

# Reusable Goal Models

Mustafa Berk Duran

A thesis submitted to McGill University  
in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

Department of Electrical & Computer Engineering

McGill University  
Montréal, Canada

November 19, 2018

© Mustafa Berk Duran 2018

# Abstract

Goal models help elicit, specify, analyze, and validate requirements as they capture hierarchical representations of system requirements, possible solutions, stakeholder objectives, and their relationships. In the early requirements phase, goal models aid requirements engineers in understanding the goals of stakeholders and exploring solution alternatives based on their impact on these goals. Despite these strengths of goal modeling, reuse of goal models has received limited attention in the goal modeling community for various reasons. We argue that the requirements that need to be fulfilled at least for the reuse of goal models are as follows: (i) ensure that trade-off reasoning via goal model evaluation is possible through the reuse hierarchy, (ii) provide the means to delay decisions to a later point in the reuse hierarchy when more complete information is available, while still allowing the reusable artifact to be analyzed, (iii) take into account constraints imposed by other modeling notations when evaluating reusable goal models, (iv) allow context dependent information to be modeled so that the goal model can be used and analyzed in various application contexts, and (v) last but not the least, provide a well-defined interface for reuse. This thesis argues that existing goal modeling approaches do not fully address these five required capabilities for reusability. To address this issue, this thesis introduces novel goal model evaluation mechanisms for bottom-up and top-down evaluation based on a lazy, recursive algorithm. The novel mechanisms allow the design, modularization, evaluation, and analysis of reusable goal models in reuse hierarchies in the presence of external constraints taking delayed decisions and contextual information into account. Required metamodel extensions are discussed and tool support demonstrates feasibility.

# Abrégé

Les modèles orientés buts permettent d’obtenir, de spécifier, d’analyser et de valider des exigences parce qu’ils décrivent les représentations hiérarchiques des exigences système, des solutions possibles, des objectifs des parties prenantes et leurs relations. Lors de la phase initiale des exigences, les modèles orientés buts aident les ingénieurs en exigences à comprendre les buts des parties prenantes et à explorer des solutions alternatives en fonction de leur impact sur ces buts. Malgré ces avantages de la modélisation orientée buts, la réutilisation des modèles de buts a reçu peu d’attention dans la communauté de modélisation de buts pour diverses raisons. Nous soutenons que les exigences minimales à satisfaire pour la réutilisation des modèles de buts sont les suivantes: (i) assurer que l’analyse de compromis via l’évaluation du modèle orienté buts soit possible via la hiérarchie de réutilisation, (ii) fournir les moyens de reporter les décisions à un point ultérieur de la hiérarchie de réutilisation lorsque des informations plus complètes sont disponibles, tout en permettant à l’artefact réutilisable d’être analysé; (iii) tenir compte des contraintes imposées par d’autres notations de modélisation lors de l’évaluation de modèles orientés buts réutilisables; (iv) permettre à l’information dépendante du contexte d’être modélisée de manière à ce que le modèle orienté buts puisse être utilisé et analysé dans divers contextes d’application; et (v) enfin et surtout, offrir une interface bien définie pour la réutilisation. Cette thèse soutient que les approches de modélisation orientée buts existantes ne répondent pas entièrement à ces cinq capacités requises pour la réutilisabilité. Pour résoudre ce problème, cette thèse présente de nouveaux mécanismes d’évaluation de modèles orientés buts pour les évaluations ascendante et descendante basées sur un algorithme récursif paresseux. Ces nouveaux mécanismes permettent la

conception, la modularisation, l'évaluation et l'analyse de modèles orientés buts réutilisables dans les hiérarchies de réutilisation en présence de contraintes externes prenant en compte les décisions différées et les informations contextuelles. Les extensions de métamodèle requises sont discutées et les outils de support démontrent la faisabilité de l'approche.

# Acknowledgements

I would like to express the deepest appreciation to my supervisor Gunter Mussbacher for his support, understanding and guidance throughout this doctoral research. Without his supervision, patience, and encouragement, this thesis would not have been completed. Members of my committee, Jörg Kienzle, Michael Rabbat, and Jin Guo, my internal examiner Clark Verbrugge, and my external examiner Luiz Marcio Cysneiros, are thanked for accepting to review this work and for their valuable comments and feedback.

Over the last years, I was fortunate to be a part of the Software Engineering lab at McGill and share the beautiful working space with dearest friends and colleagues and have wonderful discussions and collaborations with them. First and foremost, I would like to thank the highly-talented Matthias Schöttle for both his friendship and his thorough project management and professionalism that made my proof-of-concept implementations in the TouchCORE tool go smoothly. I would like to express my gratitude to Omar Alam, Romain Alexandre, Hyacinth Ali, Aprajita, Céline Bensoussan, Cécile Camillieri, Yanis Hattab, Ruchika Kumar, Rijul Saini, Nishanth Thimmegowda, Rohit Verma, and many more for their precious feedback and help, and for welcoming me to their laboratory.

I would like to express my special thanks to my friends and family, for their love, trust, understanding and all kinds of support not only throughout my thesis but also throughout my life. They (hopefully) never resented me when I rejected their plans to hang out as I needed to *work on my thesis*.

Finally, Elçin, thank you for your constant and tireless support, encouragement, patience, and understanding of usually long working hours towards the completion of this thesis.

# Table of Contents

<b>Abstract</b> . . . . .	ii
<b>Abrégé</b> . . . . .	iii
<b>Acknowledgements</b> . . . . .	v
<b>Table of Contents</b> . . . . .	vi
<b>List of Tables</b> . . . . .	x
<b>List of Algorithms</b> . . . . .	xi
<b>List of Figures</b> . . . . .	xii
<b>List of Acronyms</b> . . . . .	xiv
<b>1 Introduction</b> . . . . .	1
1.1 Motivation . . . . .	1
1.2 Statement of the Problem . . . . .	3
1.3 Research Hypotheses . . . . .	8
1.4 Methodology . . . . .	9
1.5 Thesis Contributions . . . . .	9
1.6 Publications and Presentations . . . . .	12

## *Table of Contents*

---

1.7	Formatting Conventions . . . . .	15
1.8	Thesis Outline . . . . .	15
<b>2</b>	<b>Background . . . . .</b>	<b>16</b>
2.1	Goal Modeling . . . . .	16
2.2	Analysis and Reasoning with Goal Models . . . . .	18
2.2.1	NFR Framework . . . . .	23
2.2.2	i* . . . . .	24
2.2.3	GRL . . . . .	25
2.2.4	Tropos . . . . .	26
2.2.5	KAOS . . . . .	27
2.3	Summary . . . . .	28
<b>3</b>	<b>State of the Art: Systematic Literature Review . . . . .</b>	<b>29</b>
3.1	Methodology . . . . .	30
3.1.1	Research Questions . . . . .	30
3.1.2	Selection of Publications . . . . .	31
3.1.3	Threats to Validity . . . . .	39
3.2	Results and Analysis . . . . .	42
3.2.1	Summary of Previous Reviews . . . . .	43
3.2.2	Analysis of the Approaches . . . . .	46
3.2.3	Future Research Themes . . . . .	59
3.3	Summary . . . . .	61
<b>4</b>	<b>Relative Contribution Values . . . . .</b>	<b>63</b>
4.1	Contribution Values in Goal Modeling . . . . .	63
4.1.1	Qualitative Labels . . . . .	65

## Table of Contents

---

4.1.2	Real-life Measurements . . . . .	65
4.1.3	Quantitative Values . . . . .	66
4.2	The Need for Relative Contribution Values . . . . .	66
4.3	Normalization of Goal Satisfaction . . . . .	70
4.3.1	Forward Propagation with Normalized Satisfaction . . . . .	71
4.3.2	Scale Factor and Offset Values for Normalization . . . . .	72
4.4	Summary . . . . .	74
<b>5</b>	<b>Reusable Goal Models with Constraints . . . . .</b>	<b>75</b>
5.1	Constraints and Goal Model Analysis . . . . .	76
5.2	Internal Constraints . . . . .	77
5.2.1	The Lazy, Recursive Normalization Algorithm . . . . .	78
5.3	Feature Model Constraints . . . . .	81
5.3.1	Feature Models and Goal Modeling . . . . .	84
5.3.2	Changes to the Evaluation Algorithm for Feature Model Constraints . . . . .	85
5.4	Workflow Model Constraints . . . . .	88
5.4.1	Workflow Models and Goal Modeling . . . . .	89
5.4.2	The Improved Evaluation Algorithm for Workflow Model Constraints . . . . .	93
5.5	Proof-of-Concept Implementations . . . . .	106
5.5.1	Implementation Workflow . . . . .	106
5.5.2	Design of the Test Cases . . . . .	113
5.5.3	Performance Evaluation Results . . . . .	115
5.6	Summary . . . . .	119
<b>6</b>	<b>Goal Models in Reuse Hierarchies . . . . .</b>	<b>121</b>
6.1	Reuse Hierarchies with Concern-Oriented . . . . .	121



## Table of Contents

---

6.2	Reuse Interface . . . . .	124
6.3	Goal Model Evaluation in the Reuse Hierarchy . . . . .	134
6.4	Handling Delayed Decisions . . . . .	140
6.4.1	Range Evaluation in Goal Model Reuse Hierarchies . . . . .	142
6.4.2	Performance Discussion and Tool Support . . . . .	154
6.5	Top-down Evaluation of Reusable Goal Models . . . . .	155
6.5.1	Goal Prioritization with Thresholds . . . . .	155
6.5.2	The Lazy Recursive Algorithm in Top-down Evaluation . . . . .	157
6.5.3	Top-down Evaluation in the Reuse Hierarchy . . . . .	163
6.5.4	Proof-of-Concept Implementation . . . . .	171
6.6	Contextual Information . . . . .	174
6.7	Summary . . . . .	177
<b>7</b>	<b>Conclusions . . . . .</b>	<b>180</b>
7.1	Contributions . . . . .	180
7.2	Future Work and Research Directions . . . . .	183
	<b>Bibliography . . . . .</b>	<b>186</b>

# List of Tables

3.1	Search Syntax Used per Search Engine . . . . .	32
3.2	Initial Search Results . . . . .	33
3.3	Second Phase Categorizations for Goal Model Reuse . . . . .	37
3.4	Second Phase Categorizations for Contextual Goal Models . . . . .	38
3.5	Summary of Goal Model Reuse Approaches . . . . .	47
3.6	Summary of Contextual Goal Modeling Approaches . . . . .	48
3.7	Summary of Contextual Goal Modeling Approaches (cont.d) . . . . .	49
3.8	Evaluation Rubric for R.1 . . . . .	51
3.9	Evaluation Rubric for R.2 . . . . .	52
3.10	Evaluation Rubric for R.3 . . . . .	53
3.11	Evaluation Rubric for R.4 . . . . .	55
3.12	Evaluation Rubric for R.5 . . . . .	58
3.13	Evaluation Rubric for Tool Support . . . . .	59
5.1	Performance Results for Scale Factor and Offset Calculation . . . . .	116
6.1	Performance Results for Top-down Evaluation . . . . .	173
7.1	Analysis of our approach with respect to the SLR evaluation criteria . . . . .	181

# List of Algorithms

5.1	Lazy, Recursive Algorithm for Scale Factor and Offset Calculation . . . . .	82
5.2	Lazy, Recursive Algorithm for Scale Factor and Offset Calculation (cont.d) .	83
5.3	Core Path Extraction Algorithm [24] . . . . .	99
6.1	Evaluation Mechanism for Goal Models in Concern Hierarchies [29] . . . . .	139
6.2	Surrogate Importance Calculation Algorithm . . . . .	170

# List of Figures

2.1	GRL notation with qualitative contribution and satisfactions . . . . .	17
2.2	Contribution and satisfaction values in goal models [24] . . . . .	20
2.3	Legend for software interdependency graphs of the NFR Framework . . . . .	23
2.4	Excerpt of the i* notation . . . . .	24
2.5	Excerpt of the Tropos notation . . . . .	26
2.6	Excerpt of the KAOS framework constructs for goal modeling . . . . .	27
3.1	Summary of the review process . . . . .	39
4.1	Global and relative contribution values in reusable goal models [24] . . . . .	67
5.1	Example execution of the lazy, recursive normalization algorithm for maximum [28] . . . . .	79
5.2	Excerpts of feature and goal models of a reusable artifact “Authentication” .	84
5.3	Contributions in feature model (Parent A’s children) for minimum [28] . . .	87
5.4	Excerpts of feature, goal, and workflow models of “Authentication” [24] . . .	90
5.5	Steps of path extraction [24] . . . . .	97
5.6	Optimistic option for the evaluation algorithm [24] . . . . .	103
5.7	Implementation workflow [24] . . . . .	107
5.8	Use Case Map for “TC7 + F + R” in the jUCMNav tool [24] . . . . .	109
5.9	Feature Model for “TC7 + F + R” in the TouchCORE tool [24] . . . . .	110

5.10	Goal Model evaluation for “TC7 + F + R” in the TouchCORE tool [24] . . .	112
5.11	Evaluation of a single goal in the presence of run-time constraints [24] . . . .	113
6.1	Reuse hierarchy in feature (top) and goal (middle and bottom) models [29] .	126
6.2	Extract of GRL Metamodel treating feature model concepts shown in orange as a special case of goal model concepts [225] and extended with reuse concepts shown in yellow [29] . . . . .	132
6.3	Feature Model (left) and evaluated Goal Model with reused strategies (right) in jUCMNav [29] . . . . .	133
6.4	Evaluation of Goal Models in a concern hierarchy [29] . . . . .	135
6.5	Concern hierarchy of the Bank application [30] . . . . .	142
6.6	Variation interface of the Authentication concern [30] . . . . .	143
6.7	Authorization concern reusing Authentication [30] . . . . .	145
6.8	Bank concern reusing Authorization [30] . . . . .	150
6.9	Range evaluation flowchart [30] . . . . .	151
6.10	Top-down evaluation on a single-level reusable Goal Model [32] . . . . .	159
6.11	Bank application reuse hierarchy with surrogate actors and importances [32]	165

# List of Acronyms

CARGO Context-aware Reasoning Using Goal-orientation

CORE Concern-Oriented Reuse

CPA Context-aware Personal Agents

DGM Design Goal Model

DSPL Dynamic Software Product Lines

ERD Entity Relation Diagram

GBRAM Goal-Based Requirements Analysis Method

GORE Goal-Oriented Requirements Engineering

GRL Goal-oriented Requirement Language

KPI Key Performance Indicator

MDE Model Driven Engineering

MDSD Model-driven software development

NFR Nonfunctional Requirement

RAM Reusable Aspect Models

## *List of Acronyms*

---

RE	Requirements Engineering
SAS	Self-Adaptive Systems
SoC	Separation of Concerns
SPL	Software Product Line
UCM	Use Case Map
UML	Unified Modeling Language
URN	User Requirements Notation

# Chapter 1

## Introduction

This chapter, summarizes the motivation for this research, identifies the problem and the research question as well as the requirements that need to be satisfied to tackle the problem, defines the research hypotheses and methodology, lists the contributions of the thesis and publications based on this research, and outlines the content of the thesis.

### 1.1 Motivation

Requirements Engineering (RE) is defined as a branch of software engineering involved with the goals for, functions of, and constraints on software systems, as well as the relationship of these components to behavioral specifications of software, their evolution across software families and over time [1]. On the other hand, the objective of model-driven software development (MDSD) is to capture important aspects of a software system by means of suitable models, as models can capture stakeholder intentions more directly in comparison to implementation code, are at a higher abstraction level, and less complicated to analyze [2].

Given these definitions of RE and MDSD, it is not surprising to find goal modeling in the intersection of the two disciplines as one of the most commonly used requirements engineering models. As opposed to other traditional RE techniques, goal modeling incorporates non-functional requirements into modeling and specification of the software and allows representation and assessment of alternative system configurations [3].

Requirements engineers frequently make use of goal-oriented modeling for elicitation



and elaboration of system requirements and to deal with conflicting stakeholder goals [4]. Goal models allow requirements engineers to represent the hierarchical relationships among stakeholder objectives, qualities of interest, functional and non-functional requirements, and their potential solutions. Goal modeling is important for abstracting away the unnecessary information (e.g., implementation details) to help stakeholders understand the requirements and communicate their needs and expectations.

Various goal modeling formalisms exist in the literature with similarities in their concepts and analysis capabilities. The NFR framework [5],  $i^*$  [6], the Goal-oriented Requirement Language (GRL) [7], Tropos [8], and KAOS [9] all have their own way of representing the concepts of intentional elements (e.g., goals and softgoals) and the relationships among them (e.g., contribution, decomposition, or dependency links). Furthermore, different formalisms make use of different means for trade-off analysis. The trade-off analysis and reasoning capabilities provided in various goal modeling formalisms can guide engineers in decision-making processes such as prioritization of requirements, resolution of conflicting stakeholder goals, and evaluation of alternative solutions.

Reuse is defined as the process of creating software systems from existing software artifacts instead of creating them from scratch [10] and as the use of existing artifacts in a new context [11]. Software reuse is powerful due to the potential benefits it provides such as increased quality, productivity, and reliability with faster time-to-market and lower cost.

Software reuse improves productivity regardless of the language paradigm used [12] and while goal models can be designed as concrete artifacts for a specific project, they can also be designed as generic, reusable artifacts. Similar to any reusable software artifact, potential benefits of model reuse, such as increased quality and productivity [13][14], also apply to the reuse of goal models. As each goal model encapsulates its modelers' knowledge and expertise, it is critical to take advantage of the strengths of goal modeling in the context of reuse.

However, reuse of goal models [5][6][7][15] has received limited attention in the goal modeling community and is mostly confined to the idea of goal catalogues (e.g., security, reliability, etc.) [16], which may be imported as is into the goal model of an application under development, thus indicating the need for a set of capabilities and tools to design and reuse generic goal models without sacrificing their strengths in analysis and representation.

In the context of reuse, goal models may be used either as standalone artifacts to help better understand stakeholder objectives and compare potential solutions in the early requirements phase or as connected artifacts to describe and compare reusable artifacts.

**Standalone artifacts** are used to describe a problem situation and can be reused in different contexts by themselves while maintaining the provided analysis capabilities to better understand the needs of different stakeholders and considered candidate solutions.

**Connected artifacts** are used in collaboration with other models to describe reusable artifacts and to express the impacts of the reusable artifact on high-level goals and qualities. In this situation, goal models can answer why a reusable artifact should be chosen over another candidate artifact as they allow alternative candidate solutions to be evaluated with trade-off analysis, i.e., goal models can guide a modeler through design choices among alternative reusable artifacts and solutions.

A comprehensive goal modeling language should support reuse of both standalone and connected goal models.

## 1.2 Statement of the Problem

Even though goal modeling can benefit from reuse to increase quality, productivity, and reliability, to reduce time-to-market, and to lower cost, goal model reuse has received limited

attention in the goal modeling community. Hence, the main research question to answer is:

*How can goal models be made reusable?*

Answering this question involves determining the goal model concepts and algorithms needed to reuse a goal model and benefit from the strengths of goal modeling [17].

As one of the initial and stepping stone contributions of this research, the requirements that need to be fulfilled at least for the reuse of goal models are identified and introduced in the remainder of this section.

The requirements were identified based on experience with work on requirements reuse [18], Reusable Aspect Models (RAM) [19], and Concern-Oriented [20]. The requirements are needed for these application contexts, but we cannot guarantee that other requirements may exist for other contexts. The requirements are used both in the assessment of the current state of goal modeling literature and as guidelines for a comprehensive solution for reusable goal models.

**R.1.** *A reusable goal modeling approach shall ensure that trade-off reasoning via goal model evaluation is possible through the reuse hierarchy.*

Reuse of goal models, as for any other reusable artifact, results in model hierarchies where lower-level reusable artifacts are composed together to build larger, higher-level artifacts. Analysis and validation that is available for a single model at a single level must be ensured to be available through the entire reuse hierarchy, from small, low-level artifacts to large, system-level artifacts, because the ultimate objective of goal modeling is to provide support for some form of reasoning for RE subprocesses [21] (e.g., requirements elaboration, trade-off analysis, evolution management, consistency checking, etc.).

For example, in the context of GRL, analysis in the form of trade-off reasoning is performed by a propagation-based evaluation mechanism that takes the initial satisfaction value

of a child element and propagates it up to the parent according to the type and value of the link between them [22] (e.g., a parent node's satisfaction value is calculated as the weighted sum of its inbound contribution links -with assigned contribution values- and percentile satisfaction values of the corresponding child nodes). Furthermore, the complexity of the evaluation of the composed models must also be managed with a growing reuse hierarchy for the evaluation to remain feasible.

Therefore, to support the trade-off reasoning through the reuse hierarchy, the way contribution values are assigned and satisfaction values are interpreted by different modelers working on different levels in the reuse hierarchy needs to be investigated as well as the means to cope with the complexity of the evaluation.

**R.2.** *A reusable goal modeling approach shall provide the means to delay decisions to a later point in the reuse hierarchy when more complete information is available, while still allowing the reusable artifact to be analyzed.*

The exact requirements for a reusable artifact are typically unknown as the context in which the artifact is going to be used is unknown in advance [23].

In general, two reuse scenarios are applicable to goal models. In the first scenario, the modeler has all the information about the system, the context in which the reusable artifact is going to operate, and its complete requirements. If that is the case, the modeler can pick and reuse the most appropriate reusable artifact.

In the second scenario, the modeler lacks some of the information, however, it is still important to allow them to proceed with the limited information at hand. This requires the modeling approach to support the partial reuse of a reusable artifact and leaving some decisions open to be made at the later stages of development.

Essentially, a goal model describes a set of choices that are evaluated with respect to objectives of stakeholders. The first reuse scenario allows decisions to be made for each choice,

whereas, the second reuse scenario leaves some choices to be made when more information is available higher up in the reuse hierarchy.

