 \) gives \( \text{link}(r_{l_1}, r_{l_2}, "S-", \text{some}) \).

- For every CF \( \varphi \) appearing in a CE, produce its translation \( \varphi \) into the situation calculus ontology by translating each task predicate \( t \), goal \( g \) and domain fact \( p(\vec{o}) \) mentioned in the CF to the corresponding fluent \( performed(t, s) \), formula \( \varphi_g(s) \), and fluent \( f_p(\vec{x}, s) \).
Successor State Axioms

We can now construct the successor state, precondition and initial situation axioms based on the following rules. Note that $\supset$ denotes the implication connective.

- For every effect link $t \xrightarrow{eff} e$, introduce a successor state axiom of the type:

$$\text{Poss}(\alpha, s) \land (\alpha = \alpha_t) \land \bigwedge_{\forall \phi_c \in C^\text{pre}} \varphi_c(s) \supset f_{p_e}(\vec{x}, \text{do}(\alpha, s)) \quad (1)$$

Dually, for every negative effect link $t \xrightarrow{nef} e$ introduce a successor state axiom of the type:

$$\text{Poss}(\alpha, s) \land (\alpha = \alpha_t) \land \bigwedge_{\forall \phi_c \in C^\text{pre}} \varphi_c(s) \supset \neg f_{p_e}(\vec{x}, \text{do}(\alpha, s)) \quad (2)$$

In both axioms, $C^\text{pre}$ is the set of CFs $\phi_c$ contained in each CE $c$ for which there is a $c \xrightarrow{\text{pre}} e$. Also $f_{p_e}(\vec{x}, s)$ is the situation calculus formula that results from the translation of $p_e(\vec{\phi})$ which in turn is a sole domain fact in EE $e$.

Intuitively, the axioms ensure that relationships appearing in effect elements will be enabled (or disabled accordingly) when any of the tasks that points to these elements is performed, provided that the appropriate conditions are satisfied at that time.

- For each of the fluents $\text{performed}(t, s)$ introduced above, construct a successor state axiom as follows:

$$\text{Poss}(\alpha, s) \land (\alpha = \alpha_t) \supset \text{performed}(t, s) \quad (3)$$

Thus, the fluent $\text{performed}(t, s)$ will become true once the action associated with the task $t$ is performed.

Ramification Axioms

Ramification axioms describe consequences of direct effects. In our context, the existence of indirect effects in situation calculus reflects the effect of the performance of low level tasks to the satisfaction and denial of soft-goals, which may, in turn, influence the satisfaction or denial of other soft-goals. Thus, the axioms are written in accordance to the structure of the soft-goals graph, in a way that performance of the adapted label propagation algorithm we introduced in Chapter 3 is ensured. Thus:
• For every individual \( r_t \in P_T \), \( r_g \in P_G \) and \( r_c \in P_C \) introduce a pair of axioms that associates the value of \( v_a(r_t, s, w) \), \( v_a(r_g, s, w) \), and \( v_a(r_c, s, w) \) with formulae grounded on fluents of type \( \text{performed}(\cdot, s) \). Depending on whether we are working with the quantitative or qualitative framework we respectively have:

\[
\begin{array}{c|c}
\text{Quantitative} & \text{Qualitative} \\
\text{performed}(t, s) \supset v_a(r_t, s, 1) & \text{performed}(t, s) \supset v_a(r_t, s, F) \\
\neg\text{performed}(t, s) \supset v_a(r_t, s, 0) & \neg\text{performed}(t, s) \supset v_a(r_t, s, N) \\
\varphi_g(s) \supset v_a(r_g, s, 1) & \varphi_g(s) \supset v_a(r_g, s, F) \\
\neg\varphi_g(s) \supset v_a(r_g, s, 0) & \neg\varphi_g(s) \supset v_a(r_g, s, N) \\
\varphi_c(s) \supset v_a(r_c, s, 1) & \varphi_c(s) \supset v_a(r_c, s, F) \\
\neg\varphi_c(s) \supset v_a(r_c, s, 0) & \neg\varphi_c(s) \supset v_a(r_c, s, N)
\end{array}
\]

(4) (5) (6) (7) (8) (9)

Where \( \varphi_c \) is the situation calculus translation of the formula in CE \( c \). The above formulae set the satisfaction degree of soft-goal graph nodes which are tasks, goals or CEs. Observe that we prevent partial satisfaction to such nodes.

• For every soft-goal \( l \) in the goal model, let \( R_{S_+} \) and \( R_{D_+} \) be the sets of soft-goal graph nodes \( k_i \in R_{S_+} \) and \( m_j \in R_{D_+} \) for which \( k_i \xrightarrow{w_{S_+}} l \) and \( m_j \xrightarrow{w_{D_+}} l \), respectively, where \( w \) is the respective weight.

Let \( z_{S_+} \) be an abbreviation for \( z_{S_1}^{k_1}, z_{S_1}^{k_2}, \ldots \) for \( k_1, k_2, \ldots, k_i, \ldots \in R_{S_+} \). Similarly, \( z_{D_+} \) is an abbreviation for \( z_{D_1}^{m_1}, z_{D_1}^{m_2}, \ldots \) for \( m_1, m_2, \ldots, m_j, \ldots \in R_{D_+} \). Then construct the successor state axiom:

\[
\begin{align*}
\{ & \bigwedge_{k_i \in R_{S_+}} \text{link}(r_{k_i}, r_t, \text{“S+”}, w_{k_i}) \land v_a(r_{k_i}, s, y_{k_i}) \land \text{rule}(z_{S_+}^{k_i}, w_{k_i}, y_{k_i}) \\
\land & \bigwedge_{m_j \in R_{D_+}} \text{link}(r_{m_j}, r_t, \text{“D+”}, w_{m_j}) \land v_d(r_{m_j}, s, y_{m_j}) \land \text{rule}(z_{D_+}^{m_j}, w_{m_j}, y_{m_j}) \\
\land & \max(z_{\text{max}}, z_{S_+}, z_{D_+}) \supset v_a(r_t, s, z_{\text{max}}) \}
\end{align*}
\]

Dually, for every soft-goal \( l \) in the goal model, let \( R_{S_-} \) and \( R_{D_-} \) be the sets of soft-goal graph nodes \( k_i \) and \( m_j \) for which \( k_i \xrightarrow{w_{S_-}} l \) and \( m_j \xrightarrow{w_{D_-}} l \), respectively, where \( w \) is the respective weight. Let \( z_{S_-} \) be an abbreviation for \( z_{S_1}^{k_1}, z_{S_1}^{k_2}, \ldots \) for \( k_1, k_2, \ldots, k_i, \ldots \in R_{S_-} \). Similarly \( z_{D_-} \) is an abbreviation for \( z_{D_1}^{m_1}, z_{D_1}^{m_2}, \ldots \) for \( m_1, m_2, \ldots, m_j, \ldots \in R_{D_-} \). Then construct the successor state axiom:

\[
\begin{align*}
\{ & \bigwedge_{k_i \in R_{S_-}} \text{link}(r_{k_i}, r_t, \text{“S-”}, w_{k_i}) \land v_a(r_{k_i}, s, y_{k_i}) \land \text{rule}(z_{S_-}^{k_i}, w_{k_i}, y_{k_i}) \\
\land & \bigwedge_{m_j \in R_{D_-}} \text{link}(r_{m_j}, r_t, \text{“D-”}, w_{m_j}) \land v_d(r_{m_j}, s, y_{m_j}) \land \text{rule}(z_{D_-}^{m_j}, w_{m_j}, y_{m_j}) \\
\land & \max(z_{\text{max}}, z_{S_-}, z_{D_-}) \supset v_d(r_t, s, z_{\text{max}}) \}
\end{align*}
\]

(10)
Also, \( \max(y, x_1, x_2, \ldots, x_n) \) holds iff \( y \) equals the maximum of \( x_1, x_2, \ldots, x_n \). Furthermore, \( \text{rule}(z, w, y) \) is defined as follows depending on which framework we are considering:

**Quantitative:** \( \text{rule}(z, w, y) \equiv (z = w \cdot y) \)

**Qualitative:** The definition of \( \text{rule}(z, w, y) \) is based on the following table:

<table>
<thead>
<tr>
<th>( w )</th>
<th>( z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>some (+/-)</td>
<td>( \min(y, P) )</td>
</tr>
<tr>
<td>full (+/--)</td>
<td>( y )</td>
</tr>
</tbody>
</table>

These axioms ensure that the satisfaction labels of the sources of the soft-goals graph are propagated according to the propagation rules we introduced earlier. Note, however, that the syntactic features of ramification axioms can cause an issue called the *ramification problem*, whereby unintended models of the ramification axioms are satisfied. We will discuss this problem below and show that the structure of our action theory makes it amenable to a syntactic manipulation that can lift the ramification problem.

**Action Precondition Axioms**

- For every task \( t \) in the goal model construct a precondition axiom as follows. First construct formula \( \varphi_{comp} \) as follows. Consider the path from \( t \) to the root goal. Let \( G_{OR} \) be the set of all nodes \( g_{OR} \) in the path which are OR-decomposed, including the root and \( t \)’s parent. For each such \( g_{OR} \) consider its children that do not belong to the path from \( t \) to the root goal. Let \( G_{comp} \) be the set of all such children of all \( g_{OR} \in G_{OR} \). Finally let \( T_g \) be the set of all leaf level tasks that are successors of a goal \( g \). If \( g \) is a task then \( T_g \) contains only that task. The formula \( \varphi_{comp} \) is constructed as follows:

\[
\varphi_{comp} \equiv \bigwedge_{g \in G_{comp}} \bigvee_{t \in T_g} \text{isPerformed}(t)
\]

Observe that \( t \) does not occur in any alternative together with any of the tasks in \( T_g, \forall g \in G_{comp} \). Excluding consideration of \( t \) together with any of these tasks ensures that the plans are minimal with respect the goal tree, or, in other words, no subset of the tasks included in the plan satisfies the root goal. Thus, \( \varphi_{comp} \) will be true if some of the competing tasks has already been performed making \( t \) redundant.
Then consider the set $G_{\text{pre}}$ of all hard-goals $g_i$ such that $g_i \xrightarrow{\text{pre}} g_j$, where $g_j$ is any ancestor of $t$ in the hard-goals subgraph. The precondition axiom for $t$ is the following:

$$\text{Poss}(\alpha_t, s) \equiv ( \bigwedge_{\varphi_i \in G_{\text{pre}}} \varphi_i(s)) \land ( \bigwedge_{\varphi(s) \in \Phi} \varphi(s)) \land (\neg \varphi_{\text{comp}}) \quad (12)$$

where $\Phi$ is the set of condition formulae $\varphi_c$, for which $c \xrightarrow{\text{pre}} t$.

**Initial Situation**

For the initial situation $D_{S_0}$, every predicate of type $\text{performed}(\cdot, S_0)$ is set to false and every fluent of type $f_p(x, S_0)$ is set according to information given in the domain about $p(o)$. Moreover, every fluent of type $v_a(r, s, y)$ and $v_d(r, s, y)$ is initialized depending on the framework of consideration:

<table>
<thead>
<tr>
<th>Quantitative:</th>
<th>$v_a(r, S_0, 0)$, $v_d(r, S_0, 0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualitative:</td>
<td>$v_a(r, S_0, N)$, $v_d(r, S_0, N)$</td>
</tr>
</tbody>
</table>

**Plans and Behaviors**

If $D$ is the action theory derived from the goal model and $\varphi_g$ the formula representing the root goal $g$, then we will use the term behavior to refer to a plan for $D$ that achieves $\varphi_g$.

5.6.3 **On the Ramification Problem**

The ramification problem arises from the syntactic characteristics of ramification axioms when compared to their intended meaning. Consider the simple example of Figure 5.2. The ramification axioms associated with the satisfaction values of goal $l_1$ are:

$$\text{link}(r_{t_2}, r_{l_1}, \text{"S+"}, 0.6) \land v_a(r_{t_2}, s, y_{t_2}) \land \text{rule}(z_{t_2}, 0.6, y_{t_2})$$

$$\land \text{max}(z_{\text{max}}, z_{t_2}) \supset v_a(r_{l_1}, s, z_{\text{max}})$$

For goal $l_2$ the corresponding axiom is the following:

$$\text{link}(r_{t_1}, r_{l_2}, \text{"S+"}, 0.5) \land v_a(r_{t_1}, s, y_{t_1}) \land \text{rule}(z_{t_1}, 0.5, y_{t_1})$$

$$\land \text{link}(r_{l_1}, r_{l_2}, \text{"S+"}, 0.9) \land v_a(r_{l_1}, s, y_{l_1}) \land \text{rule}(z_{l_1}, 0.9, y_{l_1})$$

$$\land \text{max}(z_{\text{max}}, z_{l_1}, z_{l_1}) \supset v_a(r_{l_2}, s, z_{\text{max}})$$

Assume now that $v_a(r_{t_1}, s, 1)$ and $v_a(r_{t_2}, s, 1)$, due to the performance of tasks $t_1$ and $t_2$. The obvious indirect effect to soft-goal satisfaction would be $v_a(r_{l_1}, s, 0.6)$ and $v_a(r_{l_2}, s, 0.54)$. 
But the way our ramification axioms are expressed may as well imply that, for example, 
\( v_s(r_{i_1}, s, 0.6) \), \( \neg v_s(r_{i_1}, s, 0.7) \) and \( \neg v_s(r_{i_2}, s, 0.56) \). Arguably the latter is not very useful for understanding the satisfaction value of \( l_2 \). Hence, we need to find a way to prevent our system of axioms from being satisfied by models (truth assignments) which do not completely calculate the satisfaction and denial values of all goals. For this to be true, the implication connective (\( \supset \)) of the ramification axioms needs to be treated as *definitional* [85], in a sense that the right-hand side of the implication connective is understood as defined in terms and only in terms of the left-hand side.