**R.3.** *A reusable goal modeling approach shall take into account constraints imposed by other modeling notations when evaluating reusable goal models.*

Goal models cannot always be considered in isolation as additional constraints on various goal model elements may be expressed in other modeling notations that are used in collaboration with goal models.

Typically, external constraints exist for tasks in a goal model, i.e., the candidate solutions. For example, causal relationships between tasks may be expressed with a workflow model, while a feature model may describe which tasks can be selected together to build a functionality or a system [24]. Ignoring these constraints during goal model analysis may result in an incorrect evaluation (e.g., two tasks that cannot be selected together according to the feature model should not be evaluated together in the goal model). Trying to model all of the aforementioned information in the goal model results in an overly complex goal modeling notation that is difficult to analyze and understand.

Consequently, different requirements models are used in collaboration with goal models to benefit the modeler as each notation has their own strengths. It is also important to note that such additional constraints make the evaluation of goal models in reuse hierarchies (see R.1) much more challenging as the analysis requires backtracking or other forms of non-linear exploration of the goal model rather than employing the usual straightforward propagation algorithms [24].

**R.4.** *A reusable goal modeling approach shall allow context dependent information to be modeled so that the goal model can be used and analyzed in various application contexts.*

Designing a generic software artifact has its challenges. Although it requires considering a wide range of scenarios in which the artifact could be reused, it often results in the use of

placeholders instead of concrete model elements to represent unknown application-specific information to keep the reusable artifact generic.

Therefore, it is important to allow elements of a goal model to adjust to different circumstances. For example, depending on the application context, a candidate solution may contribute to a high-level goal either positively or negatively or with varying degrees of impact.

**R.5.** *A reusable goal modeling approach shall provide a well-defined interface for reuse.*

Many successful reusable artifacts have clearly defined interfaces to increase encapsulation and modularity, and strengthen the reuse hierarchy [25][26]. Goal models are no exception, i.e., a reuse interface must also be defined for goal models. This interface would allow hiding lower-level goal model elements behind the interface of the current artifact from the next higher-level artifact that wants to reuse this current artifact.

A clearly defined reuse interface allows modelers to differentiate model elements of the reused lower-level artifact from those of the reusing artifact both at the model and the metamodel levels.

To summarize, an approach that can answer the main research question must address these five requirements. While we cannot claim that these requirements are sufficient for all reuse contexts, they are nevertheless necessary to make goal models reusable from our experience of applying goal models in a reuse context.

A reusable goal modeling approach should support the reuse of both standalone and connected goal models. For the reuse of a standalone goal model, the modeler realizes that the current stakeholders, their objectives, and potential solutions are similar to previous ones, and hence the standalone goal model can be reused during the early requirements phase. In this case, all requirements apply except for R.3, i.e., external constraints do not need to be taken into account as the goal model is used in isolation.

For the reuse of a connected goal model, the modeler realizes that some properties of reusable artifacts apply to the current problem situation. These reusable artifacts do not necessarily have to be requirements-level artifacts but can also describe system design or implementation. The modeler then uses the connected goal models to decide which reusable artifact to reuse. In this case, all five requirements apply, because the goal model is used in connection with other models which may impose additional constraints on the goal model.

The hypotheses for this research are built upon these requirements and explained in the next section.

## 1.3 Research Hypotheses

The goal of this research is to bridge the gap between goal modeling and reusability through identification of the challenges and obstacles that hinder reuse of goal models as well as the introduction of feasible and scalable methods and tools to overcome the shortcomings of existing goal modeling approaches.

Given the main research question and the five requirements identified in the previous section, the hypotheses for this research are stated as follows:

**Hypothesis 1.** Current goal modeling technologies do not fully satisfy the five requirements R.1 to R.5: ensuring a viable analysis and validation capability through the reuse hierarchy, dealing with delayed decisions, handling constraints imposed by other modeling notations, supporting the modeling of context dependent information, and providing a well-defined reuse interface.

**Hypothesis 2.** It is feasible to determine the required goal model concepts and tool-supported algorithms for goal model reuse in GRL that satisfy the five requirements R.1 to R.5: providing trade-off reasoning capability via goal model evaluation through

reuse hierarchies in the presence of delayed decision-making while taking constraints imposed by other models into account, providing support to analyze context dependent information, and providing an interface for reuse.

Verification of whether these two hypotheses hold in the current state of the corpus of goal modeling paves the way to improve the reusability of existing goal modeling approaches in line with the findings of this research and the five requirements that are identified, and consequently answer our main research question: How can goal models be made reusable?

## 1.4 Methodology

For the assessment of the first hypothesis (Hypothesis 1), a comparative analysis of the proposed approach with existing goal model reuse techniques is performed with respect to their ability to satisfy the five requirements. For this analysis, a two-part systematic literature review investigating *goal model reuse* and *context and goal models* is conducted.

To check whether the second hypothesis (Hypothesis 2) holds, goal model concepts required for reuse are formulated for GRL and evidences of feasibility and scalability are gathered via the implementation of tool support and necessary algorithms that support the five requirements. Proof-of-concept implementations are provided for the TouchCORE [27] tool.

GRL is chosen based on our experience with it, but the results of this work can still be applied to other languages given the similarity of major goal modeling languages.

## 1.5 Thesis Contributions

The contributions of this thesis are gathered under eight main topics and listed as follows:



**C.1.** The first contribution of this doctoral research is the *identification of the five requirements* that need to be fulfilled at least for the reuse of goal models based on our experience.

To check whether the stated hypotheses of this research hold and to address the stated technical challenges, the results of the research activities, described as part of the research methodology in Section 1.4, are listed as the following contributions.

**C.2.** A two-part *systematic literature review* to assess the first hypothesis focusing on “goal model reuse” and “context and goal models” in the existing body of work was conducted.

Findings of this literature review have shown that none of the existing goal model reuse approaches completely addresses the five requirements and that there is a need for a comprehensive goal modeling approach that supports all five requirements (see Chapter 3).

**C.3.** For the evaluation of reusable goal models in the absence of real-life measurements, a new approach to assign and interpret contribution values was defined. For this purpose, the concepts of *relative contribution values and normalization* with scaling factors and offsets were introduced [28].

The GRL metamodel was extended to cover these new concepts [29], which are prerequisites to satisfy R.1 (see Section 4.2 and 4.3).

**C.4.** The evaluation mechanism was updated to *handle external constraints* imposed by feature models [28] and workflow models [24]. The GRL metamodel was extended to support handling of workflow constraints by identifying run-time and design-time goals.

The updated evaluation mechanism allows for more accurate trade-off analysis in reusable goal models as required by R.3 (see Chapter 5).

**C.5.** To support the delaying of decisions in the goal model reuse hierarchies, partial reuse of a reusable artifact while retaining trade-off analysis capabilities is provided with the

help of a *range evaluation mechanism* [30].

The idea is to evaluate the possible minimum and maximum satisfaction ranges for intentional elements considering both the decisions made and the ones that will be made at a later point in the reuse hierarchy (i.e., they are currently unknown).

Having such an evaluation mechanism addresses not only R.2, but also R.4 to a certain extent, as the information being introduced to the model at later stages is contextual (see Section 6.4). While various contextual goal modeling approaches exist in the literature (see Section 3.2.2) that introduce different means to model contextual information, none of the surveyed approaches considers complexities related to large goal model reuse hierarchies. The range evaluation mechanism takes into account uncertain intentional elements across the full goal model reuse hierarchy and provides support to analyze contextual information in goal model reuse hierarchies..

**C.6.** The GRL metamodel was extended to include constructs for *reuse links and feature impact elements* to provide an interface for reuse (R.5) and allow the goal model evaluation to span the entire reuse hierarchy [29][31] as required by R.1.

With the proposed changes to goal model evaluation, our approach deals with the problem of large monolithic models resulting from, e.g., copy/paste-based reuse that are hard to maintain and analyze as the reuse hierarchy grows. Our approach introduces an iterative evaluation of goal models at different levels of the reuse hierarchy while keeping their modular structure.

Furthermore, our approach allows elements of the reused goal model to be differentiated from elements of the higher-level reusing goal model (both at the model and the metamodel level), resulting in a clear goal model reuse interface that hides elements of lower-level models and provides proper encapsulation (see Section 6.2).

**C.7.** In addition to the novel bottom-up evaluation algorithms introduced for different stages in goal model reuse hierarchies, a *top-down evaluation algorithm* that allows designers to find the optimal set of solutions to satisfy the stakeholders of their system and their goals was introduced [32].

The top-down evaluation algorithm takes external constraints into account and takes advantage of the reuse boundaries to deal with the complexity of the evaluation as reuse hierarchies grow larger.

For goal prioritization and the adaptation of the existing bottom-up evaluation algorithm to top-down evaluation of goal model reuse hierarchies, three modeling constructs (i.e., actor, importance, and threshold) are used by the top-down evaluation algorithm (see Section 6.5).

**C.8.** *Proof-of-concept implementations* [24][28][29][31] are realized in the TouchCORE [27] tool to provide evidence for the feasibility and scalability of the proposed improvements to validate the second hypothesis.

As a result, support is available for reusable goal modeling activities identified by the key requirements.

## 1.6 Publications and Presentations

All of the author’s publications to date that are related to this doctoral research are listed in this section in chronological order. The first author is the main author who contributed the majority of the work to the publication.

- Duran, M.B. and Mussbacher, G. (2015) Enabling Reuse With Relative Contribution Values in Goal Models. Presentation, *1st Concern-Oriented Reuse (CORE) at Bellairs Workshop*, McGill Bellairs Research Institute, Barbados, January-February 2015.

- Duran, M.B., Navea Pina, A., and Mussbacher, G. (2015) Evaluation of Reusable Concern-Oriented Goal Models. *5th International Model-Driven Requirements Engineering Workshop (MoDRE 2015)*, Ottawa, Canada, August 2015. IEEE CS, 1-10. DOI: 10.1109/MoDRE.2015.7343876.
- Alexandre, R., Camillieri, C., Duran, M.B., Navea Pina, A., Schöttle, M., Kienzle, J., and Mussbacher, G. (2015) Support for Evaluation of Impact Models in Reuse Hierarchies with jUCMNav and TouchCORE. Tool Demo, *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS 2015)*, Demo and Poster Sessions, Ottawa, Canada, October 2015. CEUR-WS 1554:28-31.
- Duran, M.B., Mussbacher, G., Thimmegowda, N., and Kienzle, J. (2015) On the Reuse of Goal Models. *17th International System Design Languages Forum (SDL 2015)*, Berlin, Germany, October 2015. Fischer, J., Scheidgen, M., Schieferdecker, I., and Reed, R. (Eds.), *SDL 2015: Model-Driven Engineering for Smart Cities*, Springer, LNCS 9369:141-158. DOI: 10.1007/978-3-319-24912-4\_11.
- Duran, M.B., Thimmegowda, N., Kienzle, J., and Mussbacher, G. (2016) On the Reuse of Goal Models. Presentation, *Montréal Software Analysis Research Talks (MOSART 2016)*, Montreal, Canada, May 2016.
- Duran, M.B., Schöttle, M., Kienzle, J., and Mussbacher, G. (2016) Support for Evaluation of Impact Models in Reuse Hierarchies with TouchCORE. Poster and Tool Demo, *Montréal Software Analysis Research Talks (MOSART 2016)*, Montreal, Canada, May 2016.
- Duran, M.B. and Mussbacher, G. (2016) Investigation of Feature Run-Time Conflicts on Goal Model-Based Reuse. *Information Systems Frontiers (ISF)*, Springer 18(5):855-875. DOI: 10.1007/s10796-016-9657-7.

- Duran, M.B. and Mussbacher, G. (2017) Evaluation of Reusable Impact Models with TouchCORE. Tool Demo, *Consortium for Software Engineering Research (CSER) 2017 Spring Meeting*, Montreal, Canada, May 2017.
- Duran, M.B. and Mussbacher, G. (2017) Evaluation of Reusable Impact Models. Poster, *Montreal Symposium on Software Engineering Research (MOSSER 2017)*, Montreal, Canada, May 2017.
- Duran, M.B. and Mussbacher, G. (2017) Evaluation of Reusable Impact Models. Poster Paper, *8th Summer School on Domain Specific Modelling Theory and Practice (DSM-TP 2017)*, Montreal, Canada, July 2017.
- Duran, M.B. and Mussbacher, G. (2017) Evaluation of Goal Models in Reuse Hierarchies with Delayed Decisions. *7th International Model-Driven Requirements Engineering Workshop (MoDRE 2017)*, Lisbon, Portugal, September 2017. IEEE CS, 6-15. DOI: 10.1109/REW.2017.66.
- Duran, M.B. (2017) Reusable Goal Models. Doctoral Symposium, *25th IEEE International Requirements Engineering Conference (RE 2017)*, Lisbon, Portugal, September, 2017. IEEE CS, 532-537. DOI: 10.1109/RE.2017.34.
- Duran, M.B. and Mussbacher, G. (2018) Top-down Evaluation of Reusable Goal Models. *17th International Conference on Software Reuse (ICSR 2018)*, Madrid, Spain, May 2018. Capilla R., Gallina B., and Cetina C. (Eds.), *New Opportunities for Software Reuse*, Springer, LNCS 10826:76-92. DOI: 10.1007/978-3-319-90421-4\_5.
- (under review) Duran, M.B. and Mussbacher, G. (2018) Reusable Goal Models: A Systematic Literature Review.

## 1.7 Formatting Conventions

Important domain-specific terms are indicated with *italic font style* while the usage of a `typewriter font` refers to model elements used in figures throughout the thesis. *Computer Modern* is used for numbers and mathematical equations.

## 1.8 Thesis Outline

The remainder of this thesis is structured as follows: Chapter 2 provides background on goal modeling and trade-off analysis. Chapter 3 reports on the state of the literature through an assessment of the capabilities of existing goal model reuse and contextual goal modeling approaches with the help of a systematic literature review. Chapter 4, then, investigates the different types of contribution values in goal modeling and how they should be assigned in a reusable goal model, followed by a discussion of how constraints modeled in other modeling formalisms may impact reusable goal model analysis in Chapter 5. Chapter 6 presents goal model reuse hierarchies built via bringing together smaller reusable goal models and how they can be evaluated with our novel approach taking delayed decisions and contextual information into account. Finally, Chapter 7 summarizes the thesis and states future research directions to build on this work.

# Chapter 2

## Background

The background chapter gives an overview of goal modeling and mainstream goal modeling languages with their commonalities and differences in Section 2.1. One of the major strengths of goal modeling, i.e., trade-off analysis, is then explained and exemplified through commonly used top-down and bottom-up evaluation methodologies in Section 2.2.

### 2.1 Goal Modeling

As motivated in Chapter 1, goal models are valuable requirements engineering tools due to their expressiveness in capturing system requirements, stakeholder objectives, and their relationships, and the trade-off analysis capabilities they provide through goal model evaluation [5][6][7][15]. In the early requirements phase, goal modeling helps requirements engineers to understand stakeholder goals and explore alternative solutions and their potential impacts on these goals.


Various goal modeling languages exist in literature. Key examples include the NFR framework [5], i\* [6], the Goal-oriented Requirement Language (GRL) [7], Tropos [8], and KAOS [9]. Even though differences exist among these languages in terms of the concepts they include, there are also a lot of similarities. In a broad sense, the concept of a *goal* exists in all languages. While NFR framework only uses *softgoals* (i.e., goals for which there is no objective measure of satisfaction), i\* and GRL further classify intentional elements into *goals*, *softgoals*, *tasks*, and *resources*. AND and OR *decompositions* among goals is possible





Figure 2.1: GRL notation with qualitative contribution and satisfactions

in all languages. All languages except KAOS cover *contribution links*, although the means to express contributions differ.

Throughout the thesis, GRL is used to illustrate goal models (see Figure 2.1). GRL is a visual modeling notation for the specification of intentions, business goals, and nonfunctional requirements (NFRs) of multiple stakeholders as well as system qualities. GRL is part of the User Requirements Notation (URN) [7], an international requirements engineering standard published by the International Telecommunication Union in the Z.15x series. GRL is based on  $i^*$  (in terms of key goal concepts) and the NFR Framework (in terms of evaluating goal models), but allows modeling elements to be more freely combined than  $i^*$ .

A GRL goal model is a graph of *intentional elements* (softgoal, goal, task, resource) that are connected via *links* (decomposition, contribution, or dependency). Intentional elements may reside within an *actor* () which represents a stakeholder of a system or the system itself. When representing stakeholders, actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, softgoals to be satisfied, and resources to be available.

A GRL goal model shows the business goals, non-functional requirements, and qualities of interest to a stakeholder and the solutions considered for achieving these high-level elements. To model these goals, non-functional requirements, and qualities, *goals* () and *softgoals* () are used. Softgoals differentiate themselves from goals in that there is no clear, objective



measure of satisfaction for a softgoal, whereas a goal is quantifiable. Therefore, the term *satisfied* is often used to indicate that a softgoal is satisfied and that this satisfaction may be dependent on a stakeholder's point of view. *Tasks* ( $\Diamond$ ) represent solutions to goals or softgoals that are considered for a system. In order to be achieved or completed, softgoals, goals, and tasks may require *resources* ( $\Box$ ) to be available.

Links in a goal model express the relationships between goals and softgoals (representing stakeholder goals, system requirements, or qualities), and tasks (representing solutions). In order to connect the elements of a goal model, various types of links are available. *Decomposition links* ( $+—$ ) allow an element to be decomposed (AND, XOR, and IOR) into sub-elements. *Contribution links* ( $\rightarrow$ ) are used to indicate desired impacts of one element on another element, which may be positive or negative. Contributions are expressed either quantitatively as an integer value between  $-100$  and  $100$  or qualitatively with labels (e.g.,  $++$  or  $--$ ). Finally, *dependency links* ( $- \blacksquare -$ ) model relationships between actors, i.e., one actor depending on another actor for something.

## 2.2 Analysis and Reasoning with Goal Models

A major reason why goal models are widely used in requirements engineering practices is their trade-off analysis capability. The ultimate objective of goal modeling is stated as to provide support for some form of reasoning for requirements elaboration, consistency and completeness checking, selection of alternatives, evolution management, etc [21]. Goal model analysis helps modelers with decision-making in the presence of conflicting stakeholder goals and requirements as well as to reason about system qualities, non-functional requirements, and their potential solutions.

In a general sense, goal model evaluation aims for either (i) finding the optimal set of solutions (i.e., design decisions, leaf nodes in a goal model; e.g., tasks) to satisfy a specific

set of stakeholder goals (i.e., root nodes) or (ii) reasoning about the impacts of alternative solutions (i.e., leaf nodes) on stakeholder goals (i.e., root nodes). While the latter is often achieved by a bottom-up evaluation, a top-down evaluation approach is used for the former [33].

Forward propagation mechanisms propagate the satisfaction values of leaf elements up to their parent elements based on the type and value of the link between them, whereas backward propagation mechanisms aim to find optimal sets of solutions (e.g., leaf nodes) given the desired satisfaction values of high-level goals (e.g., root nodes).

Commonly found types of links in goal models are contributions, decompositions, and dependencies. Contribution links indicate desired impacts of one element on another element, either qualitatively with labels (e.g., ++, +, -, or --) or quantitatively with an integer value in the range of  $[-100, 100]$ . The resulting satisfaction values are also either expressed with labels (e.g., *fully satisfied* or *weakly denied*) or quantitatively in the range of  $[-100, 100]$  or  $[0, 100]$ . A satisfaction value of 100 indicates that a task is selected or that a stakeholder/system goal is fully achieved. A satisfaction value of 0, on the other hand, indicates that a task is not selected or that a stakeholder/system goal is fully denied.

GRL supports reasoning about high-level goals, non-functional requirements, and system qualities, through its evaluation mechanism [22]. GRL shows the impact of often conflicting goals and various proposed candidate solutions to achieve the goals. In GRL, a *strategy* describes a particular set of candidate solutions that is to be evaluated by assigning initial satisfaction values to the tasks in the goal model that correspond to the candidate solutions. Typically, those are the leaf nodes in the goal model. A satisfaction value expresses the degree with which a softgoal or goal is satisfied, a task is performed, or a resource is available. Unless a satisfaction value of a node is assigned by the modeler with a strategy, the satisfaction value of a node is calculated with the help of propagation-based evaluation [5], using the incoming links (type of the link and the contribution value if it is a contribution link) and the

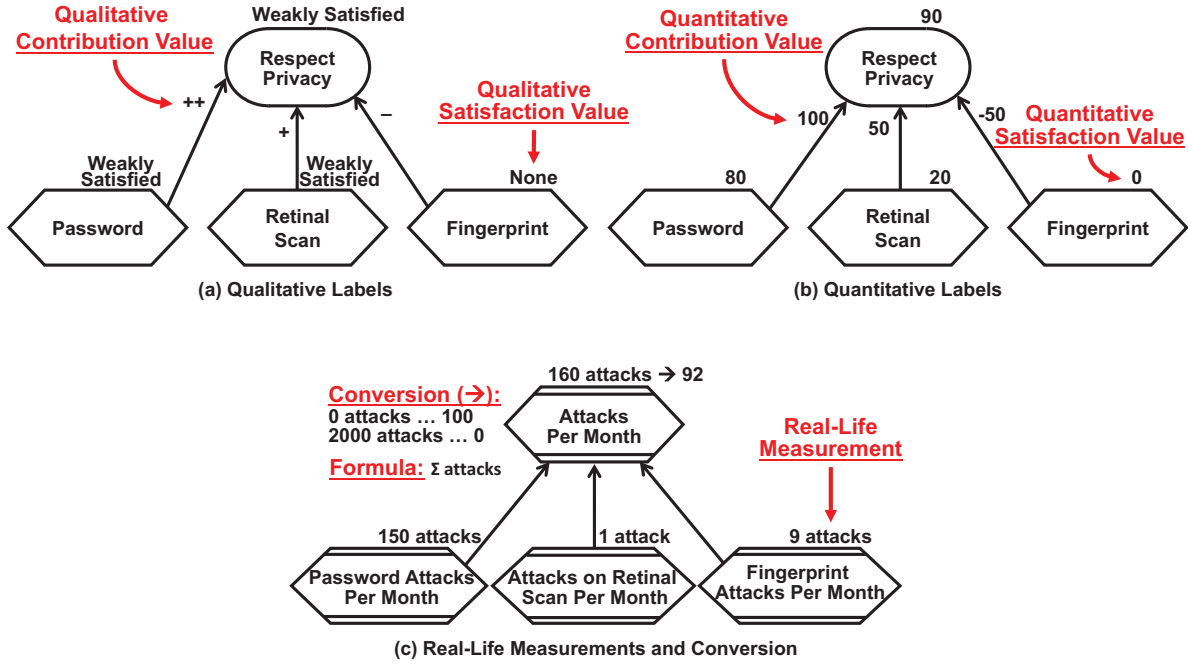


Figure 2.2: Contribution and satisfaction values in goal models [24]

satisfaction values of the children nodes. A satisfaction value can be a real-life measurement, a qualitative value, or a quantitative value as shown in Figure 2.2. Therefore, modelers can compare several strategies with each other using the resultant satisfaction values of high-level goal model elements, facilitating the decision making process between alternative solutions or prioritization of the requirements and/or stakeholder goals.

The evaluation mechanism calculates a satisfaction value for each existing parent element based on its incoming links and the satisfaction values of its children, unless the element's satisfaction value is defined directly by the software engineer.

If the values in a goal model are specified qualitatively (e.g., by labels such as *Denied*, *Weakly Denied*, *None*, *Weakly Satisfied*, and *Satisfied*), the evaluation mechanism explicitly defines a mapping that determines (a) which label should be propagated up given a child's satisfaction value, link type, and contribution value (if applicable), and (b) how the prop-

agated labels of a node's children should be combined to yield the qualitative satisfaction value of the node (see Figure 2.2a).

If the values in a goal model are specified quantitatively, the weighted sum of the children's satisfaction values is propagated across contribution links to the parent (e.g.,  $[(80 \times 100) + (20 \times 50) + (0 \times -50)]/100 = 90$ ; see Figure 2.2b). The result is limited to the allowed range of  $[0, 100]$  for satisfaction values. In the case of these globally defined quantitative contribution values, having 100 as a contribution value (e.g., the contribution link from **Password** to **Respect Privacy** in Figure 2.2b) means that when the contributor itself (i.e., **Password**) is fully satisfied, it can alone fully satisfy the goal it contributes to (i.e., **Respect Privacy**). If the contributor is not fully satisfied, a combination of contributors may still push the satisfaction of the goal to 100.

Propagation through decomposition links differ from the propagation of contribution links as it depends on the type of decomposition link. For an element with AND decompositions, its satisfaction level is calculated as the minimum of the quantitative evaluation values of its source elements (i.e., decomposed elements). For an element with IOR decomposition links, its satisfaction value is the maximum value of the quantitative satisfaction values of its source elements. For an element with XOR decompositions, the maximum is used again, but if more than one source element has a non-zero satisfaction value, a warning is generated. A dependency link, on the other hand, dictates that the source of the link (i.e., the dependent element) cannot have a satisfaction value higher than the target element [22].

The satisfaction values in a goal model may also be specified by real-life measurements with the help of Key Performance Indicators (KPIs) ( $\Leftarrow$ ) [7] to increase the accuracy of the goal model evaluation. In this case, a formula [34] defines how the KPIs of the children are combined to yield the KPI of the parent.

As needed, a real-life measurement may be converted into a goal model satisfaction value based on a conversion function [7] (e.g., the attack KPIs of the children are summed up for

the parent as specified by the formula, and the resulting KPI of 160 attacks for the parent is then converted into the goal model satisfaction value of 92 given the specified conversion function (e.g., using linear interpolation); see Figure 2.2c). A conversion function maps real-life values onto the allowed  $[0, 100]$  range of satisfaction values, enabling the comparison of KPIs measured in different units and other goal model elements.

By mapping a real-life measurement to the satisfaction value of 100, the conversion deems this measurement (and any measurement beyond it) as sufficient for the current context. For example, while a KPI simply measures waiting time in minutes, the conversion function may specify that 5 minutes of waiting time fulfills the needs of the application under development by mapping 5 minutes to the maximum satisfaction value of 100 and that 24 hours is the worst result by mapping 24 hours to the minimal satisfaction value of 0).

The evaluation algorithm consists of an initialization phase and a propagation phase. The initialization phase divides all intentional elements of the goal model into two groups based on the chosen strategy: those that are ready to be evaluated and those that are not yet ready to be evaluated. Elements with satisfaction values defined by the strategy and all leaf elements (no incoming links) belong to the former group. Elements in this group are assigned either the satisfaction value defined by the strategy or 0 if the element is a leaf element that is not set by the strategy.

During the propagation phase, the evaluation algorithm determines successively the satisfaction values of the remaining elements in the second group. An element in this group is evaluated as soon as the satisfaction values of all its children are known. Consequently, the evaluation mechanisms calculate satisfaction values for all existing elements, unless circular dependencies exist in the goal model, in which case some elements may not be evaluated (this is a general constraint of a propagation-based approach).

In the remainder of this section, we investigate briefly the differences and commonalities among five mainstream goal modeling notations (i.e., NFR framework,  $i^*$ , GRL, Tropos, and

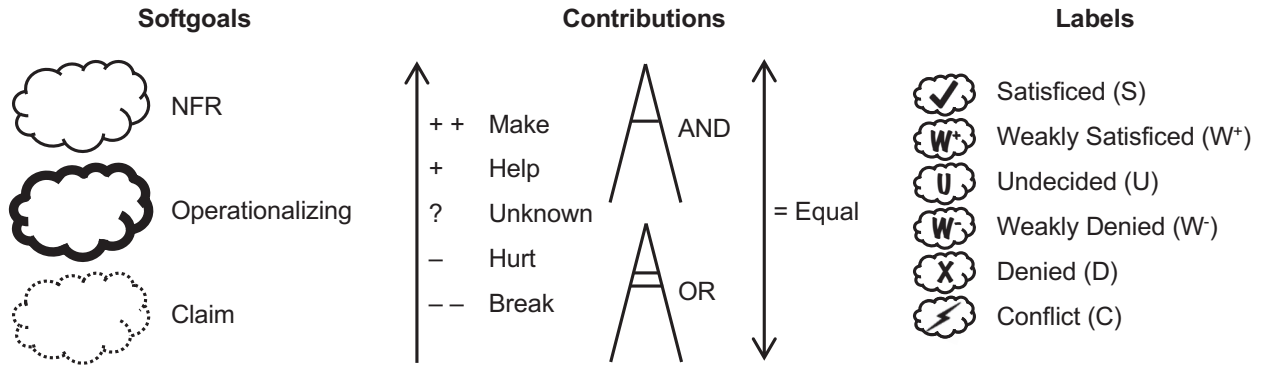


Figure 2.3: Legend for software interdependency graphs of the NFR Framework

KAOS) in terms of the concepts and modeling constructs they make use of for their provided analysis and reasoning capabilities.

### 2.2.1 NFR Framework

Softgoal interdependency graphs of the NFR Framework [5] introduces three different types of softgoals as NFR softgoals, operationalization softgoals, and claim softgoals as shown in Figure 2.3.

For goal model evaluation, the NFR Framework uses forward propagation through decomposition (i.e., *AND/OR*) or contribution links where contributions are assigned as qualitative labels (i.e., *break*, *hurt*, *unknown*, *help*, or *make*). The evaluation result is also a label (i.e., *denied*, *weakly denied*, *undecided*, *weakly satisfied*, *satisfied*, or *conflict*) that expresses the degree to which non-functional requirements are achieved.

In the label propagation algorithm of the NFR Framework, while a contribution with the label “*make*” signifies sufficient positive support, the contribution label “*break*” is used to represent sufficient negative support to a goal from its contributor [5]. In other words, having one *satisfied* contributing node with a “*make*” contribution allows the parent node to be *satisfied* and having one *satisfied* contributing node with a “*break*” contribution allows

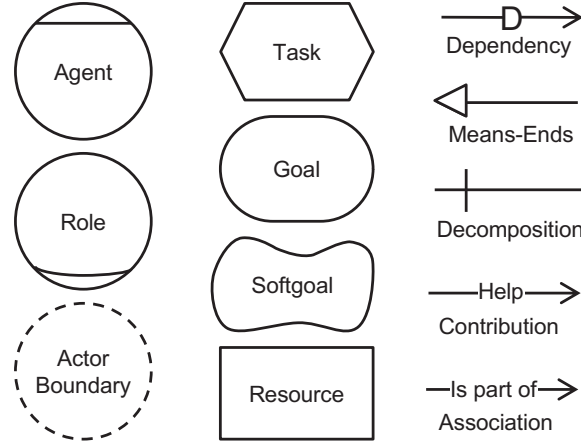


Figure 2.4: Excerpt of the i\* notation

the parent node to be *denied*.

The NFR Framework refers to *AND* and *OR* decompositions as contributions as well. For an *AND* contribution, if all of the contributing nodes are *satisfied*, their parent node will be *satisfied*. In the case of an *OR* parent, the parent node will be *satisfied* if any of the contributing nodes is *satisfied*.

### 2.2.2 i\*

The qualitative evaluation procedure for i\* [35] adopts the contribution labels used by the NFR Framework for their propagation based satisfaction analysis.

The same labels (*make/break*) are used in the qualitative evaluation procedure for i\* to represent evidence that is considered sufficient to satisfy or deny a goal.

In addition to the contribution links, i\* contains dependency, decomposition, means-ends, and association links (see Figure 2.4). A dependency is used to make sure that a node element is satisfied only if the node element on which it depends is satisfied. A decomposition link represents an *AND* relationship among children from which the propagation algorithm

selects the minimum valued child for propagation. On the other hand, a means-ends link is used to represent the inclusive *OR* relationships where the maximum valued children node amongst all children is selected by the evaluation algorithm for propagation. Association links in *i\** represent relationships between *actors*, *agents*, and *roles*.

### 2.2.3 GRL

GRL [22], bringing together the core concepts of both *i\** and the NFR Framework, provides three algorithms (quantitative, qualitative, and hybrid) for the evaluation of goal models.

For the hybrid evaluation, GRL uses a conversion table that maps qualitative labels to their corresponding quantitative contributions and conducts a quantitative evaluation upon conversion of labels.

The quantitative evaluation algorithm of GRL [22] is based on a weighted-sum approach and expresses contributions with an integer value between  $-100$  and  $100$ . The upper ( $100$ ) and lower ( $-100$ ) boundary values correspond to the two labels *make* and *break*, respectively, that are used in the qualitative evaluation.

Furthermore, a user-defined *tolerance* value is used in quantitative and hybrid evaluations. A tolerance value in GRL evaluation makes sure that the evaluation result for a node can be maximized (i.e.,  $100$ ) or minimized (i.e.,  $-100$ ) *only if* there is at least one contributor (or child node) with a contribution value (or a decomposition value) of  $100$  or  $-100$ , respectively [22].

Other types of GRL links are decomposition, dependency, correlation (a kind of contribution), and means-end links, which are all similar to links in the NFR Framework or *i\**. Decomposition links may take the form of *AND*, *IOR* (i.e., inclusive *OR*), or *XOR* (i.e., exclusive *OR*) decompositions where the last two can be represented as means-end links as well.



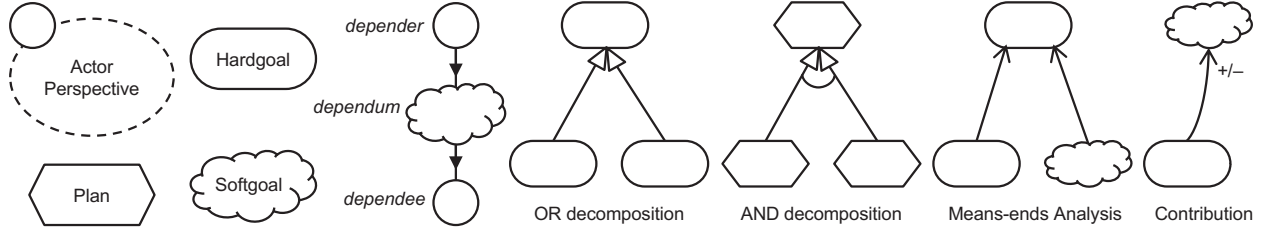


Figure 2.5: Excerpt of the Tropos notation

GRL introduces a new aspect to the evaluation of goal models with Key Performance Indicators (KPIs) [36]. KPIs represent real-life measurements and allow modeling their impacts on stakeholder goals or system requirements. Incorporating real-life measurements in the form of KPIs in GRL evaluation [36] allows modelers to express their idea of what is sufficient for the performance values being monitored.

GRL evaluation with KPIs is done via mapping real-life measurement results to GRL evaluation levels using three user-defined values (i.e., *target*, *threshold*, and *worst values* defined for KPIs are mapped to 100, 0, and  $-100$ , respectively). While *target value* represents the expected performance, *worst value* is used to set the lower-bound for the worst-case result. The *threshold value*, on the other hand, separates acceptable and unacceptable values. A real-life measurement taken for a KPI is mapped to a quantitative evaluation value according to these three levels. This means that the modeler, and hence the evaluation, has no interest in the measurements beyond *target* (or *worst*) value as it corresponds to the value that is sufficiently good (or bad) for the performance.

## 2.2.4 Tropos

Tropos [37] aims to capture goal and softgoal relationships from early requirements analysis to architectural design. To reason about a goal's satisfaction and denial, it provides forward and backward reasoning capability using qualitative labels where a goal's satisfiability (and

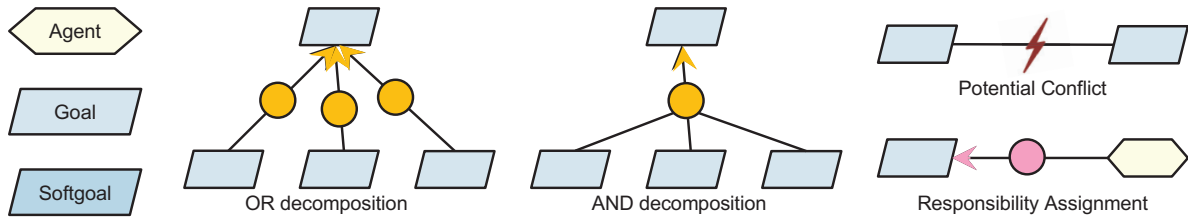


Figure 2.6: Excerpt of the KAOS framework constructs for goal modeling

deniability) is expressed as *full*, *partial*, or *none*.

Similar to the qualitative contribution labels used by the NFR Framework,  $i^*$ , and GRL, Tropos [37] uses two labels, ++ and -- to represent the existence of full evidence on the satisfaction or denial of a goal.

In terms of decompositions, *AND* and *OR* relationships are also used in Tropos (see Figure 2.5) for both forward [38] and backward [39] reasoning with the same meaning in the propagation algorithms as the aforementioned approaches.

### 2.2.5 KAOS

KAOS [9] is a methodology that combines goal modeling with responsibility modeling, object modeling and operation modeling to help analysts in the process of generating requirements documents.

The quantitative analysis approach with KAOS [40] uses probability values of goals being satisfied or denied and propagates partial degrees of satisfactions to reason about alternative designs.

Instead of contribution links, KAOS makes use of various refinement links (e.g., to express *AND* decompositions, alternatives, etc.) as well as conflicts and obstructions (see Figure 2.6). KAOS [40] uses an *AND/OR* tree similar to the other four approaches but with probabilistic contributions.

## 2.3 Summary

Goal models are valuable tools for requirements engineers due to their expressiveness in capturing requirements of systems, objectives of stakeholders, and their relationships, as well as the trade-off analysis and reasoning capabilities provided through evaluation.

Five mainstream goal modeling notations, the NFR framework [5], i\* [6], the Goal-oriented Requirement Language (GRL) [7], Tropos [8], and KAOS [9] all have their own way of representing intentional elements and the relationships among them. While the particular concepts these languages use for this representation differ, there exist a lot of commonalities among them. For instance, the concept of a *goal* exists in all languages. While i\* and GRL further classify intentional elements into *goals*, *softgoals*, *tasks*, and *resources*, NFR framework only uses *softgoals*. In terms of the relationships among intentional elements, all languages except KAOS cover *contribution links*; however, AND and OR *decompositions* among goals exist in all languages. Furthermore, different means for trade-off analysis are used by different formalisms.

Following the overview of goal modeling and analysis and reasoning mechanisms presented in this chapter, the next chapter presents an overall assessment of the capabilities of existing goal model reuse and contextual goal modeling approaches with the help of a systematic literature review.

# Chapter 3

## State of the Art:

## Systematic Literature Review

An approach that supports goal model reuse must address the five requirements identified in Chapter 1 as: (i) to enable analysis and validation of goal models through reuse hierarchies, (ii) to provide the means to delay decision making to a later point in the reuse hierarchy, (iii) to take constraints imposed by other modeling notations into account during goal model analysis, (iv) to allow context dependent information to be modeled so that the goal model can be used in various reuse contexts, and (v) to provide an interface for reuse.

To understand the existing landscape of goal modeling approaches with respect to these five requirements, a two-part systematic literature review that addresses (i) goal model reuse and (ii) contextual goal models is conducted.

The research methodology for this two-part systematic literature review including threats to validity is described in Section 3.1. The methodology is followed by a discussion on the results and analysis as well as the future research themes in Section 3.2. The chapter concludes with a summary of conclusions derived from the systematic literature review in Section 3.3.

## 3.1 Methodology

For the investigation of the existing body of knowledge on goal model reuse and representation of context related information in goal models (i.e., contextual goal models), the systematic literature review is conducted following the guidelines proposed by Kitchenham [41]. The procedure followed for the two-part systematic literature review and the threats to validity are described in this section.

### 3.1.1 Research Questions

Emerging from the previously stated requirements for reusable goal models with contextual information, the following research questions are articulated:

- Q.1)** To what extent do goal modeling approaches satisfy the five requirements R.1 to R.5: analysis and validation across the reuse hierarchy, delaying decisions, handling constraints imposed by other modeling notations, means to model context dependent information, and a defined reuse interface.
- Q.2)** Which goal modeling concepts are used or introduced to represent context in a goal modeling notation?
- Q.3)** What other requirements models, if any, are used in collaboration with reusable goal models?
- Q.4)** For which reusable goal modeling approaches does tool support exist?
- Q.5)** Which research themes need to be investigated further to realize contextual and reusable goal models and fulfill all identified requirements (R.1 to R.5).

#### 3.1.2 Selection of Publications

To answer the research questions, the initial publication set is built upon two searches on seven major academic search engines (i.e., ACM Digital Library, Google Scholar, IEEE Xplore, ScienceDirect, Scopus, SpringerLink, and Web of Science) using the search strings:

1. *goal model* AND (*reuse* OR *reusability*)
2. *goal model* AND (*context aware* OR *context oriented* OR *context based* OR *context dependent* OR *context driven*)

with the proper syntax for each search engine (e.g., for the search terms with context, terms with both space and dash are used) as shown in Table 3.1. The search for goal model reuse (i.e., search string 1) was performed on May 12, 2016 and the search for context (i.e., search string 2) on November 13, 2016.

The reasoning behind performing a second, separate search for context stems from the initial results of the first search for goal model reuse.

Initially, the publications in the field of goal model reuse were targeted very broadly to get an overview of existing reuse techniques with the first search string. While the compliance of the approaches discovered as a result of the first search with four of the requirements (i.e., R.1-3 and R.5) were satisfactory, the requirement related to contextual information (i.e., R.4) called for further investigation.

From experience, and by looking into the references that were discovered, we knew that a substantial body of work in this area is not covered by goal model reuse and our intuition was that authors do not talk about reuse explicitly while working on context. However, for reuse, it is essential to understand the context first and that is the kind of reasoning widely used by self-adaptive and context-aware systems. Consequently, the second search string was used to further explore the existing landscape of context aware goal models for their

Table 3.1: Search Syntax Used per Search Engine

Search Engine	String	Exact Syntax
ACM Digital Library	1	+("goal model") +(reuse reusability)
	2	+("goal model") +("context aware" "context-aware" "context oriented" "context-oriented" "context based" "context-based" "context dependent" "context-dependent" "context driven" "context-driven")
Google Scholar (not logged in)	1	"goal model" reuse OR reusability
	2	"goal model" AND ("context aware" OR "context oriented" OR "context based" OR "context dependent" OR "context driven")
IEEE Xplore ScienceDirect Scopus SpringerLink Web of Science	1	"goal model" AND (reuse OR reusability)
	2	"goal model" AND ("context aware" OR "context-aware" OR "context oriented" OR "context-oriented" OR "context based" OR "context-based" OR "context dependent" OR "context-dependent" OR "context driven" OR "context-driven")

ability to support goal model reuse.

To ensure repeatability of this systematic literature review, every step in the process and every decision made are clearly documented to keep track of all changes from one step to the next for transparency and verification purposes with the help of a reference management tool [42]. An overview of the whole process and the search results is given in Figure 3.1.

Table 3.2 shows the breakdown of the initial search results. The search for goal model reuse (i.e., search string 1) on Google Scholar alone resulted in 2,100 results. Considering that investigating all of them would be impractical, we limited the results from Google Scholar to the first 1,000 results listed according to the relevance criteria of the search engine. With the addition of 919 hits from the other six academic search engines, we ended up with a total of 1,919 raw search hits, which was brought down to 1,597 after the removal of duplicate results (i.e., the publications that occur multiple times in the combined results).