It has been shown in [85] that if the action theory in situation calculus is a *solitary stratified theory* then it can be re-written in a form that does not imply such unwanted models. A solitary stratified theory has the following characteristics:

**Definition 5.6.1** Suppose \( D \) is a theory in the language of the situation calculus with domain fluents, \( L \). Then \( D \) is a solitary stratified theory with stratification \( (D_1, D_2, \ldots, D_n) \) and partition \( L_1, L_2, \ldots, L_n \) if

- for \( i = 1, \ldots, n \), \( L_i \) is the set of fluents \( F_i \) that are defined in stratum \( D_i \) and \( L_1 \cup L_2 \cup \ldots \cup L_n = L \)

- \( D \) is the union \( D_1 \cup D_2 \cup \ldots \cup D_n \) of sets of axioms \( D_i \) where for each stratum, \( D_i \) is solitary with respect to \( L_i \), that is each \( D_i \) can be written as the union \( (M_i \leq \neg L_i \cup E_i \leq L_i) \), where:

  1. \( L_i \) is the set of fluents \( F_i \), such that \( \neg F_i \) is defined in \( D_i \).
\[2. \quad M_i \leq \neg L_i \text{ is a set of formulae of the form } \left(M_i \supset \neg F_i\right), \text{ at most one for each fluent } F_i \in L_i, \text{ where each } M_i \text{ is a formula containing no fluents drawn from } L_i, \ldots \cup L_n.\]

\[3. \quad E_i \leq L_i \text{ is a set of formulae of the form } \left(E_i \supset F_i\right), \text{ at most one for each fluent } F_i \in L_i, \text{ where each } E_i \text{ is a formula containing no fluents drawn from } L_i, \ldots \cup L_n.\]

We will now show that our translation rules always provide a solitary stratified theory in situation calculus. Recall that in our goal graph, depth of a node is the length of the longest path from a source to that node. From the set of soft-goals \(L\), let \(L_j \subset L\) be the subset with depth \(i\). Thus, \(L_0\) is the set of the sources (tasks, hard goals, CE). For \(i \geq 1\), \(L_i\) are soft-goals. Then the strata are shown in Table 5.1 and the corresponding partitions in Table 5.2.

| \(D_1\) | Successor state axioms of type (1) and (2) |
| \(D_2\) | Successor state axioms of type (3) |
| \(D_3\) | Ramification axioms of types (4)-(9) |
| \(D_4\) | Ramification axioms of types (10) and (11) for soft-goals in \(L_1\) |
| \(D_5\) | Ramification axioms of types (10) and (11) for soft-goals in \(L_2\) |
| \(\ldots\) | \(\ldots\) |
| \(D_i\) | Ramification axioms of types (10) and (11) for soft-goals in \(L_{i-3}\) |

Table 5.1: The stratified theory.

**Theorem 5.6.1** The theory \(D = D_1 \cup D_2 \cup \ldots \cup D_n\) of Table 5.1 is stratified with stratification \((D_1, D_2, \ldots, D_n)\):

\[
\begin{array}{c|l}
\mathcal{L}_1 & f_p(\bar{x}, s) \\
\mathcal{L}_2 & performed(\cdot, s) \\
\mathcal{L}_3 & v_s(r, \cdot, s), \text{ where } r \text{ is a task or hard-goal} \\
\mathcal{L}_4 & v_s(r, \cdot, s), v_d(r, \cdot, s) \text{ where } r \text{ is a soft-goal of depth 1} \\
\mathcal{L}_5 & v_s(r, \cdot, s), v_d(r, \cdot, s) \text{ where } r \text{ is a soft-goal of depth 2} \\
\ldots & \ldots \\
\mathcal{L}_i & v_s(r, \cdot, s), v_d(r, \cdot, s) \text{ where } r \text{ is a soft-goal of depth } i-3 \\
\end{array}
\]

Table 5.2: The partition.
Proof. To prove this we will show how the stratification of Table 5.1 complies with the definition.

- Each set of fluents $L_i$ is defined in stratum $D_i$, respectively, as seen on Table 5.2.
- For $i = 1, 2, 3$ stratum $D_i$ is trivially solitary with respect to $L_i$, $i = 1, 2, 3$, respectively.
- For $i \geq 4$, the axioms in $D_i$ are in the form $E_i \supset F_i$. To prove that $D_i$ is solitary with respect to $L_i$ we need to show that for every $j > i$, $E_i$ does not contain fluents from $L_j$. Indeed, for $i \geq 4$, $F_i$ is a fluent of the form $v_s(r_{l_i}, \cdot, \cdot)$ or $v_d(r_{l_i}, \cdot, \cdot)$ where $r_{l_i}$ is a soft-goal with depth $depth(r_{l_i}) = i - 3$ in the soft-goals graph. On the other hand, the partition $L_j, j > i$, contains (exclusively) fluents of the form $v_s(r_{l_j}, \cdot, \cdot)$ or $v_d(r_{l_j}, \cdot, \cdot)$ where $r_{l_j}$ is a soft-goal with depth $depth(r_{l_j}) = j - 3 > depth(r_{l_i})$. Since the depth of $r_{l_i}$ is less than that of $r_{l_j}$, we infer that there is no path from $r_{l_j}$ to $r_{l_i}$. Hence, nor is there a contribution link $r_{l_j} \longrightarrow r_{l_i}$. Since there is no such a contribution link, fluents of the from $v_s(r_{l_j}, \cdot, \cdot)$ or $v_d(r_{l_j}, \cdot, \cdot)$ do not appear in $E_i$. But such fluents is all what $L_j$ contains. Therefore, none of the fluents contained in $E_i$ are in $L_j$. □

5.7 The Preference Specification Language

Preference specification allows selection of behaviors that satisfy specific fitness criteria posed by stakeholders. Thus, instead of asking stakeholders to read and select from a vast set of alternatives, the stakeholders themselves describe what properties of the preferred behaviors are important for them. Alternatives that best satisfy those properties are then selected through automated search.

Our language for specifying stakeholder preferences is based on expressing priorities over temporal properties of behaviors implied by the goal model. Temporal properties are expressed through temporal logic based formulae, which we describe below.

5.7.1 Desire Formulae

We form desire formulae (DFs), or simply desires, to describe temporal characteristics of the behavior that emerges while goals are being fulfilled in a particular order and under certain circumstances. Linear Temporal Logic (LTL) is used to form DFs. Thus:

Definition 4.2 (Desire Formula - DF) A desire formula (DF) is an LTL formula formed with atoms from $H \cup L \cup T \cup R$. It is drawn from the smallest set $K$ for which:
1. \( H \subset K, T \subset K, R \subset K \).

2. If \( l, l_1, l_2 \in L \), then \( valS(l) \langle op \rangle c, valD(l) \langle op \rangle c, valS(l_1) \langle op \rangle valS(l_2) \) and \( valD(l_1) \langle op \rangle valD(l_2) \) are in \( K \), where \( \langle op \rangle \) is one of \( \leq, \geq \) and \( c \) is a real constant in \([0..1]\).

3. If \( \phi, \phi_1, \phi_2 \) are in \( K \), then so do \( \neg\phi, \phi_1 \land \phi_2, \phi_1 \lor \phi_2, \square\phi, \square\phi, \phi_1 U \phi_2 \) and \( \text{final}(\phi) \).

The symbols \( \square, \Diamond, \circ \) and \( U \), represent the temporal operators \textit{always}, \textit{eventually}, \textit{next} and \textit{until}, respectively. As opposed to CFs, which express a condition for a given time point, DFs describe properties of a whole sequence of tasks. For example, given a course of tasks, the statement \( \square(isAt(nurse, patientsRoom)) \) is true if the nurse is at the patient’s room at all times during that course.

Given a BF and a behavior for the root goal (therefore: a plan in the corresponding action theory), whether the behavior satisfies the BF can be evaluated by appealing to the situation calculus-based semantics of LTL given by Gabaldon ([47]). More specifically, let us use the notation \( \varphi[s, s'] \) to denote that \( \varphi \) holds in all situations from \( s \) to \( s' \equiv do(\bar{\alpha}, s) \). Also, \( s \sqsubseteq s' \) means that either \( s = s' \) or there is a sequence of actions \( \bar{\alpha} = \alpha_1, \alpha_2, ... \) such that \( s' = do(\bar{\alpha}, s) \).

The semantics of DFs in situation calculus terms are as follows.

\[
\begin{align*}
p(\bar{\alpha}) &\in R \text{ then } p(\bar{\alpha})[s, s'] \equiv f_p(x)[s] \\
g &\in H \text{ then } g[s, s'] \equiv \varphi_g[s] \\
t &\in T \text{ then } t[s, s'] \equiv performed(t)[s] \\
f &\in R \cup H \cup T \text{ then } final(f)[s, s'] \equiv f[s'] \end{align*}
\]

\[
\begin{align*}l_1, l_2 &\in L \text{ then } valS(l_1) \langle op \rangle c \equiv v_a(r_1, y) \land (y\langle op \rangle c)[s] \\
l &\in L \text{ then } valD(l) \langle op \rangle c \equiv v_a(r_1, y) \land (y\langle op \rangle c)[s] \\
l_1, l_2 &\in L \text{ then } valD(l_1) \langle op \rangle valD(l_2)[s, s'] \equiv v_a(r_1, y_1) \land v_a(r_2, y_2) \land (y_1 \langle op \rangle y_2)[s] \\
\Diamond\phi[s, s'] &\equiv (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s')\varphi[s_1] \\
\square\phi[s, s'] &\equiv (\forall s_1 : s \sqsubseteq s_1 \sqsubseteq s')\varphi[s_1] \\
\circ\phi[s, s'] &\equiv (\exists \alpha. do(\alpha, s) \sqsubseteq s')\varphi[do(\alpha, s), s'] \\
\phi_1 U \phi_2[s, s'] &\equiv (\exists s_1 : s \sqsubseteq s_1 \sqsubseteq s')\varphi_2[s_1, s'] \land (\forall s_2 : s \sqsubseteq s_2 \sqsubseteq s_1)\varphi_2[s_2, s']
\end{align*}
\]

Again, in the above, \( \langle op \rangle \) is one of \( \leq, \geq \) and \( c \) is a real constant in \([0..1]\).

### 5.7.2 Preferences over Desires

Two types of preference formulas are used: \textit{preference formulae} and \textit{weighted preference formulae}.
Definition 6.1 (Preference Formula (PF)) is a formula of the form $\phi_0[w_0] \succeq \phi_1[w_1] \succeq \ldots \succeq \phi_n[w_n]$, where $n \geq 0$, each $\phi_i$ is a DF, $w_0 \geq 0$, $w_n \leq 1$ and $w_i < w_j$ for $i < j$. When $n = 0$, preference formulae correspond to single DFs.

The satisfaction of a PF is assessed as follows. Define $d(\phi)$ be the satisfaction degree of a DF $\phi$, for a given behavior. If the behavior satisfies $\phi$, we set $d(\phi) = 0$, otherwise, $d(\phi) = 1$. Given a whole preference formula $\Phi = \phi_0[w_0] \succeq \phi_1[w_1] \succeq \ldots \succeq \phi_n[w_n]$ then $d(\Phi) = w_i$ where $i$ is the minimum $i$ for which $\phi_i$ is satisfied by the behavior or $d(\Phi) = 0$ if no such $i$ exists. PFs can then be used to construct weighted preference formulae as follows:

Definition 6.2 (Weighted Preference Formula - WPF) is a formula of the form $\sum_i (w_i \cdot d(\phi_i))$, where $0 \leq w_n \leq 1$, $\sum_i (w_i) = 1$, and $\phi_i$ a PF.

The weight of individual formulae $d(\phi_i)$ in WPFs is also calculated as above. Consider the model of Figure 5.1 again. A WPF over that model, given an initial configuration of the domain facts $\{p_1, \neg p_2, p_3, p_4, p_5\}$, can be given as follows:

$$(\square(valD(l_2) \leq 0.01)[0.0]) \times 0.7 +$$

$$(\Diamond t_2[0.0] \succeq \square(\neg p_2)[0.5]) \times 0.3$$

The first line is a simple DF (single-term PF) while the second line is a PF with two terms. Behaviors $[t_1, t_4, t_7]$, $[t_2, t_5, t_9]$ and $[t_3, t_5, t_9]$, satisfy the preference specification by degrees 0.15 (“good”), 0.7 (“worse”) and 1.0 (“worst”), respectively, and, thus, are preferred in that order. This can be intuitively verified by observing in Figure 5.1 that, for example, $t_9$ causes a significant increase in the denial value of $l_2 (valD(l_2))$, through a negative propagation link, that performance of $t_3$ violates $\square(\neg p_2)$ due to $t_3$’s effect, or that absence of $t_2$ from a behavior violates $\Diamond t_2$.

The preferences language we propose is a simplification of a more elaborate one presented in [17] for the purposes of preference-based planning. That work allows aggregation of preference formulae using both quantitative measures as noted here, as well as strictly qualitative measures, by considering discrete sets of weight labels rather than real values in $[0,1]$. On the other hand, that proposal does not include WPFs which turned out to be very useful in our case. Users of our goal-oriented framework that desire to adjust the expressive power of the preference specification language, can still use the same diagrammatic, translation and evaluation infrastructure introduced in this paper, but formulate preferences following [17].
5.8 In Practice

We will now refer to the example of Section 5.4 to describe how analysts can use the preference specification language to find behaviors that satisfy individual stakeholder desires and contextual circumstances. The problem of having a nurse attend to a patient can be formalized through a goal model such as the one in Figure 5.3. This goal model implies a number of alternative behaviors by which the root goal Nurse to Attend to Patient can be fulfilled. Preference specification aims at understanding the criteria that individual stakeholders use to identify subsets of behaviors that best fit to their individual needs. To achieve this, the analyst needs to a) specify stakeholder preferences, b) specify relevant situations, and, using these, c) identify behaviors of interest. We discuss each of these three steps below.

Figure 5.3: Example goal graph
5.8.1 Constructing Preference Formulae

Using the language described in Section 5.7, preference formulae are constructed by bringing together different desires from the same or different stakeholders and prioritizing them according to their relative importance. In general, the preferences over desires of the same stakeholder need to be analyzed before preferences of multiple stakeholders are placed together in a unified formula.

Consider, for example, the statement “we should definitely avoid anything that would make the patient unhappy”, expressed by the managers in our example of Section 5.4. In other words, while a behavior to attend to the patient’s needs unfolds, the patient should not be unhappy at any point. In our goal language this means that at any time, the denial value of the soft-goal Happy Patient must remain zero. Thus we would write the DF as follows:

\[ \Box (valD(\text{\textquoteleft Happy Patient\textquoteright}) \leq 0.01) \]

The expressiveness of LTL allows us to identify and represent various shades of meanings that stakeholder desires may have. Compare, for example, the above desire with the desire “in the end we want our patient to be happy”. We may choose to assume different temporal semantics between the two statements. Thus, as opposed to the former, the latter does not prevent denial of the goal Happy Patient in the middle of a behavior as long as this denial is replaced by satisfaction in the end; this is possible if the temporary denial is caused by a CE. Thus, the DF would be slightly different from the above:

\[ (\text{final}(valS(\text{\textquoteleft Happy Patient\textquoteright}) > 0)) \]

Therefore, LTL helps us define time intervals in which a desire to satisfy (or deny) a high-level goal is relevant. Consider for example, the desire “we should not avoid disturbing the nurse as long as the patient’s condition is severe”. This desire implies that the importance of the soft-goal Avoid Nurse Disturbance is relevant only when a certain condition is true and for as long as it is true. The formula to express this is:

\[ \Box (valD(\text{\textquoteleft Avoid Nurse Disturbance\textquoteright}) \leq 0.1 \rightarrow \neg \text{patientsCondition(severe)}) \]

Similarly, a behavioral detail may depend on the level of satisfaction or denial of a soft-goal. For example, “if the patient is unhappy for any reason, then the nurse should not skip the visit” would be again formalized as:

\[ \Box (valD(\text{\textquoteleft Happy Patient\textquoteright}) > 0 \rightarrow \Diamond (\neg \text{skipVisit})) \]

Observe how the use of soft-goals such as Happy Patient allows us to indirectly refer to desired operational level decisions without having to explicitly specify or even exactly know them at the time we construct the desire formulae.
Assume now that the management provides a combination of desires: “we should definitely avoid anything that would make the patient unhappy, but it would also be nice to increase nurses’ productivity somehow.” The statement implies a priority of the patient’s happiness over the productivity of the nurses. To represent this we construct a preference formula:

\[
(\Box(valD('Happy Patient')) \leq 0.1)) \times 0.8 + \\
(\text{final}(valS('Increase Nurses Productivity')) \geq 0.1) \times 0.2
\]

Thus, WPFs are used when the constituent desires are not mutually exclusive but they all contribute to the satisfaction of a preference. Furthermore, the nurse’s statement that “[they] don’t like the idea of talking to the patient remotely, but if they had to, they would at least choose to do so at the nursing station” can be formulated through this PF:

\[
(\Box(\neg \text{talkedWithThePatient})[0,0] \geq \Diamond \text{talkedFromTheNursingStation}[0.5])
\]

We use the PF when the satisfaction of a DF at a higher priority implies that we are indifferent about the satisfaction of desires at a lower priority, which is the case in this example.

Individual PFs and WPFs of various stakeholders can then be linearly combined into higher level WPFs, where the weights associated with each individual formula express the analyst’s perception over the relative importance of each stakeholder and her overall desires. Thus, by giving the preferences of the management a weight of 0.9 and to the nurses 0.1, the formula of Figure 5.4 below is the WPF resulting from combining the two individual formulas.

\[
(\Box(valD('Happy Patient')) \leq 0.1)) \times 0.72 + \\
(\text{final}(valS('Increase Nurses Productivity')) \geq 0.1) \times 0.18 + \\
(\Box(\neg \text{talkedWithThePatient})[0,0] \geq \\
\Diamond \text{talkedFromTheNursingStation}[0.5]) \times 0.1
\]

Figure 5.4: Preference Formula

For example, a statement posed by the nurses “we prefer to turn the request off before attending the patient” is a typical global precedence pattern in the LTL pattern system. According to the corresponding pattern, “nurseRespondedToCall globally precedes requestTurnedOff” is translated into:

\[
(\neg \text{nurseRespondedCall U requestTurnedOff}) \lor \Box(\neg \text{nurseRespondedCall})
\]

Obviously, the process of merging individual desires into preference statements involves the identification of the corresponding weights. The software engineering community has shown that the elicitation of priorities amongst competing requirements is possible in a variety of ways. Although the concepts under comparison in existing requirements prioritization frame-
works are coarse-grained software features, we believe that the same or similar techniques should also be effective for eliciting and quantifying priorities over behavioral and quality properties of the type we present here. We prefer here to focus on the modeling and reasoning aspects of preference specification, leaving empirical validation of this elicitation hypothesis for future research.

5.8.2 Specifying Relevant Circumstances

As we discussed earlier, a behavior (or sets thereof) is defined subject to initial conditions. Such conditions correspond to initial values of the domain facts in our goal model. Behaviors implied by the model of Figure 5.3, for example, are understood subject to the truth value of predicates such as \( \text{isAt}(\text{nurse}, \cdot) \), \( \text{isTime}(\cdot) \) or \( \text{patientsCondition}(\cdot) \).

In practice, by producing alternative configurations of the initial values of domain facts the analyst constructs alternative circumstances under which a behavior to satisfy the root goal must unfold. For example, \( \{\text{isAt}(\text{nurse}, \text{nursingStation}), \text{isTime}(\text{afternoon}), \text{patientsCondition}(\text{moderate})\} \) or \( \{\text{isAt}(\text{nurse}, \text{aisle}), \text{isTime}(\text{night}), \text{patientsCondition}(\text{severe})\} \), are different initial circumstances under which the root goal of Figure 5.3 might need to be fulfilled.

The choice of values for the initial circumstances depends on the needs of the particular exploration exercise. In fact, defining such circumstances constitutes construction of what-if scenarios, whereby analysts and stakeholders explore how environmental circumstances affect the suitability of alternative solutions, while holding the preferences fixed.

5.8.3 Identifying Behaviors of Interest

Having specified preference formulae and initial circumstances we can now select behaviors of the high-variability goal model that minimize the score of the preference formulae given those circumstances. To do so we can use the extended version of PPlan we presented earlier. Our extension of PPlan returns a ranking of the top N of the behaviors that optimize the score of the preference specification. Thus, given initial circumstances \( \{\text{isAt}(\text{nurse}, \text{nursingStation}), \text{isTime}(\text{afternoon}), \text{patientsCondition}(\text{moderate})\} \) for the goal model in Figure 5.3 the preference formula of Figure 5.4, the result is the following:
Chapter 5. Goal Alternatives and Preference Specification

<table>
<thead>
<tr>
<th>Rank</th>
<th>Behavior</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>([T_1, T_{11}, T_7, T_9])</td>
<td>.1</td>
</tr>
<tr>
<td>2.</td>
<td>([T_1, T_{12}, T_7, T_9])</td>
<td>.1</td>
</tr>
<tr>
<td>3.-12.</td>
<td>([\ldots, T_7, \ldots, T_9])</td>
<td>.1</td>
</tr>
<tr>
<td>13.</td>
<td>([T_1, T_{11}, T_3, T_9])</td>
<td>.18</td>
</tr>
<tr>
<td></td>
<td>(\ldots)</td>
<td>\ldots</td>
</tr>
<tr>
<td>24.</td>
<td>([T_2, T_3, T_{12}, T_9])</td>
<td>.18</td>
</tr>
<tr>
<td>25.</td>
<td>([T_1, T_{11}, T_5, T_6, T_9])</td>
<td>.23</td>
</tr>
<tr>
<td>26.-40.</td>
<td>(\ldots)</td>
<td>.23</td>
</tr>
<tr>
<td>41.</td>
<td>([T_1, T_{11}, T_3, T_8])</td>
<td>.72</td>
</tr>
<tr>
<td></td>
<td>(\ldots)</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

In the above table, \(T_i\) are abbreviations of the full task descriptions of Figure 5.3, as they appear in circular annotations next to the corresponding task; we do this for the interest of space. Thus, \(T_7\) is the task Nurse Talks through Mobile Device.

From the third term of the preference of Figure 5.4, we know that nurses don’t like to perform task \(T_7\), because it facilitates unnecessary disturbances by the patients. However, given that the managers’ opinion is given significantly more weight (0.9 versus 0.1 in Figure 5.4), this results to be unimportant in light of the potential productivity increase. Thus, in the above table behaviors that include performance of this task achieve the best score.

Nevertheless, if the nurses’ desires where given more importance the result would be quite different. Thus, if we choose to smoothen the difference between the manager’s and nurses’ weights to 0.7 and 0.3, respectively, then behavior No.13 in the ranking above, which does not include any communication with the patient (\(T_3\)), would be the most preferred. Therefore, different weights in a WPF result in different rankings of preferred behaviors.

Similarly, if the circumstances are different, the preferred behaviors are also different. Consider the WPF:

\[
\Box(valD(“Avoid Patient’s Disturbance”) \leq 0.1)[0.0] \times 0.7 + \\
\Box(valD(“Nurse Comfort”) \leq 0.1)[0.0] \times 0.3
\]

In circumstances in which \(isTime(night)\) does not hold, the score of the preference is minimized to 0.0 for any behavior of the form \([T_1 \ldots]\). The same behaviors, however, take a score 0.7 if the circumstances include \(isTime(night)\). In the latter case behaviors such as \([T_2 \ldots]\) are more preferred as they satisfy the WPF with 0.3.
This behavior evaluation process can be a useful tool for communicating with the stakeholders potential ways for achieving goals. By exploring the sensitivity of the output to alternative circumstances and weights, we achieve a better understanding of the problem and the actual desires of the stakeholders and what these desires imply in terms of low-level behaviors. Furthermore, the process may provide understanding of how desires of different stakeholders may actually conflict when a solution needs to be decided. In the example above, the desires of the nurses and the managers, although they don’t seem to be related when stated as high-level statements, they are found to be conflicting when interpreted into operational details.

5.9 Tool

Rankings of preferred behaviors can be produced using our prototype tool for evaluating preferences. The tool reads a goal model, a set of the domain facts representing circumstances of interest, and a preference formula, all in the form of Prolog predicates, and returns a set of sequences of tasks ranked by the degree by which they satisfy the preference formula. As mentioned above, the tool is heavily based on PPlan ([17]), which employs a best-first search strategy to find preferred plans. In the following subsections, we provide a brief overview of PPlan, describe the extensions we developed and discuss its performance in a number of examples that we ran.

5.9.1 An overview of PPlan

PPlan employs an A* best-first search to identify plans from a specified initial situation to a situation that best satisfies a given preference formula. Beginning from the initial situation and the empty plan, the algorithm progresses through possible next situations that form through the performance of actions, aiming at reaching a situation in which the goal formula is satisfied. Hence, at every step, the algorithm first identifies which actions satisfy their precondition axioms and can be considered as the next action to be performed. Thus, a list of potential extensions to the current partial plan (neighbors) are constructed and then ordered subject to an evaluation function, forming the frontier. The candidate with the best score in the evaluation function is pursued, and the same procedure repeats from there.

The evaluation function is a prediction of the best and worst score the preference formula can possibly acquire in later stages, given the current situation. These are calculated by examining whether it is possible for the basic desires of the preference formulae (which we here call
behavior formulae) to be true or false in subsequent situations, given the current situation and partial plan. For example, if in the current situation the fluent \( p \) holds, the desire \( \Box \neg p \) that may appear as part of a preference formula can obviously never be true, independent of what further choices are going to be made. In other words, both optimistic and pessimistic estimations for \( \Box \neg p \), if we continue on the current partial plan, are that it is false. On the other hand, again given that \( p \) holds, the desire \( \Diamond p \) is true and will stay true independent of further choices. Nevertheless, if \( p \) has been true in all previous situations, a prediction of the truth value for formula \( \Diamond \neg p \) can be either true (\( p \) continues to hold until a plan is found) or false (\( \neg p \) holds in some future situation due to the performance of an action). Similar observations can be made with formulas based on other temporal operators as well as compound ones; we refer the reader to [17] for a more formal account.