Table 3.2: Initial Search Results

Search Engine	Number of Hits	
	Reuse (1)	Context (2)
ACM Digital Library	5	4
Google Scholar	1,000*	702
IEEE Xplore	2	1
ScienceDirect	172	52
Scopus	27	12
SpringerLink	705	214
Web of Science	8	1
<b>Total with duplicates</b>	<b>1,919</b>	<b>986</b>
<b>Total after removal of duplicates</b>	<b>1,597</b>	<b>815</b>

\* the first 1,000 hits out of 2,100 were considered according to the relevance criteria of the search engine.

The search for context (i.e., search string 2) yielded a total of 986 raw search hits which was brought down to 815 after the removal of duplicate results. Hence, all search hits for context were examined in the subsequent phase.

### First Phase

For the first phase of evaluation of the initial set of publications, inclusion criteria are developed as follows:

1. Publication contains information on goal models
2. Publication addresses reuse (for the result set of the search for goal model reuse) *OR* contextual information (for the result set of the search for context)
3. There is correlation between goal modeling and reuse (for the result set of the search for goal model reuse) *OR* goal modeling and context (for the result set of the search for context)



for context)

4. Publication is written in English
5. Type of the publication is one of the following: conference paper, workshop paper, journal article, book chapter, book, or thesis

For a publication to be included in the second phase, it is required to comply with all of the inclusion criteria listed (i.e., there is an *AND* relationship among the five inclusion criteria).

The first phase evaluation was conducted in two stages for both search strings. Due to the large number of search results, only the author assessed the complete sets of 1,597 and 815 publications with respect to the inclusion criteria. The supervisor performed the same assessment on a smaller set to check and verify the accuracy of the selection. The control sets for the supervisor were prepared by doubling the amount of included publications from the author's assessment by including the same number of randomly picked publications from the excluded ones.

Consequently, in the first stage, the author assessed the 1,597 publications in the result set of the search for goal model reuse and concluded that 74 publications fit all of the inclusion criteria whereas 1,523 were to be excluded.

For the second stage, a second set of 74 publications were picked randomly out of the 1,523 that were excluded by the author and were merged into the original set of 74 included publications, without leaving any indication of which papers were included by the author. The supervisor then reviewed and classified this new group of 148 publications according to the inclusion criteria. After the supervisor's classification, a discussion session was held to revisit the disagreements. The supervisor initially decided to include 2 new publications that were not included by the author. The reason for disagreement on those 2 publications being missed by the author was due to the interpretation of the correlation between reuse and goal modeling as specified by the inclusion criterion 3. The supervisor included publications with

a weaker correlation. The classification of those 2 publications were changed, however, in favor of excluding them after the discussion between the author and the supervisor.

Following the discussion session, both the author and the supervisor agreed to move all 74 publications that were initially included by the author on to the second phase. Given the large overlap of the assessment results of both the author and the supervisor, the confidence in the accuracy of the list moved to the second phase is deemed high enough to continue with the second phase.

Similarly, for the result set of the search for context, 100 out of the initial publication set of 815 were included by the author in the first stage. In the second stage, a verification set of 200 publications was prepared by randomly picking an additional 100 out of the 715 publications that were excluded by the author in the first stage.

Following the classification of the supervisor and the discussion session, 7 new publications were included in addition to the 100 that were selected in the first stage, resulting in 107 publications to be moved on to the second phase in accordance with the inclusion criteria. Among the 7 newly included publications, 5 were different publications by the same authors that were already included in the initial 100. The primary reason for those 5 publications being missed by the author stems from the interpretation of inclusion criterion 3 (i.e., correlation between goal modeling and context). The supervisor included publications with a weaker correlation. Nevertheless, due to the large overlap of the assessment results of both the author and the supervisor (only 2 entirely new publications), the confidence in the accuracy of the list moved to the second phase is deemed high enough to continue with the second phase.

The large number of publications being excluded in the first phase is mainly due to not restricting the search to titles or abstracts of the publications. As a result of this, the majority of the results were excluded due to the occurrence of the search terms only in the meta-data, name of the conference, the title of the book containing the publication, or in the

list of references, and not in the body, or there was no correlation between the two concepts (i.e., a violation of the third inclusion criterion, e.g., different meanings of the words “goal” and “context” were used, words “reuse” and “context” referred to other concepts in the publication).

#### **Second Phase**

Answering the research questions requires focusing on the publications that contain a goal model reuse approach. Therefore, the 74 publications from the first phase related to goal model reuse are categorized according to their content in the second phase. The complete categorization and the full list of publications in each category is presented in Table 3.3. This categorization is not based on a predefined set of labels, but rather emerged from the examination of the publications:

- 36 contain a goal model reuse approach,
- 22 use goal modeling in a wider context, as part of a proposed approach, and do not focus or elaborate on goal models and their reuse,
- 10 introduce a new goal modeling language or an extension to an existing one but reusability is not investigated and a reuse approach is not specified,
- 2 propose an adaptation of an existing goal modeling notation to a specific domain, while not focusing on reuse, and
- 4 provide a comparison of approaches or an assessment of existing goal modeling languages.

A similar categorization is performed for the 107 publications from the first phase related to context and the complete categorization with the full list of publications in each category is presented in Table 3.4:

### 3.1. Methodology

Table 3.3: Second Phase Categorizations for Goal Model Reuse

Category Description	Publications in the Category
Contains a goal model reuse approach	[28][29][31][43][44][45][46][47][48][49][50][51] [52][53][54][55][56][57][58][59][60][61][62][63] [64][65][66][67][68][69][70][71][72][73][74][75]
Use goal modeling in a wider context, as part of a proposed approach, and do not focus or elaborate on goal models and their reuse	[18][76][77][78][79][80][81][82][83][84][85][86] [87][88][89][90][91][92][93][94][95][96]
Introduce a new goal modeling language or an extension to an existing one but reusability is not investigated and a reuse approach is not specified	[97][98][99][100][101][102][103][104][105][106]
Propose an adaptation of an existing goal modeling notation to a specific domain, while not focusing on reuse	[107][108]
Provide comparison of approaches or assessment of existing goal modeling languages	[109][110][111][112]

- 65 introduce an approach for modeling contextual information in goal models,
- 24 use one of the contextual goal modeling approaches in the first category as is,
- 11 use goal models in a wider context, as part of a proposed approach, and do not focus or elaborate on how context is/should be represented in goal models,
- 7 provide a comparison of approaches or an assessment of existing contextual goal modeling approaches.

The complete sets of 74 and 107 publications that were moved on to the second phase were categorized together by the author and the supervisor through a discussion session, i.e., each author individually assessed the publications, and the results were then compared and discussed until consensus was reached.

Table 3.4: Second Phase Categorizations for Contextual Goal Models

Category Description	Publications in the Category
Introduce an approach for modeling contextual information in goal models	[62][75][97][113][114][115][116][117][118][119][120][121] [122][123][124][125][126][127][128][129][130][131][132][133] [134][135][136][137][138][139][140][141][142][143][144][145] [146][147][148][149][150][151][152][153][154][155][156][157] [158][159][160][161][162][163][164][165][166][167][168][169] [170][171][172][173][174]
Use one of the contextual goal modeling approaches in the first category as is	[46][89][90][175][176][177][178][179][180][181][182][183] [184][185][186][187][188][189][190][191][192][193][194][195]
Use goal models in a wider context, as part of a proposed approach, and do not focus or elaborate on how context is/should be represented in goal models	[196][197][198][199][200][201][202][203][204][205][206]
Provide comparison of approaches or assessment of existing contextual goal modeling approaches	[207][208][209][210][211][212][213]

The comparisons of approaches and assessments of existing goal modeling languages are investigated further to check whether the research questions posed by this systematic literature review have been addressed before (see Section 3.2.1 for the 4 comparisons from the search for goal model reuse and Section 3.2.1 for the 7 comparisons from the search for context).

The 36 publications on a goal model reuse approach and the 65 publications on a contextual goal modeling approach are analyzed further in Section 3.2.2 to address the research questions identified in Section 3.1.1. Note that two publications appear in both result sets, i.e., the total number of publications analyzed further is hence 99.

The remaining publications are discarded (i.e., 34 related to goal model reuse and 35 related to contextual goal models). A complete overview of the results of the systematic review process is given in Figure 3.1.

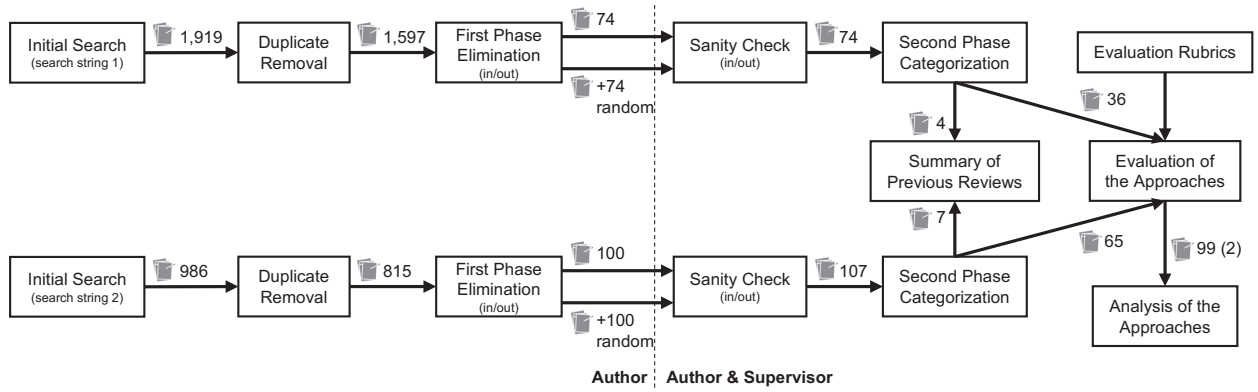


Figure 3.1: Summary of the review process

### 3.1.3 Threats to Validity

Threats to the validity of the systematic literature review are discussed following the classification of threats proposed by Petersen and Gencel [214] according to the definition of Maxwell [215].

#### Descriptive Validity

Descriptive validity refers to the factual accuracy of the research and its ability to correctly explain objective/subjective truth [214].

A threat to descriptive validity in this work is that the first phase assessment is performed predominately by the author, which could lead to bias.

As countermeasures, the supervisor performed the same assessment on a smaller, randomly generated set to check and verify the accuracy of the decisions made by the author as explained in detail in Section 3.1.2. Results of this verification showed that for both search strings, the complete sets of 74 and 100 publications that were included by the author were moved to the second phase. Only 7 new publications for the second search string were included by the supervisor, where only 2 out of the 7 did not have any related publications

included initially (i.e., 5 of the 7 publications already had similar publications included by the author).

Furthermore, clear inclusion criteria are defined and agreed upon by both the author and the supervisor to reduce room for interpretation, and all decisions made in the process are documented. The state of the publication set at each step is stored transparently with the help of a reference management tool [42].

To not risk missing an approach that might be relevant to the purpose of the literature review, all of the publications that fit the inclusion criteria and contain a clear correlation between the concepts used in the search query are taken into consideration for the second phase.

#### **Theoretical Validity**

Investigation of confounding factors and the capability of the researcher to capture what was intended, addresses theoretical validity [214].

A threat to theoretical validity is conducting the search in two stages through extending the search for goal model reuse approaches with context and goal modeling to further investigate the extent to which existing goal modeling approaches satisfy the fourth requirement (i.e., R.4).

The reason why no other tailored search strings were used for the other requirements (R.1-3 and R.5) is due to our experience in the area of goal model reuse regarding those four requirements compared to our more limited experience with context aware goal models. Our intuition was that reuse is not explicitly stated by the authors working on context and it turned out to be accurate considering that there are only three rows in Table 3.5 (the results related to search string 1) related to search string 2.

Another threat to theoretical validity arises from the search terms not being sufficient.

To address this threat, the search term is varied to be inclusive for possible variations of the use of terms (e.g., terms such as context-aware are included both with dash and space in the search phrase). Furthermore, seven different academic search engines are used to keep the sample size big enough in addition to defining the inclusion criteria in a broad way. Snowballing could have also been used as a countermeasure for this threat, but was not performed because of the broadness of the search terms and inclusion criteria.

#### **Generalizability**

Generalizability is concerned with the degree to which the results of the research can be generalized to different situations [214].

One of the threats to generalizability is the inclusion of the first 1,000 results out of 2,100 according to the relevance ranking of Google Scholar, for the search for goal model reuse.

To address this threat, all of the search results from the remaining six academic search engines are taken into consideration. Furthermore, to determine the likelihood of missing a relevant goal model reuse approach, we look at the 15 publications in the final publication set that only appeared in Google Scholar results. 13 out of 15 publications (i.e., 87%) are among the first 600 hits and the 15th publication is still in the first 900 hits. Therefore, it is not very likely that a relevant publication is not among the top 1,000 hits.

To make the results of this review more generalizable, the publication set is kept broad by conducting separate searches for reuse and context.

#### **Interpretive Validity**

Interpretive validity refers to the investigation of conclusions drawn to see whether they are reasonable according to the data depicting objective/subjective truth [214].

Formalization of the inclusion criteria and categorization rules is done by both the author



and the supervisor to address the threat to interpretive validity and to prevent bias in the evaluation of the approaches. Furthermore, the result sets at each step are verified by the supervisor.

#### **Repeatability**

Repeatability (or reproducibility/dependability) relies on a clear definition of steps followed for the research to be repeatable and is argued to follow from the previous categories listed [214].

To make sure that it is possible to repeat this systematic literature review, following the same guidelines and reproducing the same results, all steps followed at each phase of the review are diligently recorded. A reference management tool [42] is used to keep track of all of the decisions made and consequently the results of each step are clearly traceable.

## **3.2 Results and Analysis**

This section presents the results of the second phase categorizations and analysis of the approaches.

A summary of the 11 publications that provide a comparison of approaches or an assessment of existing *(i)* goal modeling languages and *(ii)* contextual goal modeling approaches is given in Section 3.2.1. This is followed by an analysis of the 99 publications in the combined, final publication set (2 publications appear in both sets of 36 and 65 publications) in Section 3.2.2 to answer the research questions Q.1 to Q.4 from Section 3.1.1. Finally, a discussion of future research themes (research question Q.5) is provided in Section 3.2.3.

### 3.2.1 Summary of Previous Reviews

According to the guidelines proposed by Kitchenham [41], a summary of previous reviews is needed for the justification of the need for the review. Therefore, we review existing comparisons and assessments related to goal model reuse and contextual goal models and present our analysis on them.

#### Reviews of Goal Model Reuse

A comparison of goal based (i\*-based) and use-case based approaches in terms of requirements reuse, reuse of provided artifacts in a later phase, and the types of reusable artifacts for security requirements modeling [110] concludes that for i\*-based approaches, the provided artifacts are not reusable in later phases. However, most of the ideas and concepts used during the early and late requirements phases are considered reusable and classified as consistent, modifiable, and appropriate for requirements reuse.

Comparison of i\* and KAOS in terms of quality [112] concludes that both languages and tools suggest means for reuse, yet they lack methodological guidelines and their tools are not sufficient to ensure the semantic quality of models. Semantic quality examines the correspondence between the information captured by the model and the modeled domain [216]. i\* models are considered difficult to reuse [111] due to their reusability being confined to a basic copy and paste of designated elements. Similarly, the evaluation of KAOS and its associated tool Objectiver [109] points out that KAOS does not provide support for libraries of goal models, and hence reuse.

To summarize, existing literature reviews in the area of goal model reuse show that, despite knowledge and expertise encapsulated in goal models being valuable and reusable, existing approaches that promote goal model reuse fall short of producing fully reusable artifacts in the end. Furthermore, the body of work targeted by the previous reviews covers

only a limited portion of the corpus of goal model reuse. Therefore, this systematic literature review is novel for exploring reusable goal models that support key characteristics of reusable artifacts as discussed in literature and from our experience.

#### Reviews of Contextual Goal Models

Modeling and using contextual information in model driven engineering (MDE) practices has always received attention from the requirements engineering community. Several approaches have been proposed and used for the representation of context in goal models in the literature. As a result of this, substantial research has been done on the evaluation and comparison of those approaches, taking multiple aspects of modeling in consideration.

Paja et al. [212] show, through a comparative study, that goal modeling is insufficient to reason about the best alternative without the capability for context modeling as the optimal decision varies depending on the context.

Kolos-Mazuryk et al. [211] analyze how the Goal-Based Requirements Analysis Method (GBRAM) [217], KAOS [218], and  $i^*$  [219] allow modeling of system context through a comparison of three goal-oriented requirements engineering (GORE) methods from the perspective of pervasive services and context modeling. Their results on context modeling affirm that, among the three, only  $i^*$  partly covers representation of the context and identification of context attributes. Context attributes whose values may change, activities that may initiate the aforementioned attribute changes, actors that perform the activities, and correlations between context changes are considered necessary for pervasive systems.

In a study, Alegre et al. [207] conduct a survey targeted at the development of context-aware systems, methodologies used, challenges involved, in addition to the reasons for their lack of adoption. They investigated requirements elicitation techniques that have been tailored for meeting the needs of context-aware system development. Adaptive and goal oriented

requirements elicitation techniques are claimed to provide means to capture and analyze variability and open the way to requirements evolution with the disadvantage of not being able to easily detect when the system behavior meets the requirements.

Requirements engineering practices also play an important part for self-adaptive systems (SAS), which are context-aware. Soares et al. [213] provide a comparative study to evaluate the support of goal modeling approaches for self adaptive systems. The goal-oriented approaches studied are Tropos4AS [220], Adaptive RML [148], and Design Goal Model (DGM) [221]. These three approaches are not used for general purpose goal modeling, but are developed to allow modeling the requirements of SAS. As the name suggests, a self-adaptive system needs to decide whether it is necessary to adapt to a change in system's context. The comparative study concludes that the three approaches have difficulties in representing the mechanisms for the adaptation and it is not clear whether they support modeling the evolution of the system.

Various modeling notations exist for modeling the system requirements and as a consequence of this, it is possible to observe the need for representing contextual information during requirements modeling in other modeling notations, too. Systematic reviews targeting variability modeling in dynamic software product lines (DSPL) highlight one common message, that the dominant modeling notation in the field is feature models. Guedes et al. [210] presents the results of a systematic mapping study to determine how variability is modeled in DSPL approaches, where two thirds of the investigated publications use feature models. Goal models are claimed to describe the variability in terms of alternatives to stakeholder or system goals.

A ranking of DSPL modeling techniques [208] places a technique using Tropos goal models integrated with context information [115] in the second place, being the only goal modeling notation among ten modeling techniques included in the survey. Shortcomings of the goal modeling technique are stated as the lack of a real case study, not using multiple layers of

abstraction, not completely supporting constraints (aside from dependency links), not supporting modeling unforeseen adaptations at development time, and not supporting decision point activation and modification. A further evaluation of variability modeling techniques for DSPLs [209] states that, although the technique using Tropos goal models integrated with context information [115] is less effective than the Context-aware Feature Model technique [222] in terms of precision, both techniques are considered effective in terms of recall.

In summary, existing literature reviews in the area of contextual goal models show that the existing body of work, although rich, is composed of research in different application domains and hence often restricted to the approaches in that particular domain. The systematic literature review performed in this work is novel for targeting specifically goal modeling and how contextual goal models are/can be used in the context of reuse and in the presence of reuse hierarchies.

#### 3.2.2 Analysis of the Approaches

The 36 publications on goal model reuse approaches and the 65 publications on contextual goal modeling approaches are taken into further consideration as described in Section 3.1. As 2 of those publications [62][75] appear in both final sets, the combined 99 publications are grouped under the name of the approach they present and compared in Tables 3.5-3.7. This resulted in 20 approaches for goal model reuse (Table 3.5) and 26 for contextual goal models (Table 3.6 and 3.7). However, early publications related to our approach are among the 99 publications. Since our approach will not be discussed in this section as it will be presented throughout the following chapters, only 19 out of the 20 approaches for goal model reuse are analyzed in this section.

The column labeled “*Search String*” shows in which part of the two-part search the approach appeared (i.e., 1 corresponds to the search for goal model reuse, 2 corresponds to

### 3.2. Results and Analysis

Table 3.5: Summary of Goal Model Reuse Approaches

Approach	Search String	R.1	R.2	R.3	R.4	R.5	Context Representation	Supplementary Models	Tool Support
GRL Catalogues [43]	1	●	●	○	●	●	strategies and beliefs	use case maps	●
i* Modules [44]	1	●	●	○	○	●	-	-	●
STREAM-A [45][46]	1, 2*	●	○	●	●	●	context annotations	acme	F
AoGRL [47][48][49][50][51]	1	●	●	●	●	●	strategies and beliefs	use case maps	F
Goal Aspects [52][53]	1	●	●	●	○	●	-	sequence d.	●
Aspects with Tropos [54]	1	F	○	●	○	●	-	use case maps	F
Aspects with i* [55]	1	F	○	○	○	●	-	-	F
Aspects with V-graph [56][57]	1	○	○	○	●	●	topics	-	F
AspectKAOS [58][59]	1	○	○	○	○	●	-	-	F
ExtendedKAOS [60][61]	1	○	○	●	○	○	-	responsibility, object, operation	●
Security Patterns [62][63][113][114]	1, 2	●	●	○	●	●	domain properties	-	●
GoPF [64][65][66][67]	1	●	●	●	●	●	strategies	use case maps	F
Task Knowledge Patterns [68]	1	○	○	●	○	●	-	role model, organization m.	○
Patterns in i* [69]	1	●	●	○	○	●	-	-	○
Attack Patterns [70]	1	○	○	○	○	●	-	-	F
Patterns in NFR [71]	1	●	●	●	○	●	-	use case model, sequence d.	F
GOPCSD [72][73]	1	●	○	●	○	●	-	state diagram	●
Variability in Goal Models [74]	1	●	●	○	○	●	-	-	F
Context Transformations [75]	1, 2	○	○	○	○	●	-	-	F
Reusable Goal Models [28][29][31]	1	This approach will be discussed in Chapters 4 to 6.							

\* not categorized as one of the 65 contextual goal modeling approaches but as one of the 24 that use a contextual goal modeling approach as is

### 3.2. Results and Analysis

Table 3.6: Summary of Contextual Goal Modeling Approaches

Approach	Search String	R.1	R.2	R.3	R.4	R.5	Context Representation	Supplementary Models	Tool Support
Contextual Goal Models [97][115][116][117][118][119][120][121][122][123][124][125]	2	●	●	●	●	○	decompositions, dependencies, contributions, goal activation	feature m. & problem frames	●
FLAGS [126][127][128]	2	●	●	○	●	○	adaptive goals	-	F
Claim Refinement Models [129][130][131]	2	●	●	○	●	○	claims, contributions	-	●
Agents with Context-dependent Goal Models [132]	2	●	●	○	●	○	conditional goals, decompositions, contributions	-	○
Pragmatic CGM [133]	2	●	●	○	●	○	pragmatic goals (contextual interpretation)	-	F
Context-enriched Goal Models [134][135][137][138]	2	●	●	○	●	○	domain properties & contextual tags	-	F
Context Sensing Goal Models [136]	2	●	●	○	●	○	context variables	ERD, state d.	○
Extended Tropos Goal Model [139][140]	2	●	●	○	●	○	context conditions & internal event conditions	-	F
Goal-oriented Context Requirement Model [141][142][143]	2	○	●	○	●	○	context conditions, decompositions	process model	●
Xipho [144]	2	○	○	○	●	○	contextual beliefs and resources	-	○
URN [145][146][147]	2	●	●	●	●	○	strategies and KPIs	use case maps	●
Adaptive RML [148][149][150]	2	●	●	○	●	○	context and resources	-	●
AwReqs [151][152]	2	●	●	○	●	○	variation points, control variables, indicators, awareness requirements	-	●

### 3.2. Results and Analysis

Table 3.7: Summary of Contextual Goal Modeling Approaches (cont.d)

Approach	Search String	R.1	R.2	R.3	R.4	R.5	Context Representation	Supplementary Models	Tool Support
GCPD [153]	2	●	●	○	●	○	contextual factors & constraints	-	●
CARGO [154][155][156]	2	●	●	○	●	○	KPIs	use case maps	●
i*-Context [157]	2	●	●	○	●	○	agent roles and role playing relations	-	○
Adaptation Goal Model [158]	2	●	●	○	●	○	adaptation goals, atomic contexts	context model	○
Capability-Driven Development [159][160][161][162][163][164]	2	○	●	○	●	○	context element & KPI	-	●
SecuriTAS [165][166]	2	●	●	○	●	○	contextual factors	-	●
REFAS [167][168]	2	●	●	○	●	○	context variables & claims	-	●
RBUIS [169]	2	●	●	○	●	○	context element	-	●
Feature-Oriented NFRs [170]	2	●	●	○	●	○	feature context model	feature model	F
Tropos4AS [171]	2	●	●	○	●	○	goal conditions	environment m. & failure m.	●
Context Feature Model [172]	2	●	●	○	●	○	feature model	context feature m. & feature m.	○
Event-based Goal Adaptation [173]	2	●	●	○	●	○	event & indicator	-	●
G+ [174]	2	○	○	○	●	○	goal functional context	-	○



the search for context and goal models, and 1, 2 means the approach appeared in both).

The five columns labeled R.1 to R.5 correspond to the five requirements identified in Section 1.2 and show the degree of satisfaction of their corresponding requirement with respect to the predetermined evaluation criteria defined in Tables 3.8-3.12. These five columns address the research question Q.1 from Section 3.1.

If an approach contains modeling constructs to represent context in goal models, and/or makes use of other requirements modeling notations in collaboration with goal models, they are shown in the corresponding columns labeled “*Context Representation*” and “*Supplementary Models*”, respectively. The “*Context Representation*” column addresses research question Q.2 whereas the “*Supplementary Models*” column addresses research question Q.3 from Section 3.1.

The last column shows whether tool support exists for the corresponding approach or not (as shown in Table 3.13) in order to answer research question Q.4 from Section 3.1.

If a criterion in one of the columns is stated as future work in the publication, it is shown with the letter F.

In order to maintain the readability of the analysis, only the names of the approaches are used in the remainder of this section. For the references to their corresponding publications, the reader may refer to the “*Approach*” columns of Tables 3.5-3.7.

#### **Trade-off Reasoning**

As shown in the evaluation rubric for R.1 in Table 3.8, for an approach to fully satisfy the first requirement of providing support for analysis or validation through reuse hierarchy, the reasoning mechanism provided should also deal with complexity resulting from large reuse hierarchies.

In Tables 3.5-3.7, R.1 is partially satisfied by 33 approaches. Even though goal model eval-

Table 3.8: Evaluation Rubric for R.1

Criteria	Evaluation
Does not specify any means for analysis or validation of the goal model (i.e., reasoning mechanism).	○
States that a reasoning mechanism is planned for future work.	F
Describes a reasoning mechanism.	◐
For the described reasoning mechanism, also deals with complexity resulting from large reuse hierarchies.	●

uation is possible with 30 of these approaches (GRL Catalogues,  $i^*$  Modules, AoGRL, Goal Aspects, Security Patterns, Patterns in  $i^*$ , Patterns in NFR, GOPCSD, Contextual Goal Models, FLAGS, Claim Refinement Models, Agents with Context-dependent Goal Models, Pragmatic CGM, Context-enriched Goal Models, Context Sensing Goal Models, Extended Tropos Goal Model, URN, Adaptive RML, AwReqs, GCPD, CARGO,  $i^*$ -Context, Adaptation Goal Model, SecuriTAS, REFAS, RBUIS, Feature-Oriented NFRs, Tropos4AS, Context Feature Model, and Event-based Goal Adaptation), the absence of a reuse hierarchy (reuse is done via copy and paste operation) leads to one big monolithic goal model which, in turn, is complex to evaluate when considering additional constraints imposed by other modeling notations on the goal model.

The STREAM-A approach modularizes  $i^*$  models via grouping the model elements into actors that can be reused in other domains. The GoPF approach also defines the boundaries of the reusable artifacts (goal templates) well. The Variability in Goal Models approach uses goal models in the decision-making process of choosing variants and divides a system into smaller aspects. These three approaches allow the evaluation to be done on the composed model, although not incrementally; due to which, the complexity issue persists.

Two approaches, Aspects with Tropos and Aspects with  $i^*$  state that a reasoning mechanism is planned for their future work while the remaining ten approaches do not specify any

Table 3.9: Evaluation Rubric for R.2

Criteria	Evaluation
Does not specify any means for delaying decisions.	○
States that support for delaying decisions is planned for future work.	F
Describes means to delay decisions and enable representation of uncertain input values.	◐
Incorporates delayed decisions and associated uncertainty into the composition and evaluation of reusable artifacts.	●

means for analysis or validation of the goal model.

### Delaying Decisions

In order to satisfy the second requirement of providing the means to delay decisions to a later point in the reuse hierarchy, a reusable goal modeling approach shall incorporate delayed decisions and associated uncertainty into the composition and evaluation of reusable artifacts.

As shown in the evaluation rubric in Table 3.9, not being able to make use of the uncertainty represented in a goal model in the composition and evaluation, results in the partial satisfaction of this requirement.

Consequently, the second requirement of providing support for delaying reuse decisions to a later point (R.2) is partially satisfied by 33 approaches as they allow use of data obtained at runtime for re-evaluation of goal satisfactions or system re-configuration (Contextual Goal Models, FLAGS, Claim Refinement Models, Pragmatic CGM, Context Sensing Goal Models, Extended Tropos Goal Model, Goal-oriented Context Requirement Model, URN, Adaptive RML, GCPD, CARGO, i\*-Context, Adaptation Goal Model, Capability-Driven Development, SecuriTAS, REFAS, RBUIS, Tropos4AS, Context Feature Model, and Event-based Goal Adaptation), or use qualitative reasoning to cope with uncertainty (GRL Catalogues,

Table 3.10: Evaluation Rubric for R.3

Criteria	Evaluation
Does not mention any other modeling notation that imposes external constraints on goal models.	○
States that support for external constraints is planned for future work.	F
Describes means to capture constraints imposed by other modeling formalisms on goal models	◐
Takes external constraints into account for the evaluation of goal models.	●

i\* Modules, AoGRL, Goal Aspects, Security Patterns, GoPF, Patterns in i\*, Patterns in NFR, Variability in Goal Models, Agents with Context-dependent Goal Models, Context-enriched Goal Models, AwReqs, and Feature-Oriented NFRs). However, these approaches do not address reuse hierarchies, require knowledge of the whole system, and hence do not explicitly specify mechanisms to delay decision making to a later point in the reuse hierarchy (e.g., indicating which decisions still need to be taken or are allowed to be delayed and the impact of doing so).

### External Constraints and Supplementary Models

In this section, the satisfaction of the third requirement is discussed together with the supplementary models used by goal modeling approaches. The rubric used in the evaluation of the third requirement of taking constraints imposed by other modeling notations into account (R.3) is shown in Table 3.10.

The column labeled *Supplementary Models* in Tables 3.5-3.7 addresses research question Q.3 from Section 3.1 and is discussed together with the models that are used in collaboration with goal models in this section.

The third requirement (R.3) is partially satisfied by 11 approaches. STREAM-A uses i\* models in collaboration with Acme architectural models. While Goal Aspects approach

explicitly use sequence diagrams, Patterns in NFR use both use case and sequence diagrams in collaboration with goal models. Aspects with Tropos explicitly allow modelers to express workflow models with UCM. Responsibility, object, and operation sub-models of KAOS are also used in the ExtendedKAOS approach. Task Knowledge Patterns use role and organizational models whereas the GOPCSD approach uses state diagrams to capture external constraints. Contextual Goal Models on the other hand, allow modelers to use feature models and problem frames to complement goal models. AoGRL, GoPF, and URN use links between UCMs and GRL goal models to capture constraints. However, these approaches do not leverage the constraints imposed by the other modeling languages in the evaluation of the goal model, which prevents the full satisfaction of the third requirement.

This requirement is not satisfied by any approach (i.e., no publication explicitly mentions any other modeling notation that imposes constraints on goal models and their evaluation).

In addition to the 11 approaches that partially satisfy the third requirement (R.3), 8 of the approaches in the final set make use of supplementary models in addition to their goal models. Although the supplementary models used by these 8 approaches do not impose any external constraints on goal models, it is important to highlight them to answer the third research question (i.e., What other requirements models, if any, are used in collaboration with reusable goal models?).

Overall, behavioral models in the form of workflow models (i.e., UCMs and process models), sequence diagrams, and state diagrams make up the majority of the supplementary models. Furthermore, several structural modeling notations such as role models, organization models, use case models, Entity Relation Diagrams (ERD), environment and failure models, and context models are also used in collaboration with goal models.

Table 3.11: Evaluation Rubric for R.4

Criteria	Evaluation
Does not specify any means for modeling context dependent information.	○
States that support for context dependent information is planned for future work.	F
Describes means to capture contextual information.	◐
Uses explicit language constructs to represent context related information.	◑
Uses contextual information that is explicitly modeled during the analysis of the goal model.	●

### Contextual Information

The fourth requirement to allow context dependent information to be modeled (R.4) is evaluated according to the rubric given in Table 3.11.

A difference in the rubric for this requirement is the use of three-quarters (◑) as a measure of partial satisfaction. This emerges from the need for a distinction between the approaches that use explicit modeling constructs to capture contextual information and those that do not. For the complete satisfaction of this requirement, an approach is required to use explicitly captured contextual information in the analysis of the goal model.

GRL possibly captures contextual information in strategies and beliefs without explicitly defining the concept of context. Therefore, GRL Catalogues, AoGRL, and GoPF do not meet the three-quarters partial satisfaction criterion.

Six of the approaches meet the three-quarters partial satisfaction criterion whereas the fourth requirement is fully satisfied by Security Patterns from the search for goal model reuse (i.e., search string 1 in Table 3.5), in addition to 22 of the 26 approaches that appeared in the search for context and goal models (i.e., search string 2 in Tables 3.6-3.7).

As shown in the column labeled “*Context Representation*” to address research question Q.2 from Section 3.1, these approaches provide the following constructs for modeling con-

textual information. As one of the six approaches that do capture context explicitly without incorporating this information into analysis, STREAM-A adopts the context representation of Contextual Goal Models with their use of context annotations. Aspects with V-graph uses the concept of topic in V-graph model to capture contextual information for goals, tasks, and softgoals whereas Goal-oriented Context Requirement Model uses a goal refinement model with and/or decompositions and context conditions. Xipho, an agent-oriented methodology for engineering context-aware personal agents (CPA), extends Tropos with contextual beliefs and resources for the representation of contextual information and to support CPA development. Capability-Driven Development uses context KPIs in their run-time adjustments whereas G+ makes use of goal functional context to represent context in the form of pre/post-condition constraints for goals.

Out of the 23 approaches overall that fully satisfy the fourth requirement, Security Patterns extend and use Contextual Goal Models for their context representation and specify context in terms of domain properties (i.e., facts related to a particular domain that are used to evaluate values of context tags assigned to model elements). On the other hand, Contextual Goal Models identify variation points in their goal models as or-decomposition, means-end, actor dependency, root goals, and-decomposed goals, and contribution to softgoals.

FLAGS adds adaptive goals to KAOS models, specifying how the system should handle its self-adaptation through application of a suitable countermeasure while Pragmatic CGM introduces pragmatic goals that have a dynamic satisfaction criterion and an algorithm for the evaluation of their achievability.

Claim Refinement Models use claims and changing contribution values whereas Agents with Context-dependent Goal Models use conditional goals, decompositions, and contributions to answer whether agents can support their goals within a given context.

The Context-enriched Goal Models approach defines a set of contextual tags and is con-

cerned with the visibility of model elements, whereas the Context Sensing Goal Models and REFAS approaches identify context variables that are later utilized in order to achieve an objective. Extended Tropos Goal Model introduces context conditions and internal event conditions to address the shortcomings of Tropos goal model in specifying software variations.

URN makes use of key performance indicators (KPI) and beliefs for the representation of context, whereas the AwReqs approach introduces variation points and control variables that carry resemblance to KPIs to their models. Context-aware Reasoning Using Goal-orientation (CARGO) also uses KPIs while Event-based Goal Adaptation uses events together with indicators.

Adaptive RML builds on the abstract early-requirements modeling language *Techne* [223] by adding the concepts of context and resource. Goal-based Contextual Problem Detection (GCPD) and SecuriTAS attempt to detect contextual changes via defining contextual factors while the RBUIS approach uses context elements that are monitored during execution. Another approach focusing on adaptive behavior, the self-adaptation framework for Tropos (Tropos4AS) uses goal conditions whereas Adaptation Goal Model makes use of adaptation goals and atomic contexts to represent context dependent information.

The Feature-Oriented NFRs and Context Feature Model approaches capture context with the help of feature models where Feature-Oriented NFRs take a step further to incorporate NFRs into the feature model. Lastly, *i\**-Context extends *i\** with categorization of agent roles and role playing relations for the representation of context dependent information.

The remaining 13 approaches do not specify any means for modeling context dependent information. The fact that all of these 13 approaches appear in the result set of the search for goal model reuse (i.e., search string 1) reconfirms the need for the search for context and goal models (i.e., search string 2) and the lack of context representation in goal modeling approaches predominantly focusing on reuse.



Table 3.12: Evaluation Rubric for R.5

Criteria	Evaluation
Does not specify any means to define module boundaries (e.g., reuse interfaces).	○
States that support for clear module boundaries is planned for future work.	F
Describes clear module boundaries.	◐
Maintains the modularity and clear model boundaries after reuse of the goal model.	●

### Reuse Interface

In order to fully satisfy the last requirement of providing an interface for reuse (R.5), an approach shall maintain the modularity and clear model boundaries even after reuse of the goal model. Therefore, loss of module boundaries upon reusing the goal model results in only partial satisfaction of this requirement as shown in Table 3.12.

As shown in Tables 3.6 and 3.7, none of the contextual goal modeling approaches except the two in common for both searches (i.e., Security Patterns and Context Transformations in Table 3.5) focus on defining model boundaries and hence fail to satisfy this requirement.

The fifth requirement is partially satisfied by thirteen of the approaches (GRL Catalogues, i\* Modules, AoGRL, Goal Aspects, Aspects with V-graph, AspectKAOS, Security Patterns, Task Knowledge Patterns, Patterns in i\*, Attack Patterns, Patterns in NFR, GOPCSD, and Context Transformations), because, even though they provide the necessary module boundaries (e.g., actor, pointcut and advice, template, etc.), modularization and hence the reuse hierarchy is lost upon composing the models and it is not possible for the user of the reusable artifact to identify model elements at different levels after that point.

Overall, five approaches fully satisfy the fifth requirement. STREAM-A groups i\* model elements into actors so that the actor can be reused in another domain, and hence maintains

Table 3.13: Evaluation Rubric for Tool Support

Criteria	Evaluation
Does not provide tool support.	○
States that tool support is planned for future work.	F
Describes tool support.	●

the boundaries of an actor. Two approaches to create modular i\* and Tropos models with the help of Aspect Oriented Software Development, Aspects with i\* and Aspects with Tropos, introduce *aspects* and *aspectual elements* for this modularization. Similar to the use of actors, intentional elements are contained in these aspects and aspectual elements that can be composed through crosscut relationships. As previously mentioned for R.1, goal templates used in the GoPF approach maintain clear model boundaries after they are reused and hence this approach satisfies R.5. Composition mechanism used in the Variability in Goal Models approach keeps the modules separate and uses composition labels to attach models.

## Tool Support

The last columns of Tables 3.5-3.7 address research question Q.4 from Section 3.1 (based on the evaluation rubric in Table 3.13) and provide some evidence that tool support is considered important in the modeling community as 20 out of 45 approaches provide tool support and a further 16 out of 45 state it as their future work.

### 3.2.3 Future Research Themes

The results of the systematic literature review point to four research themes to enhance the capabilities of existing goal modeling approaches with more advanced support for reuse and context dependent information.

**T.1)** The R.3 columns in Tables 3.5-3.7 show that very little support is available to take into account additional constraints imposed by models that are connected to goal models but expressed with different modeling formalisms. These constraints may significantly influence goal model evaluation as they may exclude otherwise permissible candidate solutions. Hence, goal model reuse approaches need to be developed that fully satisfy R.3.

**T.2)** Based on the results in the R.5 columns in Tables 3.5-3.7, it seems that the goal modeling community has not yet converged on a generally accepted definition of what constitutes the reuse interface of a goal model.

Only very few approaches consider the definition of an explicit reuse interface to encapsulate internal goal model structures [45][46][54][55][64][65][66][67][74].

An analysis of existing, common goal model constructs should be performed to determine which constructs may be visible to a reusing party and which should not be visible. Techniques should be developed that govern under which circumstances the visibility of a modeling element is allowed to change. E.g., what happens to the visibility of modeling elements of a reused goal model in the reusing model?

**T.3)** None of the existing goal modeling approaches provides a complete mechanism to delay decisions in the reuse hierarchy, as indicated by the R.2 columns in Tables 3.5-3.7.

Existing approaches go as far as enabling the specification of where and when decisions may be delayed or by supporting the re-evaluation of goal models at runtime to reassess a possibly changed reuse context. However, delaying of decisions introduces an element of uncertainty or variability that needs to be considered when evaluating goal models to determine the best trade-off for stakeholders. E.g., instead of dealing with concrete results for a goal model evaluation, possible ranges of evaluation values may have to be considered.

New evaluation mechanisms need to be developed for the various goal modeling ap-

proaches investigated by this systematic literature review to address these issues.

**T.4)** The overall challenge is to provide a holistic goal modeling environment that supports all five requirements stated in Section 1.2.

Various goal modeling approaches address individual requirements. Many different goal modeling approaches support context dependent information as indicated in the R.4 columns in Tables 3.5-3.7, using a variety of modeling constructs to express context as shown in the “*Context Representation*” columns in Tables 3.5-3.7. Despite the close relationship between reuse and context, there exists very little overlap between the two concepts (as indicated by only three overlapping approaches appearing in both result sets). Therefore, it is not clear which of the modeling constructs to express context is the most appropriate for reuse hierarchies, for which some support exists as shown in the R.1 columns in Tables 3.5-3.7. However, while some of these contextual modeling approaches employ a reasoning mechanism, none of them fully deal with complexity resulting from large reuse hierarchies and provide feasible support to analyze context dependent information in reusable goal models across such hierarchies.

## 3.3 Summary

The methodology and results of the two-part systematic literature review performed to investigate the current landscape of goal modeling approaches and the extent of their conformance to the five requirements identified in the problem statement of this thesis is demonstrated in this chapter.

The required characteristics are (i) to enable analysis and validation of goal models through reuse hierarchies, (ii) to provide the means to delay decision making to a later point in the reuse hierarchy, (iii) to take constraints imposed by other modeling notations

into account during goal model analysis, (iv) to allow context dependent information to be modeled so that the goal model can be used in various reuse contexts, and (v) to provide an interface for reuse.

The two-part systematic literature review covers both reuse and context in goal models. Research questions and inclusion criteria are specified, as well as threats to validity are discussed. Each step of the review is documented, and all results are tracked with the help of a reference management tool.

While the initial publication list retrieved from seven major academic search engines included more than 2,400 unique hits, a final list of 99 publications and 11 comparisons/assessments of goal modeling approaches are discussed in more detail.

Based on the examination of these publications, four research themes are derived to realize reusable goal models with context dependent information: (i) development of goal model reuse approaches that deal with constraints imposed on goal models by other models, (ii) definition of a generally accepted reuse interface for goal models, (iii) development of mechanisms in goal models to support delaying of decision in reuse hierarchies, and (iv) development of a holistic goal modeling environment to support all five characteristics of reusable and contextual goal models.

The following chapter investigates different kinds of contribution values that may be used in reusable goal models that are evaluated with a propagation-based algorithm.