Thus, given a partial plan and a preference formula, each constituent desire of the preference formula can be evaluated with its best and worst possible truth values, providing us with overall optimistic and pessimistic weights of the preference formula, respectively. The evaluation function for the A* search is exactly the optimistic score of the preference formula given a partial plan and the current situation. Thus, given a set of candidate partial plans for the next situation, the one with the best optimistic score is chosen. In case of a draw (equal optimistic scores), the pessimistic score is used. If there is a draw there, too, the shortest candidate plan is chosen.

The evaluation function is admissible, which means that the first solution that is found is guaranteed to be the optimal. This is because the actual weight calculated once the plan is found, cannot be better than the optimistic weight estimated for partial plan in the search process.

### 5.9.2 Extending PPlan

Two extensions were considered to serve our purposes. One was a Prolog implementation of the ALP algorithm we discussed earlier and its incorporation to PPlans search process. Thus, the ALP algorithm runs as part of PPlan’s calculation of a progression. After the progression has been performed and a new set of fluents has emerged, the set is passed to the ALP execution routine which evaluates the impact of the new situation to the satisfaction and denial values of the softgoals. The fluents of type \( v_s(r, s) \) and \( v_d(r, s) \) are updated according to the results of the ALP execution.

Our second extension to PPlan is an enhancement of the existing heuristic aiming at ex-
ploiting the structure of the goal tree. In particular, at a given situation, where a subset of leaf-level tasks of the goal model have already been performed, it is possible to calculate an estimation of the maximum and minimum number of tasks that need to be performed for the root goal to be satisfied. Let $G$ be a set of nodes $g$ comprising an AND/OR decomposition tree. For every such node $g \in G$, let $g_{\text{min}}$ and $g_{\text{max}}$ be the minimum and maximum, respectively, number of tasks that need to be performed for the satisfaction of $g$. Also let $g^i$ be the $i$–th child of $g$ and $g^i_{\text{min}}$ and $g^i_{\text{max}}$ its corresponding distances. Then the procedure for calculating the minimum and maximum distance from achieving the goal $g$, given the set of all nodes given the set $T$ of leaf level nodes that have already been performed can be seen in Figure 5.5.

The distance of a candidate plan from satisfying the root goal is used together with the heuristics that are already employed in PPLan, but it is given lower priority. Thus, PPlan’s frontier is set to sort partial plans with the following order: i) Optimistic Weight, ii) Pessimistic Weight, iii) Minimum Distance to Goal, and iv) Maximum Distance to Goal. In other words, when comparing two partial plans, their optimistic weight is first checked, and the plan with the lowest value is picked. In case of a draw (the weights are equal) the pessimistic weight is checked, and the lowest pessimistic weight is chosen. If there is a draw in the pessimistic weights too, we choose the plan with the smaller minimum distance to goal. If there is a draw there, too, we choose the one with the smaller maximum distance to goal. If all these are equal we choose non-deterministically. In Figure 5.6, the basic PPLan algorithm is sketched together with the function COMPAREVAL, which used by SORTNMERGE BYVAL for comparing partial plans.

**Theorem 5.9.1** (Admissibility) *The score evaluation is admissible.*

**Proof.** Admissibility follows trivially by the fact the distance-to-goal criterion is given a lower priority than the optimistic and pessimistic weight criteria, which have been proven to constitute admissible evaluation ([17]). □
CalculateDistance(g, T)
INPUT:
T: A set of task already been performed
g: The goal for which we calculate distance
RETURNS:
g with its \( g_{\text{min}}, g_{\text{max}} \) values updated
BEGIN
if \( g \) is leaf then
  if \( g \) in \( T \) then
    \( g_{\text{min}} := 0; \ g_{\text{max}} := 0 \);
  else
    \( g_{\text{min}} := 1; \ g_{\text{max}} := 1 \);
  return \( g \);
if \( g \) is AND decomposition then
  for every child \( g^i \)
    \( g^i = \text{CalculateDistance}(g^i, T) \);
  \( g_{\text{min}} = \sum_i g^i_{\text{min}} \);
  \( g_{\text{max}} = \sum_i g^i_{\text{max}} \);
  return \( g \);
if \( g \) is OR decomposition then
  for every child \( g^i \)
    \( g^i = \text{CalculateDistance}(g^i, T) \);
  \( g_{\text{min}} = \min_i(g^i_{\text{min}}) \);
  \( g_{\text{max}} = \max_i(g^i_{\text{max}}) \);
  return \( g \);
END

Figure 5.5: Distance-to-Goal Calculation
PPLAN(state, goal, preferences)
  \[
  \text{frontier} := \text{INITFRONTIER}(\text{state}, \text{preferences})
  \]
  while \( \text{frontier} \neq \emptyset \)
    \[
    \text{current} := \text{REMOVEFIRST}(\text{frontier})
    \]
    \[
    \text{state} := \text{UPDATESTATE}(\text{current}, \text{state})
    \]
    \[
    \text{preferences} := \text{UPDATEPREFERENCES}(\text{current}, \text{state}, \text{preferences})
    \]
    if \( \text{goal} \subseteq \text{state} \) and \( \text{optW}(\text{current}) = \text{pesW}(\text{current}) \)
      return \( \text{current}, \text{optW}(\text{current}) \)
    end if
    \[
    \text{neighbours} := \text{EXPAND}(\text{current}, \text{state}, \text{preferences})
    \]
    \[
    \text{frontier} := \text{SORTNMERGEVALUES}(\text{neighbours}, \text{frontier})
    \]
  end while

Partial Plan \text{COMPAREVAL}(\text{Partial Plan } pl_1, \text{Partial Plan } pl_2)

/* Comparisons between two partial plans \( pl_1, pl_2 \) are performed as follows: */

  \[
  \text{if } \text{optW}(pl_1) \neq \text{optW}(pl_2) \]
  \[
  \text{return } \text{argmin}(\text{optW}(pl_1), \text{optW}(pl_2))
  \]

  \[
  \text{if } \text{pesW}(pl_1) \neq \text{pesW}(pl_2) \]
  \[
  \text{return } \text{argmin}(\text{optW}(pl_1), \text{optW}(pl_2))
  \]

  \[
  \text{gRoot}^1 = \text{CalculateDistance}(\text{gRoot}, pl_1)
  \]

  \[
  \text{gRoot}^2 = \text{CalculateDistance}(\text{gRoot}, pl_2)
  \]

  \[
  \text{if } (\text{gRoot}^1_{\text{min}} > \text{gRoot}^2_{\text{min}}) \]
  \[
  \text{return } pl_2
  \]
  \[
  \text{else if } (\text{gRoot}^1_{\text{min}} < \text{gRoot}^2_{\text{min}}) \]
  \[
  \text{return } pl_1
  \]

  \[
  \text{if } (\text{gRoot}^1_{\text{max}} > \text{gRoot}^2_{\text{max}}) \]
  \[
  \text{return } pl_2
  \]
  \[
  \text{else if } (\text{gRoot}^1_{\text{max}} < \text{gRoot}^2_{\text{max}}) \]
  \[
  \text{return } pl_1
  \]

  \[
  \text{return } \text{pickNonDet}(pl_1, pl_2)
  \]

Figure 5.6: Adapted PPlan
The rationale behind using the distance-to-goal criterion is that, according to our experience, the minimum number of tasks that need to be performed for the root of a goal model to be satisfied is usually proportional to the size of the goal model. The original version of PPlan ignores that, and biases towards examining all plans of a certain length before it decides to examine longer ones, even when the goal model suggests that there does not exist a plan of that length.

5.9.3 Performance Evaluation

We now take a look at the performance of our tool. While we expect it to be close to that of PPlan (discussed in [17]), we also expect that the distance-to-goal heuristic reduces the search time for certain types of preference formulae. In general, our experiments show that the distance-to-goal heuristic significantly boosts performance with simple preferences with mostly satisfiable constituent desires. However, when preferences are more complex and involve unsatisfiable desires, the performance of the heuristic may be less stable and occasionally worsen the performance of the original PPlan.

Our first experimental study is the sensitivity of the performance of the heuristic with respect to the minimum plan length that the goal model implies. We considered goal models with AND-decompositions only and varied the number of tasks and therefore the minimum plan length. We tried minimum plan lengths from 6 to 9. We removed any temporal constraints in order to maximize the search space. For each of these four models we constructed a similar set of preference formulae. The set contains a subset of satisfiable formulae (returning 0.0), a set of unsatisfiable formulae (returning 1.0) and a set of mixed ones (returning anything in between 0.0 and 1.0). Each set is tried in PPlan with or without the distance-to-goal heuristic. A Pentium IV, with 2.5Ghz CPU and 1Gb memory is used and stack and trail sizes in SWI Prolog are set to 128MB each.

Regarding the satisfiable set the comparison between the original PPlan and our extension is revealing of the effectiveness of our distance-to-goal heuristic. In Table 5.9.3 we simply average the running times over the satisfiable preferences. By increasing the minimum plan length, original PPlan’s execution time increases exponentially, while the heuristic-enabled version remains low. The star (*) in the table means that the program run out of memory in all cases (within 8 hours of computation).

However, unsatisfiable and mixed preferences do not exhibit such encouraging results. Instead, when preferences are unsatisfiable or mixed, the performance of the heuristic may
Table 5.3: Performance for satisfiable preferences.

<table>
<thead>
<tr>
<th>Min. Plan Length</th>
<th>Original</th>
<th>Heuristic-Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0.79 ± 0.02</td>
<td>0.04 ± 0.006</td>
</tr>
<tr>
<td>7</td>
<td>10.14 ± 0.05</td>
<td>0.05 ± 0.008</td>
</tr>
<tr>
<td>8</td>
<td>353 ± 3</td>
<td>0.07 ± 0.013</td>
</tr>
<tr>
<td>9</td>
<td>*</td>
<td>0.135 ± 0.013</td>
</tr>
</tbody>
</table>

demonstrate remarkable fluctuations. Characteristic is the case of our model with minimum plan length 7, which we present through a boxplot in Figure 5.9.3. The graph summarizes experimental results with 13 different arbitrarily constructed unsatisfiable and mixed preference formulae. Observe that while the median of the heuristic-enabled version is lower than that of the original version, there is significant fluctuation as indicated by the great distance between the quintiles.

Figure 5.7: Performance for unsatisfiable and mixed preferences.

Our exploration reveals that the distance-to-goal heuristic exhibits such negative results
when it builds a plan prefix (i.e. a partial plan) that leads to a situation with no solution or with a (“suddenly”) sub-optimal one. This can happen due to a choice that has been made early in the plan building process. The tenancy of our distance-to-goal heuristic to backtrack the progression towards plans that tend to be closer to the minimum distance-to-goal, may imply that the heuristic will take longer to correct the early choice. To confirm this we worked qualitatively. We developed the AND/OR tree of Figure 5.9.3. The “troubling” property of this tree is the precedence that connects \( t_1 \) with \( t_{13} \). A preference specification of that includes a desire for both, for example, \( t_2 \) and \( t_{10} \) will obviously lead our search procedure in a “trap”: the planner will look for plans beginning from \( t_2 \) which however implies that \( t_{10} \) cannot be performed. We are, then, questioning the ability of our distance-to-goal heuristic to escape the “trap” sooner than the original PPlan. In Table 5.9.3, the performance given particular preference formulae is given (times are in seconds). It is clear that while in simple formulae our heuristic performs better, formulae that lead the search routine in such traps can lead our heuristic to perform worse.

Figure 5.8: A goal model with non-local constraints.

As a last experimental step, we aimed at establishing the worst- and best-case performance boundaries of our heuristic-enabled tool. We experimented with a number of “artificial” goal models, such as the one in Figure 5.1 but of several sizes, and arbitrarily constructed preference formulae. From the several randomly constructed goal structures, which vary in terms of the number of OR-decompositions and temporal constraints, we chose ones that appeared to generally worsen performance. We run the experiments on the same hardware infrastructure as above (a Pentium IV, with 2.5Ghz CPU and 1Gb memory). Using the number of leaf level
### 5.10 Application and Findings

#### 5.10.1 Preliminary Evaluation

We considered our nursing case study for an initial feasibility evaluation of the techniques we described in this chapter. As in the previous chapter, we organized the application process as follows. Acting as goal analysts we iteratively developed goal models and temporally extended them to a point where we thought that the result would reflect our understanding of the domain. We then attempted to construct simple desire and preference formulae for each stakeholder, again based on our sense of what the desires and preferences of the stakeholders we interviewed were. We then qualitatively examined the result and reflected on the processes.
The result suggests that both temporal extensions and preference formulae are possible, as we had hypothesized in the beginning. “Possible” means usable (i.e. possible to put together the elements of the language) and potentially valid (i.e. leading to results that can be argued to reflect the reality of the domain). In terms of temporal extensions, qualitative analysis of the result indicates extensive use of temporal elements (such as CEs and precedence links), which suggests that these were actually usable. Validity of the result was further investigated and confirmed by randomly producing admissible behaviors and testing them against our intuition of the domain.

In terms of preference formulae our results indicate that formulation of such is possible. While desire formulae were closer to actual desires expressed or implied by stakeholders during interviews (e.g. nurses wanting to avoid visiting patients for no reason), the construction of preference rankings thereof is mostly artificial yet judged as a realistic possibility given, again, our understanding of the domain and its stakeholders. The only part of preference formulation that maintained a degree of obscurity is the choice of numbers for the weights of individual desires; in our application several different ad-hoc choices of numbers turned out to make equal sense. We will return to the numbers issue below.