# Chapter 4

## Relative Contribution Values

In this chapter, we investigate different means to assign contribution values in goal modeling and motivate the need for a new approach to assign contributions in reusable goal models.

We identify the shortcomings of existing types of contribution values in the context of reuse in Section 4.1 and introduce relative contribution values to overcome the issues related to reuse of goal models in Section 4.2. We then discuss in Section 4.3 the interpretation of satisfaction values with relative contributions and introduce the normalization of goal satisfactions. Finally, we conclude the chapter with a summary in Section 4.4.

### 4.1 Contribution Values in Goal Modeling

The capability to perform trade-off analysis through goal model evaluation is one of the major strengths of goal modeling. This evaluation allows modelers to reason about system qualities, high-level goals, non-functional requirements, and solutions.

Furthermore, in the context of reuse, a goal model may be used to describe why a reusable artifact should be chosen over another candidate artifact, or why a variation offered by a reusable artifact should be chosen over another offered variation. Therefore, a goal model enables trade-off analysis among these choices by capturing the impact of a choice on high-level goals and system qualities. When a reusable artifact is assembled with other reusable artifacts in a reuse hierarchy, the goal model describing the reusable artifact must also be composed with the goal model of the other reusable artifact to enable reasoning about the

whole system.

A propagation-based evaluation mechanism, which propagates the satisfaction value of a child element up to the parent element according to the type and value of the link between these two elements, is often utilized by goal models. For example, in GRL, a *strategy* depicts a candidate solution via assigning initial satisfaction values to a set of elements in the model. A satisfaction value expresses the degree with which a softgoal or goal is satisfied, a task is performed, or a resource is available. Unless a satisfaction value of a node is assigned by the modeler with a strategy, the satisfaction value of a node is calculated with the help of propagation-based evaluation [5][22], using the incoming links (type of the link and the contribution value if it is a contribution link) and the satisfaction values of the children nodes. Therefore, modelers can compare several strategies with each other using the resultant satisfaction values of high-level goal model elements. This facilitates the decision making process among alternative solutions or prioritization of the requirements and/or stakeholder goals.

Contribution values and how they are assigned are an integral part of goal model evaluation. Three types of contribution values are known for main goal modeling approaches such as the NFR Framework, GRL, Tropos, and i\*: qualitative, quantitative, and real-life values. In the remainder of this section, we discuss these three types of contribution values and their usability in reusable goal models. The modeler who builds a reusable artifact is referred to as the *designer* and the modeler who uses the reusable artifact is referred to as the *user*.

For the illustration of goal models in this chapter, the GRL notation [7] is used. Although three different link types (i.e., contribution, decomposition, and dependency) exist in GRL, our research has focused on contribution links. Findings, however, can also be applied to decomposition and dependency links and the support for decomposition and dependency links is planned as part of the future work.

### 4.1.1 Qualitative Labels

Qualitative values are favorable for capturing relationships among choices at the early stages when there is not a lot known about the domain. They provide limited accuracy and do not allow nuanced differences among reusable alternatives to be modeled. Because qualitative values are typically quite broad (e.g., for positive impacts, only two labels - *Help* and *Make* - are defined in the NFR framework), a designer might be forced to classify many different choices the same using the same label (e.g., as *Help*).

As a result of this, a differentiation cannot be made by the evaluation algorithm among these choices, making it difficult for the user to determine the most appropriate choice. Since a carefully-built, reusable artifact is well-understood, qualitative values are not appropriate for a reusable artifact.

### 4.1.2 Real-life Measurements

Real-life measurements are the opposite extreme compared to qualitative values. Concrete, real-life measurements can guarantee consistent assessment and allow for a nuanced description of reusable artifacts.

If such measurements are available, then they should be used to describe reusable artifacts (e.g., cost can typically be straightforwardly measured in \$). However, it may not always be easy to obtain these measurements as it may be rather difficult to define appropriate units of measurements for some common high-level goals and system qualities. Although some high-level goals and system qualities can be measured unambiguously (e.g., cost), convenience, security, and privacy are examples where this is not the case.

Furthermore, even if a unit of measurement can be devised, it may be difficult or costly to collect the required data to calculate such measurements. Consequently, due to the problems with availability and feasibility of real-life measurements, an alternative is needed for those



cases where the use of real-life measurements is not feasible.

### 4.1.3 Quantitative Values

Quantitative values are the middle ground as they save the modeler from the cost and feasibility problems of real-life measurements while modeling nuanced differences among alternative solutions, and hence are the preferred approach when real-life measurements are not feasible.

Quantitative values can be defined within a specific range, typically  $[-100, 100]$ . However, there are two issues that need to be taken into account before quantitative values can be used for reusable goal models. First, the designer of a reusable goal model must provide reasonable values for the quantitative contributions. Secondly, the designer must be able to do so without making any assumptions about the application under development, which eventually will use the reusable goal model. As stated earlier, a reusable artifact must not know about application-specific elements because it has to be generic.

## 4.2 The Need for Relative Contribution Values

In the absence of objective real-life measurements, a designer of a reusable goal model has the choice of either using global or relative contribution values. In the case where global values are used, it is essential to keep those values consistent across all models, which requires an understanding among designers and users on what each value stands for. Since it is also necessary to use these contributions to reflect the impact of goal model elements on ambiguous qualities (e.g., convenience, security, privacy, usability, etc.), it is not likely to have such an understanding.

Consider the example shown in Figure 4.1, which shows a hierarchy of reusable goal

## 4.2. The Need for Relative Contribution Values

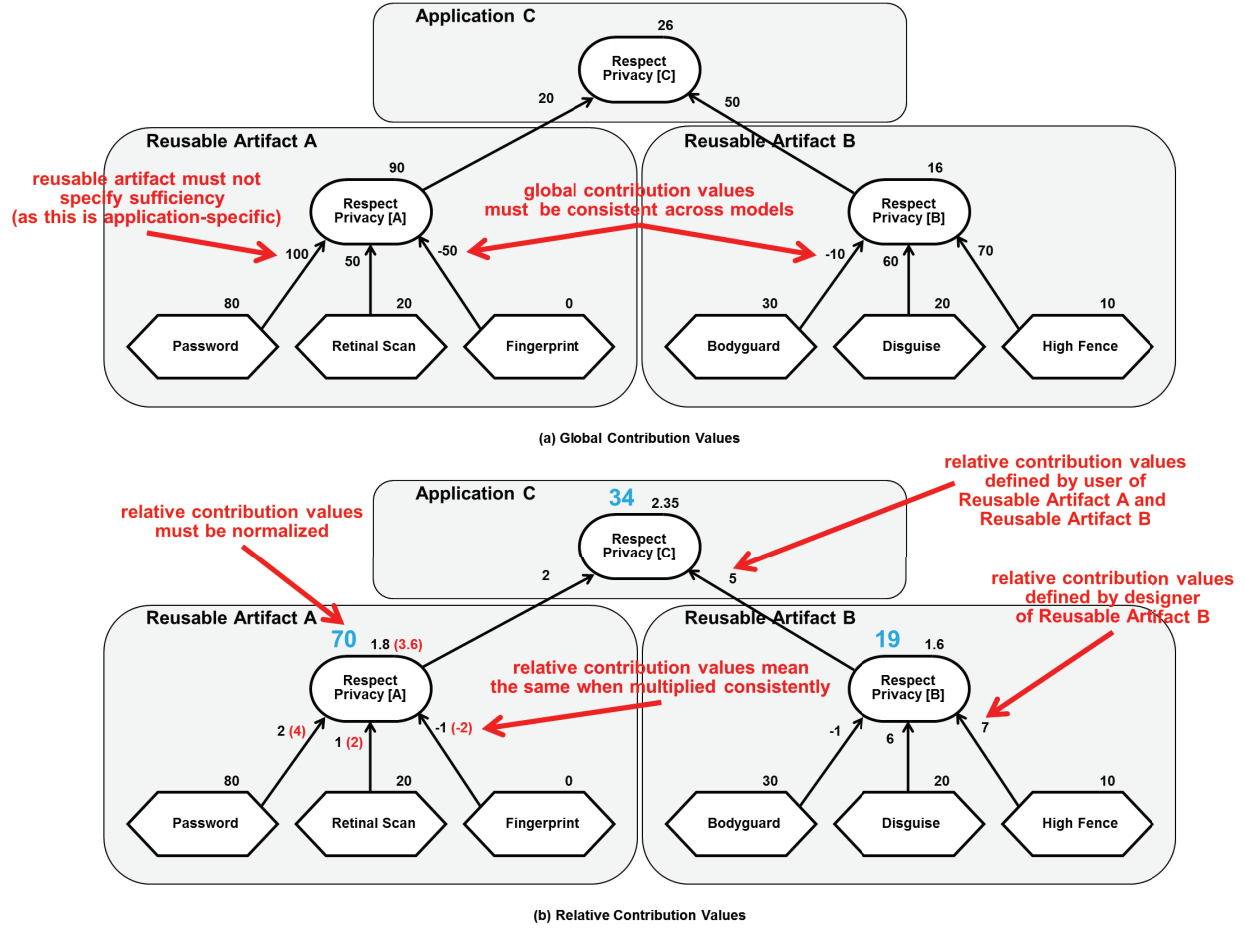


Figure 4.1: Global and relative contribution values in reusable goal models [24]

models with global contribution values at the top and the same hierarchy of goals with relative contribution values at the bottom.

The application under development reuses two goal models. **Reusable Artifact A** describes privacy implications of online security features, while **Reusable Artifact B** describes privacy considerations of famous people interacting with the public. Solutions offered by the reusable artifacts are shown with the tasks (e.g., **Password**, **Retinal Scan**, etc.), each having a different impact on privacy as shown on the contribution links. As the application

under development reuses both artifacts, it undertakes a more thorough assessment of the privacy of famous people, taking some of their online interactions into account.

As explained in Section 4.1.3, the first issue with quantitative contribution values is that the designer of a reusable goal model must provide reasonable values as contributions. If a designer uses global values, then the implications are that the designer of **Reusable Artifact A** and the designer of **Reusable Artifact B** must agree on rules regarding the assessment of contribution values to ensure consistency across all goal models for a specific quality. If not, then the impact on the quality (e.g., Privacy) may be inflated by one reusable artifact compared to the other, which makes it impossible to combine them into a goal model hierarchy for Privacy.

For example in Figure 4.1a, the designers would have to agree that **Password** contributes more (i.e., 100) to **Privacy** than **Disguise** (i.e., 60). In the absence of concrete units of measurements, an agreement is unlikely to be feasible, as each designer has expertise in their own domain and not necessarily in others. In the bigger picture, such agreement would require awareness of all designers building a reusable artifact that impacts Privacy.

The second issue with the use of quantitative contribution values relates to the concept of *sufficiency*.

Let us examine in more detail the contribution value of **Password**, which is assigned as 100. The contribution value indicates that choosing **Password** is sufficient to achieve the parent goal, i.e., **Respect Privacy**, in this case. If it were not sufficient, then a value lower than 100 would have to be chosen. However, this decision cannot be made by the designer of the reusable goal model, as it depends on the application context. While the **Password** solution may be perfectly acceptable for one context, it may not be sufficient in another context.

A generic reusable goal model cannot state that a particular solution is sufficient, because that would mean the designer of the reusable artifact is able to anticipate each and every use

of the reusable artifact. This, however, is not possible without violating the key characteristic of a reusable artifact being generic.

At the heart of this problem is the fact that a contribution value in goal models captures two dimensions: first, it defines the degree of impact on the parent goal, and second, it defines the threshold for what is sufficient to achieve the goal and what is not. In a reusable goal model, the former but not the latter must be specified.

Note that real-life measurements do not have this problem, because in that case the two dimensions have already been separated. The real-life measurement captures the degree of impact, while the conversion into a goal model value determines what is sufficient and what is not sufficient. Therefore, a designer of a reusable goal model can specify the real-life measurements and easily postpone the definition of the conversion function to the application under development and to the user of the reusable goal model. This separation is not possible for global contribution values where the two dimensions are combined into one value.

Relative contribution values address the aforementioned two issues. First, there is no need to coordinate an assessment with other designers, because the designer of the reusable artifact assigns relative contribution values with the sole intention of differentiating competing solutions locally within the reusable artifact. Relative contribution values only intend to differentiate the children of a parent goal (e.g., `Password`, `Retinal Scan`, and `Fingerprint` in Figure 4.1). This approach eliminates the need for a global agreement among modelers on what contribution values mean as is needed for non-relative contribution values. Therefore, the designers of `Reusable Artifact A` and `Reusable Artifact B` do not need to know about each other.

Secondly, a relative contribution value does not specify which level of impact is sufficient to achieve the parent goal. A relative contribution only states that a child contributes  $x$  times more than another child of the same parent goal. The decision of whether a contribution is sufficient can again be postponed to the user of the reusable artifact, who knows best what

is sufficient and what is not.

Fundamentally, relative contributions lead to a different way of thinking about reusable goal models, based on two distinct spheres of knowledge.

The first sphere of knowledge belongs to the designer of a reusable goal model, who focuses on understanding the impacts of the choices in the reusable goal model and ranking them relatively. The designer is the most qualified person for this ranking, as she is the domain expert for the reusable artifact (e.g., the designer can determine the relative privacy implications of the `Password`, `Retinal Scan`, and `Fingerprint` choices). Each designer of a reusable artifact has her own sphere of knowledge, which is limited to the reusable artifact.

The user of a reusable artifact has a different sphere of knowledge, which focuses on the application under development. The user is the most qualified person for this task, because she is the domain expert for the application (i.e., she can determine what the relative contributions are of the `Password`, `Retinal Scan`, and `Fingerprint` choices versus the `Bodyguard`, `Disguise`, and `High Fence` choices). The user knows best how these choices are employed in the application and therefore is in the best position to determine the relative contributions of `Respect Privacy [A]` and `Respect Privacy [B]` towards the application-specific `Respect Privacy [C]`.

## 4.3 Normalization of Goal Satisfaction

As motivated in the previous section, relative contribution values must be used in reusable goal models, if real-life measurements are not available. Relative contribution values lead to relative satisfaction values, which need to be normalized for composability and reusability reasons as well as to avoid making a statement about satisfaction.

Without normalization, goal model elements with relative satisfaction values cannot be composed together (i.e., one would compare apples and oranges). Normalization ensures that

it is theoretically possible for each goal to reach the minimum and maximum values of the defined range for satisfactions (i.e.,  $[0, 100]$ ). For this purpose, it is necessary to determine an element's actual minimum and maximum satisfaction values, to be normalized to 0 and 100, respectively.

For example in Figure 4.1b, the designer of **Application C** reuses **Reusable Artifact A** and **Reusable Artifact B**. Without normalization, satisfaction values for the two top-level goals of these two reusable artifacts (i.e., **Respect Privacy [A]** and **Respect Privacy [B]**) would be similar (i.e., 1.8 and 1.6, respectively). This would make it difficult for the designer of **Application C** to understand and compare the extent to which the two top-level goals of these two reusable artifacts are satisfied and whether it is possible to increase it. When goal satisfactions are normalized (i.e., **Respect Privacy [A]** to 70 and **Respect Privacy [B]** to 19), it becomes possible for the designer of **Application C** to compare the satisfactions for the two reused goals. Furthermore, the application designer can now see that for **Reusable Artifact B** there are much better possible internal solutions for privacy that can be used in the application as 19 corresponds to a relatively low value in the  $[0, 100]$  range.

#### 4.3.1 Forward Propagation with Normalized Satisfactions

The existing GRL forward propagation mechanism needs to be adapted to handle relative contribution values.

As can be seen from the example in Figure 4.1b, the actual number for a relative contribution does not matter, but rather the ratio of the relative contributions of a pair of children. Regardless of whether the relative contributions are stated as  $2 / 1 / -1$  or  $4 / 2 / -2$ , the resulting satisfaction value of the parent goal should be the same, because the ratio of the relative contribution values has not changed (see **Reusable Artifact A**). The existing

GRL evaluation mechanism, however, calculates the resultant satisfaction of **Respect Privacy [A]** as 1.8  $([(2 * 80) + (1 * 20) + (-1 * 0)]/100)$  for the first set of relative contributions and 3.6  $([(4 * 80) + (2 * 20) + (-2 * 0)]/100)$  for the second.

Clearly, the satisfaction value needs to be normalized, as indicated by the larger values next to the Privacy goals in Figure 4.1b (i.e., 70, 19, and 34), before allowing the satisfaction value to be propagated further up.

### 4.3.2 Scale Factor and Offset Values for Normalization

The normalized satisfaction value is calculated by determining the potential maximal and minimal contribution of all children (3 and  $-1$  for **Respect Privacy [A]** and 13 and  $-1$  for **Respect Privacy [B]**), and then mapping this range to the goal model range of  $[0, 100]$ . This normalization step also enables reusable goal models to be combined into goal model hierarchies as illustrated in Figure 4.1b with the **Respect Privacy [C]** goal in **Application C**.

If there are no constraints among children of a parent goal model element, the minimal and maximal satisfaction value for a parent goal  $g$  is easily calculated. Let  $C_i$  stand for the relative contribution of child  $i$  towards the parent goal  $g$ , and  $S_i$  stand for the satisfaction value of child  $i$ . In that case, the actual maximal and minimal satisfaction values for parent goal  $g$  are:

$$S_{max_g} = \sum_{C_i > 0} \frac{C_i * S_i}{100} \quad (4.1)$$

$$S_{min_g} = \sum_{C_i < 0} \frac{C_i * S_i}{100} \quad (4.2)$$

The maximum/minimum is 0 if there are no  $C_i$  that are greater/less than 0, because such

a maximum/minimum of 0 is always given with all  $S_i = 0$ .

**Example 1.** For the goal **Respect Privacy** [B] in Figure 4.1b, the actual maximal satisfaction value ( $S_{max} = 13$ ) is reached when all children that contribute positively, i.e., **Disguise** ( $C_{Disguise} = 6$ ) and **High Fence** ( $C_{HighFence} = 7$ ) are selected, i.e., their satisfaction values ( $S_{Disguise}, S_{HighFence}$ ) are 100. For the same goal, the minimum ( $S_{min}$ ) is  $-1$ , which occurs when all children that contribute negatively are selected (only **Bodyguard** ( $C_{Bodyguard} = -1$ ,  $S_{Bodyguard} = 100$ ) in this case).

The mapping (e.g.,  $-1$  is mapped to 0 and 13 is mapped to 100 for **Respect Privacy** [B] in Figure 4.1b) is done using a *scaling factor* ( $SF$ ) and an *offset* ( $OF$ ) that are calculated as follows:

$$SF_g = \frac{100}{S_{max_g} - S_{min_g}} \quad (4.3)$$

$$OF_g = 0 - S_{min_g} \quad (4.4)$$

Where  $S_{max_g}$  and  $S_{min_g}$  are the maximum and minimum possible values that the goal  $g$  can achieve. If these two values for a goal are the same, it means that its children has no effect on the parent's satisfaction evaluation. In that case,  $SF_g = 1$  and  $OF_g = 100 - S_{min_g}$ .

Using the calculated  $SF_g$  and  $OF_g$  values, the resultant satisfaction value of goal  $g$  ( $SAT_g$ ) can be calculated with the following formula where  $SAT_{graw}$  is the satisfaction value of the goal before normalization:

$$SAT_g = (SAT_{graw} + OF_g) * SF_g \quad (4.5)$$

**Example 2.** For the goal **Respect Privacy** [B] in Figure 4.1b, the scaling factor is calculated as  $100/(13 - (-1)) = 7.143$  and the offset is  $0 - (-1) = 1$ . For the given GRL strat-



egy, the raw satisfaction value of **Respect Privacy** [B] is first evaluated as 1.6 ( $[(-1 * 30) + (6 * 20) + (7 * 10)] / 100$ ) by the GRL evaluation algorithm, and then it is scaled to 19 ( $((1.6 + 1) * 7.143 = 18.57$  rounded up).

Calculating these scaling factor and offset values for each goal model element and using them to normalize the satisfaction values before propagating them further ensures that it is possible for every goal model element to reach the minimum and maximum possible satisfaction values with relative contribution values.

## 4.4 Summary

This chapter investigates which kind of values may be used in reusable goal models that are evaluated with a propagation-based algorithm. We argue that real-life measurements should be used, and if not available, quantitative values with relative contributions must be used instead.

A reusable goal model with relative contribution values must guarantee to the user of the reusable goal model that the possible maximal/minimal satisfaction values of all goal model elements are the same, i.e., they must be normalized. In this chapter, an algorithm is presented that performs such normalization to the  $[0, 100]$  range using two values, scale factor and offset that are determined by the minimum and maximum possible satisfaction values, assuming that no external constraints exist among the goal model elements. Impacts of having such external constraints and their repercussions on the evaluation of reusable goal models are presented in the following chapter.

# Chapter 5

## Reusable Goal Models with Constraints

In this chapter, constraints imposed by feature models and workflow models and their repercussions on the goal model evaluation are investigated.

The chapter is structured as follows: Section 5.1 motivates the need for handling constraints for the accurate analysis of a goal model. Section 5.2 discusses the impact of internal constraints (i.e., constraints that can be represented in the goal model alone) on goal model evaluation. Section 5.3 introduces feature models as complementary models to goal models and how the constraints among features in the feature model may influence the evaluation of a goal model. Section 5.4, then, describes how causal relationships between goal model elements can be expressed in a workflow model and presents our novel approach to incorporate external feature model and workflow model constraints into normalization with scale factor and offset values and overall goal model evaluation. Dealing with these external constraints when evaluating reusable goal models, our approach fully satisfies the third requirement (R.3) for goal model reuse stated in Section 1.2. A discussion on the feasibility of the novel approach dealing with constraints through a proof-of-concept implementation and performance evaluations is provided in Section 5.5, followed by a summary of the chapter in Section 5.6.

## 5.1 Constraints and Goal Model Analysis

When investigating goal models in the context of reuse, another important factor to take into consideration is that goal models are rarely used in isolation. Goal models are often combined with models of other notations that impose other constraints on the elements of the goal model.

Although goal modeling provides means for describing some constraints such as XOR decompositions, separate, dedicated models are also used to express more sophisticated constraints. Such constraints may include the causal relationships of tasks that are introduced with workflow models or limitations on the possible configurations of tasks (i.e., which tasks can be selected together) that are modeled with a feature model.

Ignoring this additional information during goal model analysis may result in an incorrect evaluation. For instance, two tasks that cannot be selected together according to the feature model, should not be evaluated together in the goal model. However, dealing with all of the aforementioned information that is modeled either internally or in an external model may result in increasing the difficulty to analyze and understand the goal model.

A significant impact these constraints may have on reusable goal model analysis is on the normalization and evaluation of goal models. As motivated in Chapter 4, relative contribution values must be used in reusable goal models, and to specify them correctly the user of the reusable goal model relies on the fact that the possible maximal/minimal satisfaction values of goal model elements are all the same. To ensure that this is the case, the satisfaction value of each goal model element needs to be normalized so that each goal model element can be reused by the user.

Normalization in this context means that the possible maximal/minimal satisfaction value for each goal model element is always 100/0. For this purpose, it is necessary to determine the actual minimal and maximal satisfaction values that can be achieved by a

goal model element. However, having additional constraints imposed on some of the goal model elements may change the actual minimal and maximal satisfactions they can achieve. As a result of this, normalization methods described in the previous chapter may require certain adjustments to handle different types of constraints.

In the remainder of this chapter, the suggested solutions of this research for dealing with additional constraints that can be modeled in the goal model, in a feature model, or in a workflow model during the analysis and evaluation of a reusable goal model as well as their feasibility is discussed.

## 5.2 Internal Constraints

While different modeling formalisms are often used in collaboration with goal models to represent additional information, there are still internal constraints (e.g., XOR relationships among model elements) that can be modeled with goal models, and the calculation needs to take them into account.

If there is an XOR constraint among all children (i.e., exactly one child's satisfaction value must be greater than 0 and all other ones 0), then the maximum for the parent is the maximal incoming relative contribution (i.e., 4 for **Parent A** in Figure 5.1) and the minimum is the minimal incoming relative contribution (i.e., -3 for **Parent A** in Figure 5.1).

Constraints may occur across any branches in the goal model (e.g., between the left and right branch of **Parent C** in Figure 5.1), because **Parent A** may require **Solution 1** (one of its children) to be 100 while **Parent B** may require the same **Solution 1** to be 0. Hence, it may be the case that a constraint does not allow both **Parent A** and **Parent B** to be maximized at the same time. As a result, when determining the maximum satisfaction value of **Parent C** it may be necessary to use the next lower maximal satisfaction value for **Parent A** or **Parent B**, until a combination is found that does not violate a constraint. In

other words, some of the satisfaction values of children in the maximum calculation may be strictly less than 100. The same reasoning applies to the minimum calculation, where some of the satisfaction values of children could be strictly greater than 0.

### 5.2.1 The Lazy, Recursive Normalization Algorithm

This section describes the novel algorithm that calculates the maximum (and by analogy also the minimum) and takes constraints into account.

In the worst case,  $2^n$  combinations have to be examined to find the maximal satisfaction value without a violated constraint, where  $n$  stands for the number of nodes with user-defined satisfaction values (i.e., typically the leaf nodes in the reusable goal model).

The proposed, novel evaluation algorithm avoids combinatorial explosion with a recursive, top-down approach that uses a lazy calculation of the next highest sum. This reduces the calculation time significantly, but the worst case still requires  $2^n$  combinations to be examined.

**Example 3.** An example calculation for the maximal satisfaction value of **Parent C** is shown in Figure 5.1, assuming a virtual list with all possible satisfaction values sorted from highest to lowest for each child of **Parent C** (i.e., **Parent A**, **Solution 8**, and **Parent B**; see left side of Figure 5.1). This virtual list does not exist a priori. Instead, each entry in the virtual list is determined on demand. An entry in the virtual list identifies the selection of **Parent C**'s grandchildren and the corresponding normalized satisfaction value for **Parent C**'s child in the range of  $[0, 100]$ . Each entry does not violate any constraints. In our example, there is only one constraint shown at the top right of Figure 5.1 that indicates that **Solution 3** and **Solution 8** cannot be used together.

The algorithm gets the next highest combinations in Step I, then calculates the potential maximal satisfaction value for each next highest combination (Step II), and checks if any

## 5.2. Internal Constraints

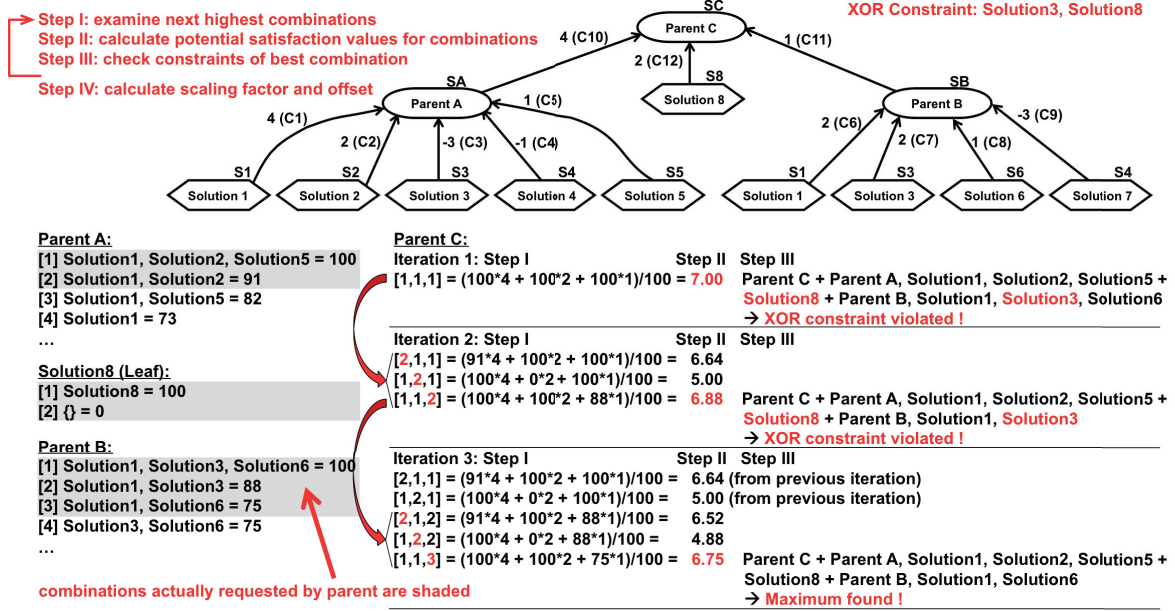


Figure 5.1: Example execution of the lazy, recursive normalization algorithm for maximum [28]

constraints are violated by the best combination (Step III). If a constraint is violated, the algorithm goes back to Step I. If no constraint is violated, the algorithm has found the maximal satisfaction value for the current parent.

The algorithm proceeds by getting the minimal satisfaction value using an analogous approach except that the list of all possible satisfaction values is sorted from lowest to highest instead of from highest to lowest. Once both the maximum and the minimum are known, the algorithm calculates the scaling factor and the offset for the parent in Step IV.

**Example 4.** As illustrated in Figure 5.1 for Parent C, the combination  $[1,1,1]$  is examined in Step I of the first iteration (i.e., this indicates the combination with the highest satisfaction value of each of Parent C's children (Parent A, Solution 8, and Parent B, respectively), because each child contributes positively to Parent C). The combination results in the potential maximum of 7 (Step II of first iteration), but because of the XOR constraint between

**Solution 3** and **Solution 8** this result is not valid (Step III of first iteration).

Therefore, the next best combinations are examined in Step I of the second iteration. These combinations are determined by going to the next best possible satisfaction value for each child in the invalid combination. Because each child's list is sorted from highest to lowest, the next best combinations can only be those where the index of each child is advanced individually by one. Since  $[1,1,1]$  is invalid according to the first iteration, the combinations  $[2,1,1]$ ,  $[1,2,1]$ , and  $[1,1,2]$  must be examined next. This results in the potential maximums of 6.64, 5.00, and 6.88, respectively, i.e., the third combination  $[1,1,2]$  is the next best option (Step II of second iteration). However, this combination still includes **Solution 3** and **Solution 8** and hence is again invalid (Step III of second iteration).

In Step I of the third iteration, the next best combinations are again examined (i.e., since  $[1,1,2]$  is invalid according to the second iteration,  $[2,1,2]$ ,  $[1,2,2]$ , and  $[1,1,3]$  are the next combinations added to the already examined combinations). This results in the new potential maximums of 6.52, 4.88, and 6.75, respectively. Considering the remaining combinations from the second iteration (i.e.,  $[2,1,1]$  with 6.64 and  $[1,2,1]$  with 5.00), the combination  $[1,1,3]$  with 6.75 is the next best option (Step II of third iteration). This time, the combination is valid as no constraint is violated (Step III of third iteration). Therefore, the maximal satisfaction value for **Parent C** is 6.75. Once the minimal satisfaction value has also been found, the scaling factor and the offset can be calculated (Step IV).

At the end of the example above, only the maximal and minimal satisfaction values of **Parent C** have been determined and there is no need to calculate anything else. However, if the goal model were bigger and the parent of **Parent C** requested the next highest (or lowest) satisfaction value, then the algorithm would continue with further iterations of Steps I–III to determine the next highest/lowest satisfaction value. Therefore, the list of all possible satisfaction values as illustrated in Figure 5.1 does not exist for each child of **Parent C**

right from the start, but instead only the needed combinations are calculated recursively on demand (see shaded combinations in Figure 5.1).

In summary, the normalization of the satisfaction values is a lazy algorithm that starts at each root node of the goal model and recursively determines the maximal/minimal satisfaction values for each node in the reusable goal model. The recursion stops at the leaf nodes of the goal model, because the maximal and minimal satisfaction value of 100 and 0, respectively, are known for leaf nodes (e.g., see **Solution 8** in Figure 5.1).

The contributions to **Parent C** in the example shown in Figure 5.1 are only positive. If there were a negative contribution, then the algorithm would ask the corresponding child for its minimal value instead of its maximal value when calculating the maximal value for the parent (and vice versa for the minimal value of the parent).

An overview of the lazy, recursive algorithm used in scale factor and offset calculation is given in Algorithm 5.1-5.2.

## 5.3 Feature Model Constraints

As mentioned earlier, while some constraints among tasks can be modeled directly with goal models (e.g., an XOR constraint), more sophisticated constraints cannot be modeled with goal models alone and have to be specified in dedicated models that are used in conjunction with goal models.

Since dedicated models have to be used anyway, the proposed approach advocates that goal models only focus on the impact of candidate solutions on high-level goals and that all constraints among those candidates are to be modeled separately.

The first class of such constraints are configuration constraints that are specified with a feature model [224]. A feature model, in the context of reuse, captures the variations by identifying variable and common *features* (visualized as tasks  $\Diamond$  to highlight that goal



**Algorithm 5.1** Lazy, Recursive Algorithm for Scale Factor and Offset Calculation

---

```

1  calculateScaleFactorsAndOffsets(GoalModel gm, Constraints c) {
2      Map<Goal, ContributorsToEvaluate> evaluationCounter;
3      Set<Goal> goalsReadyToEvaluate;
4      // Initialization finds the contributor counts for goals, sets the known min/max
5      // values for leaf nodes (i.e., solutions/tasks), finds the leaf goals to be
6      // evaluated based on the evaluationCounter, and adds a goal that does not have
7      // any not-yet-evaluated contributors to the ready list.
8      initialize(gm, evaluationCounter, goalsReadyToEvaluate);
9
10     // After the initialization, a table for a goal model element has the following
11     // structure and is kept sorted by value throughout the execution:
12     // index | value | configuration | childrenIndex
13     // e.g., The table for a leaf node (e.g., Solution 1), for the calculation
14     // of maximum achievable satisfaction looks like the following:
15     // index | value | configuration | childrenIndex
16     // -----
17     // 0 | 100 | Solution 1 | N/A
18     // 1 | 0 | | N/A
19
20     while (goalsReadyToEvaluate.size() > 0) {
21         Goal goal = goalsReadyToEvaluate.get(0);
22         evaluateGoal(goal, c);
23
24         // When a goal is evaluated, counters for the goals it contributes to are
25         // decremented. Ready list is updated accordingly. The last goal that gets
26         // evaluated is the root goal.
27         updateCounterAndReadyList(goal, evaluationCounter, goalsReadyToEvaluate);
28     }
29 }
30
31 evaluateGoal(Goal g, Constraints c) {
32     // First, evaluate for maximum.
33     isConfigurationValid = false;
34     index = 0;
35     while (!isConfigurationValid) {
36         // find a solution set corresponding to the index
37         {solutionSet, goalSatisfaction} = getIndexedMaximumForNode(g, index);
38
39         // check the solution set for constraint violations
40         isConfigurationValid = checkConfigurationValidity(solutionSet, c);
41
42         if (isConfigurationValid) {
43             maximumAchievable = goalSatisfaction;
44         } else {
45             index++;
46         }

```

---

**Algorithm 5.2** Lazy, Recursive Algorithm for Scale Factor and Offset Calculation (cont.d)

---

```

47     }
48     // Similarly, evaluate for minimum and find minimumAchievable.
49     {...}
50     // Now, calculate and store scale factor and offset values.
51     g.setScalingFactor(100 / (maximumAchievable - minimumAchievable));
52     g.setOffset(0 - minimumAchievable);
53 }
54
55 // This method is recursively called for a goal and its children to evaluate
56 // the maximum achievable value corresponding to an index.
57 {Combination, MaxValue} getIndexMaximumForNode(Goal g, int index) {
58     if (indexAlreadyCalculatedForGoal(g, index)) {
59         return storedMaxCombinationForGoalWithIndex(g, index);
60     }
61     // This method finds the last checked index for the goal, generates an array of
62     // children indexes to be checked. e.g., if [2,3] index was checked the last
63     // time for a goal, next indexes would return [3,3] and [2,4].
64     List<int[]> indexesToAskChildren = calculateNextIndexesToCheck(g);
65
66     for (anIndexToAsk : indexesToAskChildren) {
67         if (!indexAlreadyInGoalsTable(g, anIndexToAsk)) {
68             // Ask each contributing child node
69             for (i = 0; i < anIndexToAsk.length; i++) {
70                 cIndex = anIndexToAsk[i]; // child's index
71                 aChild = getChildOfGoal(g, i); // corresponding child
72                 // If a child contributes positively, look for next maximum.
73                 if (contributionOfChildToGoal(aChild, g) >= 0) {
74                     {childSlnSet, value} = getIndexMaximumForNode(aChild, cIndex);
75                 } else { // If a child contributes negatively, look for next minimum.
76                     {childSlnSet, value} = getIndexMinimumForNode(aChild, cIndex);
77                 }
78                 // Combine and sum incoming solution sets and values for the goal.
79                 {combinedSlnSet, combinedVal} = combineSolutions(childSlnSet, value);
80             }
81             // Store the combination in the goal's table
82             insertIndexToGoalsTable(g, anIndexToAsk, combinedSlnSet, combinedVal);
83         }
84     }
85     // Get the next best set from the goal's table with its value and return.
86     return storedMaxCombinationForGoalWithIndex(g, index);
87 }
88
89 // The minimum achievable value calculation is analogous to the maximum.
90 {Combination, MinValue} getIndexMinimumForNode(Goal g, int index)
91 {...}

```

---

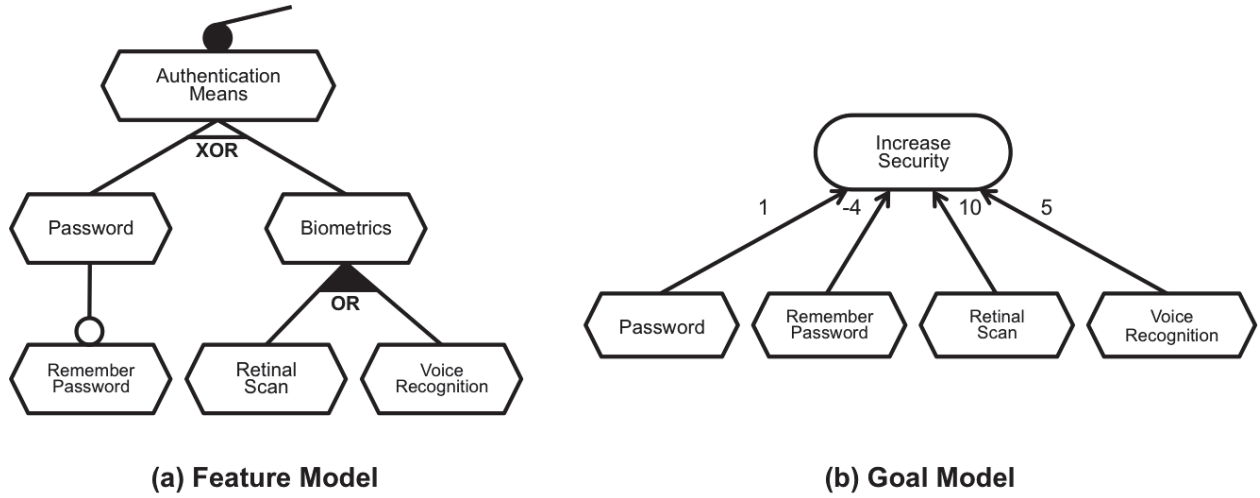


Figure 5.2: Excerpts of feature and goal models of a reusable artifact “Authentication”

model elements are constrained) for the possible reuses of the artifact. *Mandatory* (—●) and *optional* (—○) parent-child relationships, alternative (*XOR*) and *OR* (inclusive) feature groups, as well as integrity constraints (*includes* and *excludes*) are captured by the feature model.

### 5.3.1 Feature Models and Goal Modeling

When the feature model is used in combination with a goal model, some of the features are represented as tasks (candidate solutions) in the goal model (e.g., **Password**, **Remember Password**, **Retinal Scan**, and **Voice Recognition** in Figure 5.2a and b), which imposes the feature modeling constraints on the goal model evaluation as it restricts the possible task/feature selections [225].

Types of restrictions that can be imposed by feature models may vary. A child in a mandatory relationship with its parent must be selected when the parent is selected. A child in an optional relationship with its parent may be selected when the parent is selected. Exactly one child in an XOR group must be selected when the parent is selected, while at

least one child must be selected in an OR (inclusive) group when the parent is selected. Finally, if *feature A* has an includes relationship with *feature B*, then *feature B* must be selected when *feature A* is selected. If *feature A* has an excludes relationship with *feature B*, then *feature B* cannot be selected when *feature A* is selected.

For example, in Figure 5.2, part of the feature model that represents the relationships between the four solutions in the goal model is shown. For the goal **Increase Security**, since **Password** and **Biometrics** are modeled as XOR'ed features in the feature model, the possible maximum would represent the case where only two of the three positively contributing children is selected (i.e., **Retinal Scan** and **Voice Recognition**) as it is not possible to select three of them together (hence leaving **Password** out and yielding 15 as the possible maximum instead of 16).

Consequently, the external constraints introduced by the feature model complicate the evaluation of scaling factor and offset values as a straightforward sum of positive contributions for the maximum and negative contributions for the minimum cannot be used anymore.

Therefore, a novel evaluation mechanism is implemented that takes additional constraints on tasks in the goal model into account to enable reuse of goal models with relative contributions through two lazy, recursive algorithms for feature and goal models as well as a SAT solver [226] for checking the feature model constraints.

### 5.3.2 Changes to the Evaluation Algorithm for Feature Model Constraints

If a feature model is used to define these additional constraints, then Step III in the lazy, recursive algorithm introduced in Section 5.2 involves the evaluation of a feature model to determine whether a constraint is violated.

The check for constraint violations is typically done by converting the feature model and

the selected features into a propositional formula, which is then given to a SAT solver. In this work, the feature modeling tool FAMILIAR [226] is used as the SAT solver and its library is used for the implementation. A straightforward integration of goal and feature models therefore involves simply replacing the constraint check described in Section 5.2 with a check performed by a SAT solver. The constraint check is the only part of the algorithm that needs to be tailored to whatever model is chosen to capture the constraints. The rest of the algorithm, however, stays the same and constitutes an essential part of reusable goal models.

This section describes an improvement to the evaluation algorithm for performance reasons, because calling the SAT solver is an expensive operation.

The SAT solver is best used for cross-tree constraints (i.e., includes/excludes), because many levels of indirections potentially have to be considered, which is difficult to do without converting the feature model into a logic formula. OR and XOR constraints, however, are much easier to handle, because violations can be determined by simply looking at the feature tree, hence reducing the number of evaluations done by the SAT solver. As it turns out, the same lazy, recursive algorithm used for calculating the minimal/maximal satisfaction values in the goal model can also be used to reduce calls to the SAT solver.

When goal and feature models are used in conjunction, the tasks in the goal model are also features in the feature model [225]. In this case, the recursion in the goal model stops at a parent of tasks and hands the tasks over to the recursive algorithm for the feature model to determine their candidate contribution to the maximal/minimal satisfaction of the parent (e.g., in Figure 5.1, the children of **Parent A**, i.e., **Solution 1** to **Solution 5**). Note that mandatory features do not need to be included in the goal model, because their contributions are always the same regardless of the selected features.