Apart from this empirical evidence on preference formulation, we believe that the current bibliography further supports the possibility of constructing valid preference formulae for a domain. The problem of constructing desires is, at least partially, a problem of formulating requirements in LTL. This is assumed to be possible by established formal requirements modeling frameworks, such as KAOS ([31]), and largely in the model checking community (e.g. [25] and [59]). The possibility of specifying desired degrees of satisfaction of soft-goals, producing sub-formulae similar to our e.g. \( valS(\cdot) \geq c \) construct is also assumed in [108]. In addition research on requirements prioritization ([73]) claims empirically tested methods for numerical characterization of priorities amongst features. This is close to our preference specification, with the important difference that we suggest behavioral formulae instead of software features. The effect of this difference needs to be measured through follow-up experimental studies. We however believe that the bibliographic evidence is strongly suggestive of the feasibility of LTL-based preference formulation.

In addition to the nursing domain we also tried the same process on other domains through the construction of artificial examples. Thus, we also created models for the meeting scheduling problem (about 75 goals), for the ATM domain (about 50 goals), as well as for a (hypothetical) on-line bookstore (about 30 goals). The results also supported our feasibility hypotheses, though in these applications the validity argument is weaker due to the pure artificiality of the
5.10.2 Experiences and Lessons

Our application both on the nursing case study and the other examples helped us form a better view on strengths and weaknesses of our approach as well as the conditions under which these are exhibited. Our findings can be summarized as follows.

Firstly, adding the simple temporal extensions to goal models is natural and does not appear to obstruct the model development effort or comprehensibility of the result in any way.

Secondly, specification of temporal preferences was found to be particularly useful for adding soft-constraints where hard precedence constraints could not possibly be decided for the general cases. The usefulness of preference specification was particularly revealed in the bookstore example, in which alternative ways to temporally arrange the business process of selling books could be explored through preference specification. For example, dilemmas such as whether to ship an order before or after a payment has been received or whether to order from suppliers before the customer has confirmed her own order, can be left for individual booksellers to decide upon through specifying their preferences.

Thirdly, soft-goals appeared to be useful for hiding operational details such as the above. In the Bookstore example, soft-goals such as Reduce Payment Delay Risks, Maintain Happy Customer, Maintain Minimum Stock and preferences thereof appeared to be able to facilitate the corresponding operational decisions (e.g. when to ship an order), without direct reference to the latter. Such reliance on soft-goals, however, appeared to require careful selection of contribution values and explicitly showing how the context influences such contributions. In the bookstore example, any influence to the soft-goal Reduce Payment Delay Risks depends on the amount of the payment in question. We found the use of CE s as an effective way to add conditions to soft-goal contributions.

Fourthly, in some cases, the construction of LTL formulae seemed to be a non-trivial process, although no need for complicated or deeply nested formulae was revealed. We used LTL patterns to avoid this problem. The LTL-pattern system, proposed in [35], offers a mapping of natural language descriptions of, for example, precedence, response, bounded existence into templates of LTL specification. We frequently referred to these patterns to ensure correctness of desire formulation. We believe that the pattern system offers a first step towards making our preference specification framework useable by users that lack knowledge of LTL.

Fifthly, the ad-hoc process that we followed for choosing numbers for the weights of the
individual desires in preference formulae may lead to disputable results. This observation was not unexpected, as this quantification has to be based exclusively on purely subjective data. However, we argued above that the choice of weights can be investigated in the context of any prioritization process that involves specific procedures for translating rough priorities of stakeholders into actual weights. We offered AHP ([73]) as an example. In addition, qualitative ways for ranking preferences have been proposed ([17]) and our framework is of course readily adaptable to such qualitative measures as well. By using real numbers in this chapter we simply explore the upper limit of preference specification in terms of expressiveness. Thus, in practice, users may not use arbitrary real numbers for specifying preferences, but, for example, quantize the real interval into a small subset of samples, or directly define qualitative values together with a way to aggregate them as in [17].

Surprisingly, most of the difficulties and issues that emerged concern the AND/OR decomposition structure itself and several expressiveness limitations it exhibits. We will discuss potential extensions of the goal decomposition structure in the last chapter.

5.11 Summary

In this chapter, we introduced a method for specifying fitness criteria for selecting goal alternatives that best match a given problem instance. We introduced a preference specification language, through which analysts can describe both high-level qualities and low-level temporal properties of preferred solutions, using the expressive power of LTL. Extensions to the existing notation for building goal models were proposed to allow the expression of behavioral variability with respect to how stakeholders may wish to achieve their goals. Through translating the goal model to situation calculus, preference specifications can be used to identify subsets of behaviors of the goal model, that best match with desires of individual stakeholders. A tool based on an existing preference-based planner was built to automate this process.

In the following chapter, we will see how the preference specification framework we introduced can potentially be used for configuring software, after associating the generic alternatives in the goal model with configuration options of the software system.
Chapter 6

Goal-Based Software Configuration

6.1 Overview

In the previous two chapters we presented a method for acquiring variability in goal models and a technique for selecting alternatives that best fit specified criteria. We showed that this is useful for understanding the variability of a problem and for exploring solutions that best match the individual constraints and stakeholder priorities that characterize a particular problem instance.

Nevertheless, the major motivation for developing these techniques is the potential of associating this variability with customizations of the actual software system. If such an association were possible, goal models would, firstly, provide a guideline for designing the configurability aspect of software systems ([29]), and, secondly, they would allow the configuration task to be a matter of criteria specification and alternatives’ selection at a goal level, rather that the technical level of the configuration details. In this chapter\(^1\) we present an initial effort to understand how goal alternatives can be associated with configuration options of existing systems, and provide a demonstration on how preference specification over goal alternatives, the way it is presented in Chapter 5, can potentially be used for configuring existing systems. In particular, we introduce a method for constructing goal decomposition models by reading the configuration screens offered by software applications. This allows leaf-level tasks of the goal models to be directly associated with configuration values of the application, which paves the way for using preference specification as a way to configure software via the overlaying goal model.

We organize our presentation in this chapter as follows. In Section 6.2, we provide an

\(^1\)The material presented in this chapter has been published at the 13th IEEE International Conference on Requirements Engineering (RE’05). See [79] for full citation.
overview of the configuration problem in common personal software and explain why the current state of practice poses certain difficulties. As an example, in Section 6.3.1, we then take a close look at the “Options” widget of Mozilla Thunderbird. We examine its structure and discuss its problems. Then, in Section 6.3.2, we discuss related research work. In Section 6.4, we present a series of steps that can be followed for the construction of the goal model, and, in Section 6.5, we show how we can use the goal model to configure the software system. In Section 6.6, we also investigate the use of goal model parameters that could facilitate the association of goal models with low level parameter configurations. We discuss how well our method performed in practice in Section 6.7 and we conclude in Section 6.8.

6.2 The Problem of Configuring Common Personal Software

With the term common personal software, one can refer to software systems that are developed to be installed and used in home or mobile personal computing and communication devices, in order to support everyday life activities such as personal communication, learning, entertainment etc. For example, one can use a word processor to write one’s school homework, a web browser to read the news, an e-mail client and several instant messaging systems to stay in touch with one’s friends and relatives or media viewers and editors to maintain one’s photograph/video albums. Although such systems are used extensively at the workplace as tools that support business productivity, they are also intended to be used as personal software systems.

When they are viewed as such, though, two basic issues may arise that make the corresponding requirements engineering effort difficult. Firstly, unlike software systems that are used at work, personal software cannot be seen as a production tool to which the user has to adapt as part of her responsibilities. On the contrary, because personal software generally faces the unrestricted human creativity and behavior in their leisure hours, it is the designer’s responsibility to make the system adapt to the goals, preferences and abilities of the user. Secondly, the intended users of personal software are many and diverse to such a degree, that it is practically impossible to acquire requirements and produce a unique software system for each one of them. Hence, there is a need to develop software that can be easily personalized to accommodate the needs of each intended user as well as possible, without requiring from them any effort or technical knowledge.

However, a close look at today’s practices reveals that ease of personalization is still questionable. In the general case, the user purchases/downloads one of the system’s “editions” (e.g. the “professional” versus the “home” edition), installs the components that she needs and
further configures the details of the installed system hoping to perfectly tailor it to her needs and abilities. In all these personalization phases, the user deals with a number of screens that display features (or packages of features) to be selected, deselected or adjusted through the definition of parameters. In some cases, special configuration agents (“wizards”) are used to direct the user configure the necessary groups of options. But the parameters and options usually refer to the solution domain rather than the domain of the problems they can potentially solve. Hence, on one hand, they are expressed in a technical language that is not understood by users with little or no computer expertise. On the other hand, even if the user understands what she configures, she is unaided in her effort to come up with the configuration that best suits her needs. Further, the number of configuration options inevitably increases as the functionality of systems grows, making personalization an increasingly cumbersome process.

We believe that the basic problem behind the current practice is that it does not take into account the user goals behind configuration. In other words, the designers of the configuration widgets do not seem to ask why the particular set of options should be exposed to the user, and why a user would configure them this way and not another. Hence, our goal in this chapter is to show how goal models can be constructed by studying such configuration screens, and how they can be used as a mediator between the user and the system, so that users can configure the latter by simply referring to the specifics of their problem, through preference specification.

To illustrate our approach, we consider configuration that takes place after a personal software system has already been installed on a computer. At this point of the life-cycle, configuration is done through the use of specially designed dialogue windows under the title “Options”, “Preferences”, “Customize”, “Settings” etc (from now on: the “Options” widget). The vast majority of software of the genre contains at least one such dialog window. In the rest of the chapter, we will use the term configuration\(^2\) to refer to personalization that takes place at this phase. We focus on the “Options” widget of a particular e-mail client, namely, Mozilla Thunderbird 0.5 ([123]). Mozilla Thunderbird is the e-mail component of the Mozilla application suite, offered as a separate stand-alone product. We chose Mozilla for this study because it is the most popular open-source and multi-platform e-mail client available to date. In light of Mozilla’s accessibility, no barriers are introduced in replicating, confirming or challenging the findings presented in this paper. We further believe that observations made in Mozilla reflect to an adequate degree the general practice for designing “Options” widgets.

\(^2\)The use of this term should not be confused with the one found in the literature about configuration management.
6.3 Current state of Research and Practice

6.3.1 Exploring the “Options” Widget

Let us have a close look at the “Options” widget of Mozilla Thunderbird. We focus on the screens under Tools -> Account Settings... and Tools -> Options... menu items. The former contains default settings for each of the accounts that have been created for this instance of the application. The latter contains application-wide default settings.

We define configuration item to be any visual control (e.g. text boxes, drop-down menus, sets of radio buttons) with which the user can view and change configuration information. The domain of the configuration item is the set of possible values it can take. For instance, the text box Account Name accepts a (practically) infinite number of strings, whereas the checkbox labeled Check spelling before sending accepts only two values (“enabled” or “disabled”). A configuration of a set of items is simply a set of values to which these items can be set.

We examine a total of 17 configuration screens that are included in these two dialog windows, ignoring a small set of dialogs that can be called from these screens. We count a total of 120 configuration items. Although we cannot assume that this is the complete set of items that the system offers, they certainly constitute a good sample for discussing the nature of the problem and for performing an initial evaluation of our proposal.

We first investigate what entity each configuration item configures. We observe that there are five such entities: the system, the application, the e-mail account, the outgoing message and the incoming message. Each configuration item configures one of these entities. But it may be defined in a different entity from the one it actually configures. For example, although the boolean item Check spelling before sending configures each outgoing message, the “Options” widget keeps the configuration value as part of the e-mail account. This value serves as the default value for every message sent out from the particular account. 69 out of the 120 configuration items we examined would be part of objects other than the ones they actually configure.

This scheme (i.e. pushing configuration options to container entities as defaults) serves an obvious practical reason: we do not want the user to configure “from scratch” every object she creates. Nevertheless, the choice of accommodating these defaults to one of the entity’s containers is arbitrary. For example, the default value for whether to Check spelling before sending could depend on who the recipients are or which time of the day it is
send; these possibilities are not less reasonable than having the particular default option depend on the account from which the message is send.

More problems are revealed if we approach the configuration items from a point of view of their meaning. Figure 6.1, for example, depicts the screen where the user can configure the fonts to be used for messages. In short, even if the users know what the difference between “Monospace”, “Serif”, “Sans-serif” and “Proportional” is, it is not certain whether they have a strong opinion about which of the available values is the one they really need.

![Figure 6.1: Configuring the Fonts for Mozilla Thunderbird](image)

Furthermore, the number of different combinations of values in the upper frame of the screen depicted in Figure 6.1 is at least 80 billion. Detection of configurations that will not reasonably be used (obviously there are such) is arguably possible.

We can summarize the problems with the current practice of designing “Options” widgets as follows. Firstly, the configuration items are arbitrarily many and organized in an ad-hoc manner, making it impossible for the user to handle them without significant effort commitment. Secondly, due to this complexity, a one-size-fits-all approach to configuration has to be followed. Such an approach demands the configuration parameters of an object to be statically defined as default configurations of the entire class of such objects. Thirdly, the user might not be capable of understanding what the available options mean; even if she knows what the available options mean, the user might fail to understand which of them is the most appropriate according to her needs, capabilities and generic preferences. Fourthly, there might be...
configuration items or values, that do not interest a particular user, as well as combinations of configuration values that could not reasonably apply to any user.

In conclusion, the current approach to configuring common personal software is one that ignores the importance of user requirements. The users are expected to literally intervene to the software system’s design, by interpreting their goals and requirements into configurations, without any external help. As we discuss in the next section, complementing the discussion of Chapter 2, requirements for software customization is a subject that has not enjoyed the appropriate attention by the SE and RE communities.

6.3.2 Related Work

The problem of configuration, as described so far, has mostly been studied from a Human Computer Interaction point of view. In [81], a number of users are observed subject to when and how they customize a number of software applications. Among other things, it was found that users cannot afford the time needed to customize the software, given that there is also a risk of failing to achieve the intended result. Thus, users would not go to the trouble of configuring their systems, when this seemed “too hard” and the respective documentation was not rich enough. In a similar investigation, reported in [93], it was also found that users are more likely to engage to customizing their systems when the respective task is easy.

In [84], McGrenere et al. propose a method for customizing the user interface of a word processor. According to the proposed technique, the interface should initially contain only a basic core of the system’s functionality. While using it, the user gradually adds the functions she needs, leading to an interface that best suites to her. However, the user is assumed to know how to use the system with optimal efficiency and effectiveness and is therefore capable of making the correct personalization choices easily. Furthermore, the focus remains on the availability of functions on the main interface, which is only a subset of what can be customized in a software system.

In [82], the construction of models that describe the design rationale of an intended user interface is proposed. The models imply a space of design alternatives, each contributing positively or negatively to high level selection criteria. Although the modeling idea is very similar to the one that we are using here, the design rationale paradigm alone, as described in [82] is not sufficient for our purposes. Firstly, it does not follow the necessary user-centered approach, as it assumes that the same rationale model can be read from both users and designers. Secondly, it does not necessarily focus on describing the intentional content of a potential customization
decision; it rather focuses on articulating and understanding visual and behavioral aspects of the system without asking why these aspects are considered in the first place.

From a Requirements Engineering point of view, work on requirements monitoring ([41]) is also related to our discussion. In [39], Feather et al. have proposed the use of monitors that detect violations of requirements specifications, and change the systems’ behavior accordingly. The goal is to optimize the effectiveness and efficiency of low level task performance according to observed behavior. This can be done by changing the parameters of the system’s functionality or even switching to a different design. Again, though, high level user preferences or system-wide qualities are not taken into account.

As we discussed in earlier chapters, goal models can effectively be used to represent such high-level concerns and their relationships. In addition, they allow measurement of the impact of low-level decisions (alternatives in the hard-goals graph) to high-level qualities (soft-goals), allowing us to use preferences over the latter to select alternatives of interest. By associating these low-level alternatives with software configurations, preferences automatically become tools for configuring software. In the next section, we show how such an association can be constructed by constructing the hard-goals graph in a bottom-up fashion through studying the items of the configuration screen.

6.4 From Configuration Options to Goals

The method for establishing the mapping between configuration options and goals, is based on examining the candidate values of each configuration item and identifying both the user goals that each selection achieves and the qualities it contributes to. The result is a goal model built on top of the configuration items. The process of constructing the goal model is organized in a sequence of steps, which we describe in the following subsections, using examples from the Mozilla Thunderbird case.

6.4.1 Step I: From Configuration Items to Goals

We create a list of the configuration items of our “Options” widget. For each of the items we ask two questions:

- “What is the goal that is achieved by the function being configured?”
• “What high level aspects of the user experience are influenced by the value that is selected in this item?”

The first question refers to the functional goal that is associated with the particular configuration item, for example Use Encryption or Check Spelling. The second question refers to qualities of the system as a whole that reflect generic user goals and preferences. Privacy is an example of such a goal. Obviously, in our goal modeling framework the former are modeled as hard-goals and the latter as soft-goals.

Consider, for example, the pair of configuration items depicted in Figure 6.2. This pair of items allows the user to define whether she wishes the system to periodically connect to the mail server and check for new messages and if yes, how often this should happen.

At this step our method requires us to understand what goal we are trying to achieve by the function being configured. Trivially in this example, the purpose is to Be Notified for New Messages. But different alternatives in terms of the frequency with which this check can occur, imply different contributions to particular soft-goals. Thus, having very frequent checks contributes positively to the soft-goal Increase Availability, in a sense that the user can respond to the senders promptly, making communication more intense and productive. But this contributes negatively to the goal Reduce Network Use/Dependency, as a connection to the mail server will need to be established every little while. Further, the subsequent notification frequency contributes negatively to the soft-goal Reduce User Distractions. Obviously, when less frequent checking is chosen, the contributions are defined accordingly.

6.4.2 Step II: Alternative Options as OR-decompositions

Having identified the goal behind the function being configured, we continue by assuming that each value of the configuration item implies an alternative way to satisfy that goal (perhaps including the option not to satisfy it). By picking one of the possible values for the configuration item, the user specifies one of her possible intentions on how she expects the system
to satisfy a goal of a higher level. This conceptual pattern can be nicely illustrated by an OR-decomposition of the original goal. Such decomposition is depicted in Figure 6.3.

The goal **Be Notified for New Messages**, which was identified for the items of our example in the previous step, is decomposed into a small set of subgoals, each expressing the different frequencies with which the checking can be performed. Each of the alternative subgoals is associated to a specific value of the configuration items. Thus, **Be Notified as Frequently as Possible** would check for new messages every minute, whereas **Be Notified as in Regular Mail** means that the system should check for new mail daily.

By expressing groups of configuration items as goal decompositions, we have reduced a practically infinite domain to a small set of meaningful values. Moreover, we have understood why the particular configuration items are there and what would alternative values of the item mean in satisfying (or not satisfying) this purpose.
6.4.3 Step III: Alternatives and their Impact to Soft-goals

Having defined the OR-decomposition in the previous step we can now question each alternative subject to its impact on the soft-goals we identified in the first step. In Figure 6.3, soft-goals are represented using cloud shaped elements. For example, choosing Be Notified as in Regular Mail, which in turn implies that the checking occurs every 24 hours, contributes positively to the soft-goal Reduce User Distractions. But, as we saw previously, this choice also contributes negatively to the softgoal Increase Availability. In Figure 6.3, we represent this by adding the corresponding contribution links between the goals. The contribution to these soft-goals can be further propagated to soft-goals of a higher level. In our example, the soft-goal Reduce User Distractions contributes positively to a more generic one we have called Productivity.

Notice that we use the qualitative modeling framework and its corresponding contribution degrees: $+,++,--,--$ (see Chapter 3). Moreover, we assume symmetric propagation of satisfaction and denial values. Thus, $g_1 \leftrightarrow^{++} g_2$ means both $g_1 \leftrightarrow^{+s} g_2$ and $g_1 \leftrightarrow^{+D} g_2$.

6.4.4 Step IV: Integration

If we follow the same steps for every configuration item of the system under investigation, we end up with a forest of elementary goal decomposition trees. For Mozilla Thunderbird the result is depicted in Figure 6.9. A closer look at the diagrams will reveal that many soft-goals receive multiple positive or negative contributions from different goal trees. Moreover, soft-goals contribute to each other the way we described above. In Figure 6.4, we isolate all soft-goals in a separate diagram and relate them through contribution links, also introducing soft-goals of an even higher level, when possible. The result of this process allows us to talk about the configuration of the whole system using a high level non-technical language. More interestingly, if we pose constraints on the total degree of contribution that particular soft-goals can accept, these constraints will be propagated to the low level, requiring particular configuration values to be set, without the user having to intervene.

In other words, as we discuss next, having concluded the development of the goal model, we can move conversely and use it in order to leverage low level configurations.
6.5 Using the Goal Structures

6.5.1 Goal Driven Configuration

The models that have been developed in Figures 6.4 and 6.9 can be used for both assessing a given configuration in terms of how it meets high-level user goals and for producing a configuration by specifying preferences over soft-goals of interest, the way we discussed them in Chapter 5. In this section, we will discuss these two opportunities in more detail.

6.5.2 Assessing Existing Configurations

Configuration assessment can be performed through direct use of the bottom-up LP algorithm (see Chapter 3) over the low level hard-goals that are satisfied with the given configuration values. Recall that each low level subgoal is related to a particular configuration value of one or more items. Thus, given the configuration of these items, we may be able to match them...
exactly with one of the subgoals and label the latter as fully satisfied. Then, we can apply the algorithm to calculate the degree of satisfaction (or denial) of each of the high level soft-goals.

Consider again the example of Figure 6.3. If we know that a particular user instance has configured automatic checking of new messages every 1 min, then we infer that the goal Be Notified as Frequently as Possible is satisfied. Application of the LP algorithm will help us calculate the satisfaction and denial values of the overlaying soft-goals. In Figure 6.5, soft-goals are annotated with the pair of satisfaction and denial values they acquire for the given configuration.

![Figure 6.5: Assessing Alternative Configurations](image)

Of course, due to the sampling we have performed to reduce the size of the domains, many configurations might not have an exact correspondence to a low level goal. In this case, we need to select a hard-goal that is best satisfied by the configuration at hand. In the example of Figure 6.3, for a configuration $x = 50$ of the option Check Messages every $x$ Minutes we would decide that the goal Check very Frequently is satisfied and provide this as an input to the LP algorithm.
6.5.3 Preference-Based Configuration

The goal models we have developed are suitable for application of the analysis techniques we introduced in Chapter 5. We can construct preference formulae and use the reasoning tool we introduced there, in order to select low level hard-goals that best satisfy these formulae. These low level goals are, in turn, directly connected to configuration values allowing thereby configuration of the software system through high-level preference specification.

The preference formulae we can construct for this purpose are simpler than the ones we saw in Chapter 5, as there is no temporal dimension to consider. In the case in which AND-decompositions occur in our model (through logical groupings of elementary OR-decompositions), we add strict temporal constraints so that no permutations of the same set of tasks is considered by the tool. Also, in the preference specification we use only statements of desired satisfaction or denial of soft-goals which we turn into Behavior Formulae (BF) by externally adding the final() temporal operator. This way our tool for reasoning with preferences about temporally extended goal models can also be used for simple time independent goal models. In the example below we omit the temporal operator final() for simplicity. Assume that a user poses the following preference formula:

\[ \text{val}_S(\text{‘Increase Availability’}) = ‘F’)[1.0] \geq \text{val}_S(\text{‘Increase Availability’}) \geq ‘P’)[0.5] \]

With this formula the user expresses the fact that under her given circumstances, it is important to be available to some degree, if not at a maximum one. The alternative that best satisfies this preference specification is to Check [for new e-mail] as Frequently as Possible which is in turn interpreted into checking every 1 minute (Figure 6.6). At different circumstances Productivity might be an important quality, implying that configurations with less frequent notifications will be preferred.

By using preferences to configure software allows us both to hide the design details from the users and to aggregate configurations that are relevant subject to an intentional aspect as defined through a preference statement. However, designing an interface for effectively eliciting user preference statements such as the one above is not a trivial problem. In the next section we explore some characteristics we believe such interfaces should have.
6.5.4 Redesigning the “Options” Widget

Bottom-up and top-down analysis of impact propagation introduces new possibilities for the design of configuration interfaces. Interfaces that support goal-based configuration can be built in order to function in two modes. The untrusted mode allows the user to edit low level details of the system, while ensuring she maintains awareness of the impact of their options (“macro-awareness”), using bottom up analysis of the goal model. The trusted mode allows the user to set desired degrees of soft-goal satisfaction while making her aware of the changes that automatically occur in the low level options (“micro-awareness”). Different kinds of users may choose different modes and appreciate micro- or macro-awareness in different degrees. Thus, computer experts and “techies” will probably choose to work with the untrusted mode, whereas elderly, children and people without technical background may choose to use the trusted mode and even turn micro-awareness off.

The exact way with which desired satisfaction of soft-goals is defined in the trusted mode poses an interesting interface design problem. The obvious practice of directly assigning degrees may prove unintuitive, especially if we consider the inevitable trade-offs between satisfaction of different soft-goals. Direct preference specification as we described in the previous section would perhaps be more promising, if the users were significantly assisted in their effort
to construct such rankings. Preference specification in this case would be a statement the user makes under given circumstances, rather than a long term decision. For example the comparison between Productivity and Availability we discussed above is not meaningful unless it is made with respect to specific conditions the user is in. The user will spontaneously specify the preference in response to a change in her conditions (e.g. she discovered she has a lot of work to do versus she is expecting somebody to contact her). But again, what is the language (visual or other) that is usable enough for the common user? Clearly, the problem of constructing the appropriate interfaces for eliciting user preferences requires an extended investigation over several dimensions.

6.6 Gaining Accuracy: Parametric Goal Models

The method we have been presenting relies significantly on the correct use of numbers. These occur in two cases:

1. When the domain of a configuration item that is continuous needs to be appropriately sampled in order to be associated to a small set of subgoals.

2. When a contribution to a soft-goal needs to be defined.

Consider, for instance, the configuration items given in Figure 6.7. With these items, the user can define whether the system should ask permission before downloading a message that exceeds a particular size. The corresponding goal decomposition can be seen in Figure 6.8 (ignore the soft-goals for the moment). The user may choose either to allow all messages to automatically be downloaded, or to allow those that do not exceed a particular size category (small (i.e. without attachments), medium and large). However, we should not expect a general agreement among users of what can be considered as a medium message (or a small or a large one respectively).

Figure 6.7: “When should I truncate?”
Figure 6.8: Parameterizing goal models

Hence, we should rely on parameters which are valuated using facts particular to each user. In our example, we can use simple statistics based on the user’s e-mail traffic. The result would be descriptions like the following:

\[ \text{[MsgSize].[S]: Average size of messages that do not contain attachment. 50k by default} \]

\[ \text{[MsgSize].[M]: Average size of messages that contain at least one attachment. 1Mb by default} \]

... 

Such parameter descriptions are essential part of the goal model as they further enhance its adaptability to different cases of users.

The same approach can be used to parameterize degrees of contribution. In Figure 6.8, let us focus on two of the soft-goals that are influenced by the several options. The one is Avoid Occasional Network Congestions that may occur when downloading a large attachment through a slow connection; the goal refers to the frustration or reduction of productivity this may cause. The second is Reduce Connection Time Cost which might be of importance depending on the charging policy of the internet service provider.

The degree by which downloading a message of a particular size can hurt these goals cannot be generally assessed. Thus we will again use parameters. For the first goal we will base the estimation on the time it takes for the message to download given the available bandwidth:

\[
e = \begin{cases} 
N, & \text{if } \frac{(\text{selected msg size})}{(\text{bandwidth})} < 2 \text{ sec} \\
-, & \text{if } 2 \text{ sec} < \frac{(\text{selected msg size})}{(\text{bandwidth})} < 1 \text{ min} \\
-- &, \text{if } 1 \text{ min} < \frac{(\text{selected msg size})}{(\text{bandwidth})}
\end{cases}
\]
In the above, \(N\) implies absence of the link. Similarly, the contribution to the goal Reduce Connection Time Cost

\[
f_{\text{score}} = \frac{\text{(selected msg size)}}{\text{(bandwidth)}} \times (\text{cost per second}) \times (\text{scaling factor})
\]

Here, the scaling factor expresses the “significance” of the objective cost for the particular user, in a way that the formula returns a value between 0 and 2. The final contribution degree \(f\) can be the following:

\[
f = \begin{cases} 
N, & \text{if } f_{\text{score}} < 0.5 \\
-, & \text{if } 0.5 \leq f_{\text{score}} < 1.5 \\
--, & \text{if } 1.5 \leq f_{\text{score}} < 2
\end{cases}
\]

Models for parameters \(d\) and \(g\) can be constructed similarly.

Of course, all models we mention above are naive examples and are given in order to illustrate how we can define parameters in goal models. How elaborate the parameter models should be, varies depending on the degree of granularity and accuracy that needs to be achieved, the available domain expertise, as well as the cost for the acquisition of the appropriate measurements.

### 6.7 The Mozilla Case

In this section, we discuss our experience in applying the method to Mozilla Thunderbird.

A preliminary step is to understand what aspect of the system each item actually configures. In Table 6.1, we attempt a categorization: although the majority of the items configure functional and behavioral aspects of the system, many of them are mainly either what we call structural data (e.g. Login Name, Server Address, Folder Name) or user interface appearance concerns (e.g. Background/Foreground Color).

Further, we try to identify which of the configuration items can be associated with user goals as advised in Steps I and II of our method. It turns out that there are four major categories of items according to their suitability to the construction of the goal models.

**Suitable.** These are configuration items that can be associated with a goal that is served by the functionality being configured. Moreover, goal decomposition that reflects alternative customization values is possible and each alternative has a different impact on generic properties of the system. Thus our method applies well in these items.
Figure 6.9: Configuration Goals for Mozilla Thunderbird
### Chapter 6. Goal-Based Software Configuration

<table>
<thead>
<tr>
<th>Aspect</th>
<th>#items</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>2</td>
</tr>
<tr>
<td>Function/Behavior</td>
<td>64</td>
</tr>
<tr>
<td>Input/Output Data Format/ Appearance</td>
<td>8</td>
</tr>
<tr>
<td>Interface Appearance</td>
<td>19</td>
</tr>
<tr>
<td>Structural Data</td>
<td>27</td>
</tr>
</tbody>
</table>

Table 6.1: Configuration items per configuration aspect

**Non-Intentional.** These are items information about the user and the system. An example is the e-mail address of the user. Our goal oriented method could not apply to such configuration items.

**Non-Trivial.** These are items that should relate to some user goals, but neither these goals nor the subgoals to which they are decomposed nor the impact to any generic properties can be trivially articulated. An example is the panel orientation in Thunderbird’s main screen. The user can select among three different orientation options. However, neither the intentions that lead to the selection of one of the three options nor how this influences general system qualities or generic user preferences can be expressed in a useful way.

**Unnecessary.** These are mainly configuration items for which it seems that only one value is meaningful; the purpose of the item is therefore challenged. For instance, the question whether the setting for the encoding of the incoming message should override the corresponding default setting (see Figure 6.1), should have a positive answer at all times; we could not find a case where the default settings of the client should have priority.

Notice that the result of the classification will depend on the subject that performs it. The authors attempted to classify the 120 items of this study into one of the above categories. If we combine the result with the categorization we produced in Table 6.1, some interesting observations can be made. Figure 6.10 illustrates the degree of suitability of our method with respect to the type of the items that are contained in the “Options” widget.

It is clear that the goal-oriented method can be applied when the configuration items are about functional and behavioral aspects of the system. However, items that constitute “structural data” cannot be associated with user intentions and cannot thus be approached using a goal-oriented method.
Interestingly, we would also fail to articulate goal descriptions for options that reflect the appearance of the user interface. However, there is no evidence that such options do not serve practical goals. For example, the selection of Serif over Sans-Serif fonts is known to facilitate ease of reading given that the font size and the resolution is adequate; Sans-serif fonts on the other hand are often characterized as “modern”. But such rules are not necessarily agreed among users, unless empirical studies have indicated so (and experts are available to confirm it). Furthermore, when such practical considerations meet the aesthetics of the users, the result is not predictable; one could claim that the interface appearance should be something that the user should better customize directly.

In Figure 6.9, 32 of the configuration items for Mozilla Thunderbird are analyzed through our method. Through the soft-goal analysis of Step IV (Figure 6.4) the soft-goals that can be used to configure the respective items are as many as 12. These are the soft-goals that do not contribute to any goals of higher level (i.e. they are “sinks” in the goal graph) and we can use them in the procedures described in Section 6.5. Observe that if we introduced additional functionality to Mozilla, this would necessarily increase the number of required configuration items. However, it is unlikely that the number of soft-goals would be influenced by this increment.

6.8 Summary

In this chapter we discussed goal-driven software configuration as a potential application of our framework for modeling and reasoning about goal variability. The central idea is that we can configure the numerous details of the system by only dealing with high level system-wide
user goals. We presented a method for mapping these goals to the low level configuration options. Given this mapping, the users only need to communicate their high level strategy of the use of the system; the configuration details will be automatically set to support this strategy. We believe that this will certainly benefit the vast majority of the users who don’t have (and don’t want to have) technical knowledge required for configuring software following today’s practices.

We are, of course, still not close to having a general and complete requirements driven framework to configuration. We saw that the goal oriented approach is suitable only for a subset of the configuration options, calling for an investigation to other requirements views that complement the intentional one, such as, for instance, entity views. In addition, the design of the appropriate preference elicitation interfaces remains an open challenge, especially in the case of common desktop applications, where users must be assumed to be lacking human support in specifying their goal preferences.

Nevertheless, we believe that our exploration with Mozilla clearly shows that there is a correspondence between goal alternatives and alternative software customizations, which strongly motivates our goal variability acquisition and reasoning framework.
Chapter 7

Conclusions

7.1 Summary and Reflections

7.1.1 Summary of Contributions

In this thesis, we presented a framework for acquiring and reasoning about problem variability using goal models. The framework is motivated by the need for more effective ways for identifying and exploring design alternatives (i.e. alternative solutions to given problems) during requirements analysis. Such alternatives’ identification and exploration is a natural part of the requirements analysis process in one-of development projects, as the several design ideas for fulfilling the stakeholder goals need to be evaluated and compared to fitness criteria the same stakeholders pose. More importantly, the process of identifying and exploring alternatives is necessary when analyzing requirements for customizable systems, that is systems that allow the fulfillment of a variety of stakeholder goals under a variety of individual preferences and environmental circumstances. Each of the identified requirements alternatives prescribes a different configuration of the customizable system, in a way that selection of a requirements alternative implies the selection of a software configuration. Thus alternatives analysis has the potential of becoming a user-centered problem-oriented software configuration facility.

The contribution of this thesis is a set of concepts, tools, and techniques for facilitating the process of identifying and reasoning about problem variability in requirements engineering. We use goal models as the main problem variability representation tool. Goal models allow concise description of large sets of alternative ways by which stakeholders can solve their high level problems. This is based on the existence of OR-decompositions of goals, which show how a goal can be fulfilled through the fulfillment of a set of alternative subgoals. We introduced
a categorization of such goal decompositions that allow their systematic identification during the goal development process. We also showed how it is possible to construct domain specific categorizations of OR-decompositions of goals through annotating text corpora taken from that domain. This way, analysts are provided with a systematic way to elicit problem variability and incorporate it in goal models, allowing identification of problem variant which would otherwise remain hidden in the domain.

The technique for facilitating the alternatives identification paves the way for the development of even larger spaces of alternatives, which in turn introduces the problem of exploring this space. Each goal alternative is a course of low level tasks that human and machine actors need to perform for the root goals to be fulfilled. One problem that arises is the size of the alternatives space and the inability of humans to be able to manually go through the potentially thousands of implied courses to find the one that best fits their needs. But what does fit means? Stakeholders prefer one course of tasks over another due to the fact that it matches better their higher level desires and preferences, as well as the particular circumstances under which they need to fulfill their goal. Thus, the second problem is interpretation of stakeholder priorities over high-level desires into priorities of low level operationalizations. We introduced a way by which stakeholders can create models of environmental factors that best describe their current circumstances, as well as specifications of their preferences over high-level desires. A tool reads these specifications and searches the space of alternatives for ones that best match the specified criteria. This way stakeholders can find the set of requirements that fits best their high-level needs.

This variability analysis technique can be useful for configuring customizable software, if there is a way to interpret courses of tasks into alternative configurations of a software system. This would allow software configuration via problem-oriented preference specification. In this thesis, we introduced an initial exploratory study of how this could be possible for configuring existing systems. We introduce a way to extract goal models from configuration screens of common desktop applications, and use the latter to manipulate the former. Our study shows that this is possible, but only to a certain degree leaving significant room for more investigation.

While we are hoping that this thesis serves as inspiration for a wide spectrum of future research endeavors, in this Chapter we will focus on three such dimensions for which we believe we have more information on how one could (or could not) proceed. We look at the issue of formally evaluating several parts of this research, the problem of developing customizable designs from high variability goal models, as well as the problem of increasing the expressiveness of our goal language.
7.1.2 Analysis of Principles, Assumptions and Limitations

The thesis is founded on certain general principles and assumptions on how certain problems in the area of requirements and variability identification, modeling and analysis could be approached. One is the enumerative approach to variability representation and binding that we have adopted. The second is the separation of problem and solutions, and how we conceptually structure the two. A third one is goal-orientation and its range of applicability. In this section we discuss why these principles are sound and/or well-served in the thesis and how they compare to alternative options.

Variability Representation and Binding

The approach of this thesis to variability representation and binding can be described as a pruning one. A pruning approach is based on the identification of large sets of concrete alternatives. These sets form the basis for further analysis. In our case, these sets are large sets of models (alternatives) that satisfy a theory (goal model). Variability binding rules out (prunes) alternatives that do not fit certain criteria. Work on feature modeling and product configuration follows this philosophy as well.

One may argue that enumeration of alternatives is problematic when applied in practice. Indeed one can claim that alternative identification is an expensive task, impossible during early lifecycle phases, and resulting in a very complex, therefore incomprehensible, high-variability model. A pure generative approach may be suggested instead, whereby variability is represented through abstract patterns that can be extended or refined to address individual problems in a case-by-case manner. In requirements, such a generative approach is represented by work on, for example, requirements reuse (e.g. [83] and [43]) and, from a point of view, on Problem Frames ([65]). Thus, abstract reusable requirements patterns can be extended and refined to provide requirements for a particular problem instance, without assuming a-priori enumeration of all such possible instances. It is just the selection and completion of the appropriate pattern that needs to be done.

Thus, in light of the generative possibility, some may claim that goal alternative enumeration just adds ill-motivated burden. But the generative approach may require effort too. Where are the patterns coming from and what is the process for exhaustively and correctly identifying them? How do we choose the right pattern and what guidance can we have in instantiating it? More importantly: how do we know that we have created the best instantiation if we dont envision a set of other alternatives to compare it with? Even in a generative approach we would
have to introduce a traceable gradual decision making process over refinements, which would, however, greatly resemble a hierarchical enumeration.

We, therefore, believe that enumeration of alternatives, though it is not necessary in general, is inevitable when analysis and decisions of options needs to be made. The more the alternatives that are available for the comparison, the most likely we will make a better decision. The practicalities and constraints of a particular real world application will of course define the boundaries. Hence, instead of adding a burden, this thesis suggests a tool for making an inevitable process more systematic and effective. Systematizing identification of goal variants as presented in Chapter 4 is one step towards cost-effective generation of alternatives and thus better support for the subsequent decision process.

Moreover, the resulting high-variability goal model may be criticized as complex and incomprehensible. But this can of course be said for any model that represents something complex and the same visualization practices that apply to any model may apply to goal models too (e.g. fragmentation and slicing). Furthermore, in our case, complexity needs to be seen in light of the opportunity to perform automated reasoning with our models. The preference-based reasoning techniques of Chapter 5, motivated exactly by the need for coping with the increasing complexity of models, illustrate this potential. In that chapter, the complexity of the high-variability goal model is hidden behind the specification of formulae that, in effect, specify characteristics of a desired slice of the model. Similar techniques can be envisioned for several other tasks such as for example analysis of correctness and consistency of alternatives. It is therefore our strong belief that high-level leverage of models through automated reasoning is a key to coping with their increasing complexity.

**Problem Variability**

We argued in the beginning of the thesis on the necessity for addressing variability at the problem level. Together with the modeling and reasoning techniques to do so, this thesis suggests a conceptual framework for understanding what constitutes problem and solution variability and how they are separated.

In particular, we saw that in goal models the road from the problem to the solution, that is from the abstract states of the world that stakeholders want to hold true to the concrete specifications of the machine is a continuum expressed through the decomposition structure. Indeed, high level goals, which constitute problems, are recursively decomposed into more concrete ones until at the leaf of the decomposition emerge concrete human and machine actions that
need to take place. These actions and their combinations actually constitute alternative solutions. Thus, in our framework, a problem is modeled as an abstract goal that needs to be decomposed, together with a set of facts about the domain and a set of preferences of the involved stakeholders. These domain facts and preferences are different for each domain and stakeholder instance and hence constitute the distinguishing factor among problem variants. Thus, we said that scheduling a formal meeting in a preferably quick way and scheduling an informal meeting in an office with an emphasis on reliability and privacy are two different variants of the same problem. The alternative solutions of the AND/OR tree constitute alternative solutions for a problem variant, each, however, satisfying the domain and stakeholder preferences by a different degree.

The validity and usefulness of this conceptual construction could be assessed through comparison to alternative proposals for conceptually structuring problem variability and associating it to solution variability at both a conceptual and a technical level. To our knowledge complete such proposals do not exist to date. Attempts in the area of Problem Frames were reported in Chapter 2. There, we observed that, apart from obvious abstract formulations (e.g. the core requirements and the “delta’s”), which are, not surprisingly, satisfied by the specific conceptual structure for problem variants we offer here, concrete techniques for systematically formalizing problem and solution variants and reason about them and their relationship are absent.

We therefore believe that the way goal models are extended and matched with preference specification, constitutes a sensible way to formulate problem variability. Furthermore, the interplay between problem specification and solution selection that occurs through reasoning about preferences (Chapter 5) only exemplifies in our framework the clear separation of problem and solution as opposite ends of a continuum.

**Goal Orientation and Non-intentional Variability**

Our choice of goal models as fundamental concept for identifying and reasoning about variability is motivated by several factors. Firstly, as we also discussed above, goal models make it possible to associate problems to solutions. Secondly, the rich and intuitive language they provide is amenable to formalizations and automated processing. Goal-orientation assumes that goals can be (one of) the foundation(s) for eliciting requirements and play a role similar to that of objects and use-cases in Object Oriented Analysis ([19]). However, unlike the object-oriented requirements paradigm that starts with what functions need to exist, the goal-oriented approach starts with *why* is anything pertaining to objects and functionality needed,
turning thereby the focus from the functions of the solution to the problem. This way, the focus on goals may help identification of sets of functional specifications that are more likely to be sound and complete.

However, at least in terms of variability and within the scope of this thesis, completeness can be hypothesized only with respect to the discourse of things that emerge through analysis of stakeholder goals. The variability identification procedure we offer in this thesis does not help us identify variability that is independent of stakeholder intentions. Thus, the domain facts we used throughout the thesis to model contextual conditions and their variability did not become a subject of systematic identification. They were instead introduced in an ad-hoc way when they were needed as conditions or effects for our goals. In addition, we have strong evidence that there exists software variability that cannot be associated with stakeholder intentions. Indeed, in Chapter 6, our attempt to connect goals with configuration options was only partially successful, as many of the configuration options would not intuitively relate to goal variability. Not surprisingly, those options reflected structural aspects of the system and the context. Therefore, we should not infer that goal variability, at least the way it is understood and treated in this thesis, covers every variability aspect of the problem.

However, it is unknown whether there are requirements analysis paradigms other than goal-orientation that could systematically deal with problem variability in a holistic sense. To date, problem variability was not even distinguished from solution variability, while, in general, variability identification was an ad-hoc process and variability analysis was ill-supported. Our thesis certainly offers a starting point for dealing with problem variability and presents strong evidence of applicability within its premises.

### 7.2 Towards An Evaluation Programme

Our intention in this thesis was to focus on the conceptual, theoretical and technical challenges in developing tools and methods for identifying and reasoning about goal variability. From here, the application of these techniques needs to be better understood in practice. Evaluation work for this thesis should be centered around two axes, described below.

#### 7.2.1 Variability-Intensive Goal Decomposition

Our technique for developing high-variability goal models offers a systematic way of thinking about and incorporating alternatives in goal models. Our informal exploration demonstrated
that the technique allows discovery of alternatives that would otherwise remain hidden. However this particular effect is yet to be formally proved.

The seemingly straightforward experimental evaluation for showing this property, involves a series of challenging design and conceptual issues. The identified goal alternatives do not come from a set of universally “true” alternatives, but reflect the analysts and stakeholders view of the domain, expressed in their own terms. Different analysts or stakeholders have a different way to divide their (different) knowledge about their intentions and domain into elementary goals and tasks. Thus, each viewpoint is different from the others in ways that may be other than subset relationships amongst set of alternatives. For instance, an alternative identified by one stakeholder may have been identified by some other stakeholder only partially. Compare the goals Invite Participants by E-mail and Secretary Send Message to Participants, coming from two different analysts analyzing the same problem. Do they mean the same thing? Is a “Message” intended to mean “E-mail”? Does the first analyst also imply the “Secretary”, or some similar (but not identical) role, or she didn’t think of the actor? What is the difference between “sending a message” and “inviting by e-mail”; the latter seems to refer to a more interactive (and therefore different) process. Obviously, quantifying the amount of information contained in each goal model cannot be objective, which in turn does not allow a trivial assessment of “how much” variability an experimental participant has identified compared to another.

In addition, even if the quantity of variability in goal models can be measured, meaningfulness and usefulness, that is the quality of each variant also needs to be assessed. If the concern-driven decomposition allows the identification of more alternatives, are these alternatives important or otherwise useful to the stakeholders?

Finally from a different point of view, the goal oriented approach for identifying variability needs to be compared to existing feature-based approaches for variability identification exercised for analyzing domains for product lines. How does goal-oriented variability identification support the domain analysis process in a way that the latter results in a better and richer software configurability aspect? For this question, a longitudinal case study would allow understanding the impact of applying goal-oriented variability analysis techniques on the whole development lifecycle of the product line(s).
7.2.2 Preference Elicitation

The preference specification framework we introduced needs to be empirically investigated at two different levels. One level is the language. The specification formalism we presented is based on LTL, which should be understood as inaccessible to common users, without resorting to LTL patterns, which are, however, less expressive. But can even pattern-based LTL desire specification be used by ordinary users and how? Is there a need for an additional interaction layer for supporting the construction of preferences and what is it? An experimental design on the effectiveness of alternative front-end languages and interfaces for eliciting preferences introduced the challenge of understanding whether the resulting desires is what the stakeholders actually wanted. In parallel, the use of numbers for weighting individual desires introduces an additional question. We proposed AHP for eliciting these numbers. However AHP has been used for prioritizing general software features rather than qualities and behavioral properties. Does this change in the nature of the subjects of comparison make AHP less effective?

The second level of empirical investigation is the preference elicitation per se, and its position in the software lifecycle. One can argue that most users should not be able to prioritize over high-level qualities in general and without a context. For example, prioritizing over Privacy and Profitability depends on the situation; many users would occasionally trade one for the other, without however being able to make a general statement on which one they would prefer over the other. So is it sensible to separate between long-term (default) and ad-hoc preferences? An exploratory empirical investigation would shed more light on the role that the context plays to the formation of stakeholder preferences. This understanding is necessary for obtaining a picture of how effective preference elicitation and preference-based configuration tools can be built in practice.

7.3 From Goal Variability to Design Variability

The value of the variability identification and exploration framework we presented would multiply if goal variability could be mapped to software variability in a systematic way. While we presented evidence of the connection between goal alternatives and configuration options of existing systems, how goal variability can be translated in design variability of a new system remains an open question. Every alternative of the problem should be addressed by a distinct configuration of the software system, while the software system itself should not imply configurations that are not somehow connected with configurations of the problem (stakeholder
intentions and environmental properties). Therefore goal models should play an important role in constructing the configurability aspect of software systems.

A hypothesis we have been testing in the context of exploring goal-driven development of software configurability is the *isomorphism* between the goal structure and the design structure. According to this hypothesis the hierarchical structure of goal models, can be translated into hierarchies of software components whereby each goal and link among goals maps to a component and a link between components, in a way that the two structures are the same. An example of such a correspondence is seen in Figure 7.1. In the figure, the goal hierarchy is translated into a control hierarchy of components. For every goal there is a component that is responsible for supporting its fulfillment. Higher level goals map to “super-components” containing other more “concrete” components that correspond to lower level goals. Special connectors called “switches” (introduced in [118]) allow choice in terms of where an interface call is directed to. Thus, the component model is designed in a way that every alternative of the goal model reflects a design alternative, through configuring the “switches” appropriately.

Figure 7.1: From Goals to Software Components

This, however, constitutes only a schematic solution to the question. Obviously, the structure of software components does not have to obey the hierarchical structure of goal models. The former reflects technical necessities such as performance, maintainability or reliability, whereas the latter reflects the way the analyst has organized the problem elements in her mind and the dynamics of her interaction with the stakeholders or the exploration of other sources of information. In addition, the choice of control flow as the subject of isomorphism with the goal
models is arbitrary: is data flow or specialization structure also isomorphic to the goal structure? Clearly, the control isomorphism assumption alone is arbitrary. The mapping between the goal variability space and the software configuration spaces can perhaps be established without having to rely to any isomorphism whatsoever.

Nevertheless, it is still meaningful to assume that the goal structure is the structure that best communicates configurability. This is particularly interesting in product derivation in product-lines, which may not be a process of simple binding of variation points, but one of creating derivation and “gluing” scripts over fully or partially implemented software modules. A typical example is object-oriented frameworks, in which a set of abstract classes is reused for creating individual members of a product-line. Allowing the structure of the reusable matter (precisely: its derivation “facade” - [66]) to reflect that of the goal models can arguably facilitate the derivation process, in that it would allow developers think in terms of the problem they are trying to solve.

7.4 Enriching the Goal Modeling Notation

The last topic for future consideration we will discuss is the limitations of our visual goal modeling language and potential improvements. Our experience with developing a great number of models on several domains (Meeting Scheduling, Messaging, ATM and Nursing to name a few) in the context of this thesis, revealed the need for two potential extensions which would greatly increase the expressiveness of the language and facilitate the modeling process.

One opportunity is the introduction of optional goals. In feature models an optional sub-feature is one that may or may not be included in a featureset if its super-feature is included, too. In goal models, such facility does not exist, and analysts are forced to introduce a special, rather “inelegant”, type of OR-decompositions. In Figure 7.2 part of the decomposition of the goal Send A Meeting Invitation is shown, in which the sender may or may not want to add a digital signature. While the left hand side shows the current way for modeling the situation, which is admittedly unnatural, the right hand side of the Figure shows how this could be written more naturally through an optional subgoal. As in feature models, the optional subgoal is recognized by the cyclic decoration of the respective arrow. Notice, however, that in order to maintain the same level of expressiveness, we need to introduce a “trigger” link which enforces the selection of the optional goal if a condition is true. In the diagram we use the link \( \xrightarrow{\text{tri}} \) for the purpose.

The second extension that would significantly increase the expressiveness of goal models
is cardinalities. The need for cardinalities arises from the fact that goals may be decomposed into an arbitrarily large amount of AND-subgoals. For example a Secretary has achieved the goal **Confirm Participants** only if every single potential participant has received an invitation and responded to it. The number of participants is, of course, unknown. In Figure 7.3, we show the use of annotations for abbreviating the potentially great number of AND-subgoals. We are using the syntax `<op> [var] in [Set]`, where `<op>` is one of `*` (meaning “for all”), `at-least(num)`, `at-most(num)` and `between(min,max)`, with the obvious meaning. Thus, `between(2,5) m in g.members` means, “any number of members of `g.members` that is greater or equal to 2 and less or equal than to 5, each being further referred to as `m`”. In the figure, the Secretary is responsible for scheduling a number of meetings for each group. Notice how the addition of repetitions directly turns the attention of the goal modeler to the entity model of the domain.

Obviously the extension of the language is not a trivial process and involves careful consideration of, for example, the implications that arise from combining the new elements with the existing ones. Repetition in combination with contribution links to soft-goals is an example of a potential challenge. Nevertheless, increasing the expressiveness of goal models will make them significantly more useful in showing in more detail how abstract state of affairs the
stakeholders want to achieve relate to specific operations in the domain.

7.5 Summary

In this chapter we once again discussed the motivation behind this thesis, briefly summarized each contribution and offered a critique of its basic assumptions and principles. Then we presented three of the areas for future work: we explored potential ideas for empirical and experimental evaluation, discussed the challenge of producing high-variability designs from goal models and presented some ideas for extending the goal modeling language. We believe that pursuing work in either direction would significantly contribute to the interests of our research community.
Bibliography


[37] Erich Gamma and Richard Helm and Ralph Johnson and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.