Since the example in Figure 5.1 showed the calculation of a maximal satisfaction value, the feature model example in Figure 5.3 shows - for variety - the calculation of a minimal

### 5.3. Feature Model Constraints

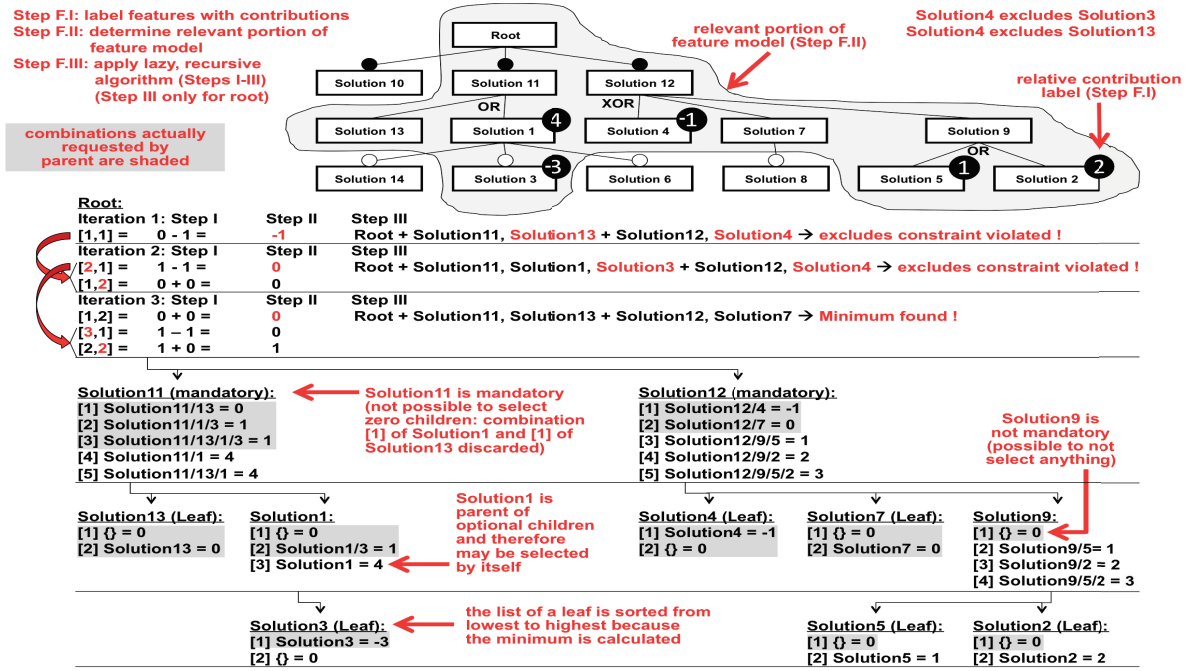


Figure 5.3: Contributions in feature model (Parent A's children) for minimum [28]

contribution of features to their parent.

For each parent, the algorithm starts with labeling the features with their contributions to their goal model parent (Step F.I; e.g., the contributions shown for Parent A in Figure 5.1 are added as labels in Figure 5.3). From then on only a portion of the feature model needs to be considered, i.e., the union of the paths from the labeled features to the root feature including any siblings in OR/XOR groups encountered on a path (Step F.II), because these paths capture all relevant OR/XOR constraints. The siblings are included, because the selection of a sibling influences the minimal/maximal contribution to the parent in the goal model (e.g., Solution 13 with contribution 0 may be selected instead of Solution 1 with contribution 4). The last step (Step F.III) uses the same lazy, recursive approach as for reusable goal models (i.e., Steps I to III).

Steps I to III are exactly the same, except that the calculation of the potential satis-

faction value is now a simplified calculation of the potential contribution value. Because a feature is either selected or not (i.e., satisfaction value of a feature is either 100 or 0, respectively [225]), the potential satisfaction value (i.e., sum of  $C_i * S_i$  divided by 100, as explained in Equation 4.1) reduces itself to the sum of relative contributions of the children.

Even though the labels appear at different levels in the feature model, they can still be added up directly, because they all represent contributions at the same level in the goal model (e.g., option [2] of **Solution 1** adds up 4 and -3 of **Solution 1** and **Solution 3**, respectively).

The reduction in calls to the SAT solver is achieved by considering only the possible feature combinations as restricted by the feature model constructs and hence performing the constraint calculation (Step III) only for the root. Considered constructs are OR/XOR groups with or without a mandatory parent and parents of optional children. E.g., an XOR group (**Solution 12**) only considers each child, but not any combination of its children, and for an OR/XOR group with a mandatory parent (**Solution 11**), 0 children cannot be selected.

Note that the SAT solver is only needed by the designer of the reusable artifact to determine the offsets and scaling factors when the feature and goal models are created. The offsets and scaling factors are then stored with the reusable artifact. Therefore, the user of the reusable artifact does not need the SAT solver, because the user only needs the offsets and scaling factors to evaluate the goal model based on feature selections.

## 5.4 Workflow Model Constraints

The second class of external constraints is described by causal relationships in workflow models. Some of these constraints may also be modeled in goal modeling notations, but a dedicated workflow notation is again needed to express more complex workflows.

The key concepts of a workflow notation that fall under the context of this research are the start and end of a workflow, actions (which here correspond to features/tasks, i.e., candidate solutions), alternative and concurrent behavior, and loops. The Use Case Map (UCM) notation [7], which is also part of the User Requirements Notation (URN), is used in this work to model workflows. A start node is modeled as a filled circle ( $\bullet$ ) whereas a thick bar ( $\blacksquare$ ) is used for the end of a workflow. Actions are shown with a diamond called stub ( $\diamond$ ). Alternatives are represented with a fork ( $\negthinspace\lrcorner$ ) followed by a merge ( $\negthinspace\lrcorner$ ) whereas concurrency is represented with a bar with multiple outgoing branches ( $\negthinspace\vdash$ ) followed by a bar with multiple incoming branches ( $\negthinspace\lrcorner$ ). Loops are also modeled with a fork and a merge.

Even though the UCM notation is used in this work, any other workflow or scenario notation may be used as long as it covers the aforementioned workflow concepts.

The proposed approach advocates the use of feature models, workflow models, and goal models to describe a reusable artifact as suggested by Concern-Oriented Reuse (CORE) [20] [28], and assumes that the workflow notation is well-nested, which is the case for many workflow notations (e.g., Unified Modeling Language (UML) activity diagram).

### 5.4.1 Workflow Models and Goal Modeling

Workflow models are connected to goal models because they may express the run-time behavior of the tasks/features of a reusable artifact in the workflow and the causal relationships among them. The causality may address relationships such as which feature is executed before, after, as an alternative to, or in parallel to another feature.

In this context, goal models are graphs that are free of circular dependencies. To motivate why constraints on tasks imposed by feature models and workflow models have to be taken into account when evaluating goal models with relative contribution values in a reuse hierarchy, consider the following illustrative example in Figure 5.4 based on a reusable



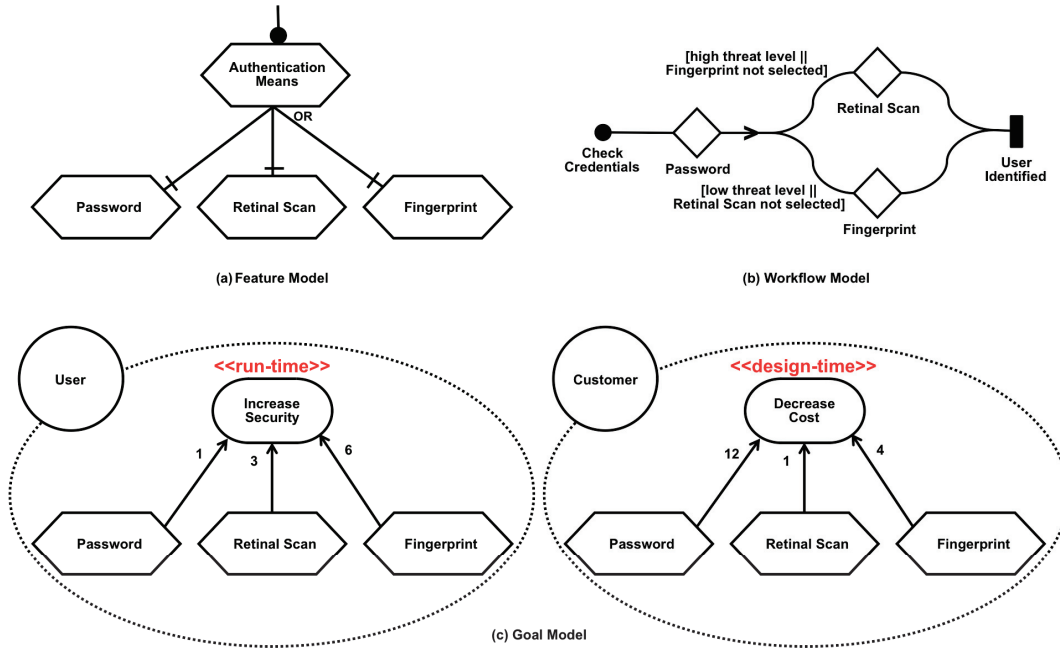


Figure 5.4: Excerpts of feature, goal, and workflow models of “Authentication” [24]

artifact for Authentication. Figure 5.4 demonstrates how the strengths of three modeling notations can complement each other to model different aspects of a reusable artifact.

The feature model in Figure 5.4a expresses that an application developer, while reusing this Authentication artifact, may choose to integrate the functionality of all three authentication means in the application under development. The OR feature group of the parent feature **Authentication Means** with the three children **Password**, **Retinal Scan**, and **Fingerprint** allows for that and also dictates that at least one of the children must be selected, i.e., integrated into the application under development.

The workflow model in Figure 5.4b shows the causal relationships of the features in the feature model, i.e., the sequence in which the tasks/features **Password**, **Retinal Scan**, and **Fingerprint** occur at run-time as designed by the creators of the Authentication artifact. If all three authentication means are selected in the feature configuration, the checking

of a user's credentials is handled as follows. First, the user is asked for the password. Upon successful login with the password, the system presents one of two options to the user as a second layer of security based on the current threat level, i.e., either retinal scan when the threat level is high or otherwise fingerprint authentication. Consequently, two of the authentication means, **Retinal Scan** and **Fingerprint**, cannot occur within the same operational scenario at run-time, i.e., these two features have a *run-time conflict*. The feature model does not capture this constraint, because the feature model only indicates that a selected feature exists in the application under development. The run-time conflict cannot be expressed in the feature model (e.g., with a cross-tree constraint), as doing so would prevent the features from being selected together. However, both features must be selected for the example with the run-time assessment of the threat level. Note that if an authentication method is not selected in the feature configuration, then the diamond-shaped stub representing the feature in the workflow model is simply bypassed (e.g., if the user does not select **Password** and **Retinal Scan**, only **Fingerprint** is used to identify the user regardless of the threat level).

The goal model in Figure 5.4c represents the impacts of selecting one of the authentication means over another on the cost and security of a system. Since not all system goals care about the run-time conflicts, a classification of goals as run-time goals (labeled with  $\langle\langle run-time \rangle\rangle$ ) and design-time goals (labeled with  $\langle\langle design-time \rangle\rangle$ ) is introduced. For design-time goals (e.g., **Decrease Cost** in Figure 5.4c), the satisfaction level is independent of run-time behavior whereas for run-time goals (e.g., **Increase Security**), the satisfaction level can be observed at run-time.

Relative contribution values are assigned by the creators of the reusable Authentication artifact, who are the domain experts, to allow the users of this reusable artifact to perform trade-off analyses among different candidate solutions. As explained in Chapter 4, use of relative contribution values requires scale factors and offsets to be determined to make sure

that the maximum and minimum achievable satisfaction values for the goal are mapped to 100 and 0. This in turn requires the maximum and minimum non-normalized satisfaction values of a goal to be determined.

Without taking the run-time conflicts into account, it is possible to select all three authentication means given the feature model. Therefore, the best configuration for maximal security is **Password**, **Fingerprint**, and **Retinal Scan** ( $1 + 3 + 6 = 10$ ) and the worst configuration for minimal security is **Password** (1). However, considering the workflow model of the Authentication artifact, because of the conflict between **Fingerprint** and **Retinal Scan**, this best-case configuration can never occur at run-time. Taking the run-time conflict into account adjusts the best configuration to **Password** and **Retinal Scan** ( $1 + 6 = 7$ ), since the contribution of **Retinal Scan** to **Increase Security** of the **User** actor is higher than that of **Fingerprint**.

The adjusted maximum value reflects the characteristics of the Authentication artifact in a more precise manner while showing the necessity of taking both feature model and workflow model constraints into account for a more accurate assessment for reuse. Although it is possible to select both **Fingerprint** and **Retinal Scan** in the same feature configuration and use them together in the application, their contributions to the same goal must not be evaluated additively because they cannot occur together at run-time.

The evaluation algorithm described in the previous section takes into account only the constraints imposed on the goal model by the feature model and therefore needs to be improved to deal with workflow model constraints.

In summary, an accurate evaluation of the goal model of a reusable artifact essentially has to take into account that the following cannot contribute together to the satisfaction value of a goal:

- (i) two features that are not selectable together in a valid feature configuration of the feature model and

(ii) two features that do not occur together at run-time in all operational scenarios defined by the workflow model.

### 5.4.2 The Improved Evaluation Algorithm for Workflow Model Constraints

The proposed approach takes additional run-time constraints into account only for features in an OR feature group. The reasoning behind this is that an OR feature group typically captures alternatives that can be used interchangeably. For instance, different authentication means that identify a user or different payment methods that allow something to be bought can be used in the same operational scenario for the same purpose as alternatives to each other. However, the feature model does not express whether these features can be used together at run-time or not (e.g., authentication means or payment methods may be stacked sequentially or may be pure alternatives at run-time). Consequently, workflow models need to be considered.

Run-time constraints do not apply to features in an XOR feature group, because only one feature may be selected at any time. For example, if the OR feature group in Figure 5.4a were an XOR feature group, the maximal security would be Retinal Scan (6) and no run-time conflict could change this. On the other hand, optional features often do not describe similar features. Instead, they describe vastly different features that have no run-time relationships. As the number of constraints becomes larger, the evaluation of the goal model becomes more complex and time-consuming. Consequently, the proposed approach restricts run-time constraints to OR feature groups.

As mentioned earlier, an important aspect to consider for run-time conflicts in workflow models is that not all goals care about run-time conflicts. Therefore, the creators of the reusable artifact have to identify the type of a goal in the goal model. A goal is either

classified as a run-time goal or a design-time goal:

- a) **Run-time goal:** a goal for which the satisfaction level can be observed at run-time. For example, the previously discussed security goal (i.e., **Increase Security**) is tagged as `<<run-time>>` in Figure 5.4c.
- b) **Design-time goal:** a goal for which the satisfaction level is independent on run-time behavior, but can be observed at design-time. For example, the cost goal (i.e., **Decrease Cost**) is tagged as `<<design-time>>` in Figure 5.4c. In this case, the goal model indicates that the cost goal depends on whether the feature has been included in the system at design-time, but not on the operation of the system.

Satisfaction values for design-time goals are to be evaluated with the evaluation algorithm described in Section 5.3, as run-time information does not play a role in the calculation of the satisfaction values. However, for the evaluation of run-time goals, it becomes essential to check whether two features that can be selected in the same feature configuration are modeled as run-time alternatives in the operational scenarios of the system.

The improved algorithm introduced for the purpose of dealing with workflow model constraints consists of three major parts. These three parts are briefly described below. In the remainder of this section, we focus on giving a detailed description of how the three parts of this algorithm work and how the prior evaluation mechanism is altered to achieve a more accurate representation of the satisfaction values of the goals of a reusable artifact.

- Part 1: To better understand the run-time interactions among features in the workflow model, all possible alternative paths that could be taken during the operation of the application are extracted. This allows determining whether some features never appear together in the same operational scenario. If this is the case, then those features are said to be conflicting at run-time and consequently shall not contribute to the same goals in a cumulative manner, thus allowing for more accurate trade-off analyses.

- Part 2: The second part of the algorithm concerns the adjusted calculation of scale factors and offsets with the help of the best/worst possible non-normalized satisfaction value that can be achieved for a goal. The existing evaluation mechanism processes possible feature configurations in a lazy recursive fashion until a feature configuration is found that yields the best/worst satisfaction value without violating feature model constraints. This mechanism needs to be adapted to also check for run-time conflicts before checking for the violation of feature model constraints. This order of checks is chosen because the feature model check requires the use of a SAT solver, which is a rather expensive operation. The check for run-time conflicts, on the other hand, is comparatively fast given the results of the first part (i.e., whether a feature appears together with another feature in the same operational scenario).
- Part 3: The last part of the algorithm takes place during the propagation phase of the goal model evaluation mechanism, i.e., the propagation of the initial satisfaction values of tasks to the higher-level stakeholder/system goals according to the contribution links. The chosen feature configuration determines the initial satisfaction values and consequently drives the goal model evaluation and trade-off analysis. However, a feature configuration may now contain features that conflict with each other at run-time and therefore should not be selected at the same time for the goal model evaluation. If this is the case, the new evaluation mechanism determines – starting from the chosen feature configuration – a more accurate feature configuration solely for the goal model evaluation that takes the run-time conflicts into account.

### **Part 1: Extraction of Paths from Workflow Model**

Extracting the feature interaction information from the workflow model is the first part of an improved goal model evaluation mechanism.

The set of operational paths specified by the workflow model helps reveal those features that cannot appear together in any scenario of the workflow. The proposed algorithm traverses the paths in the workflow model and determines the operational paths without user intervention and regardless of complexity of the workflow.

For this path traversal algorithm to work as desired, the workflow model is assumed to be well-nested, which is already the case for many workflow notations and can be enforced even for those workflow notations that do not require it. This restriction keeps the complexity of the path extraction algorithm at a manageable level.

A prerequisite for the determination of interactions among features in workflow models is that the actions in the workflow model are linked to their corresponding features. If a feature is selected, then the linked actions are executed at run-time. If a feature is not selected, then the linked actions are simply ignored in the workflow.

The extraction of paths from the workflow model begins with the removal of those actions in the workflow model that are not linked to a feature in an OR feature group.

In the example path extraction illustrated in Figure 5.5, actions denoted with letters from A to H are actions that are linked to features in an OR feature group. Actions denoted with `action1-6` in the original workflow model in Figure 5.5a are not linked to a feature in an OR feature group and are hence removed during the filtering step (b). During the filtering step, whole branches may be removed if no action remains on the branch (e.g., see the alternative branch with `action3` in Figure 5.5a).

Keeping in mind that the workflow model is well-nested, having a loop within the workflow model means that all of the actions that are placed between the beginning and end points of the loop may be executed together in one operational scenario at run-time. Even if the actions are placed on alternative branches inside the loop in the workflow model, different alternative branches may be visited during each loop iteration, so that in practice those alternative paths are executed sequentially in the same operational scenario. As a result, all

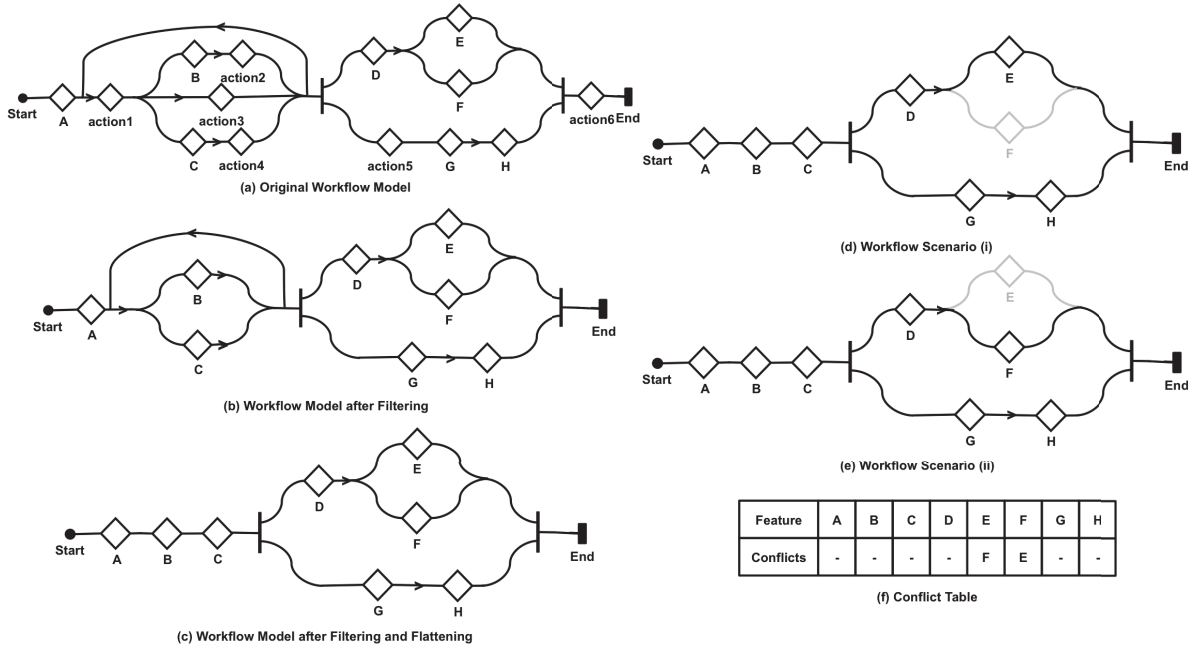


Figure 5.5: Steps of path extraction [24]

of the actions inside a loop are treated as being in the same operational scenario for the sake of identifying run-time conflicts among features.

The treatment of loops follows a conservative approach, because it is based on a static analysis of the workflow model (even though the conflicts are called run-time conflicts). A static analysis does not allow the exact number of loop iterations to be determined, and hence all path elements inside the loop are deemed to be in the same operational scenario, leading to a flattened workflow model. The flattening step simplifies both the workflow model and the path extraction procedure. Figure 5.5c illustrates how the flattening is done for the simplified workflow model in Figure 5.5b. The loop around actions B and C is removed as both actions may appear in the same operational scenario, hence not creating a run-time conflict.

Once the filtering and flattening steps are completed, the actions (i.e., features) that



cannot appear in the same operational scenario (e.g., E and F in Figure 5.5d and Figure 5.5e) are determined with the core path extraction algorithm described in the next paragraph and in Algorithm 5.3. Based on the extracted paths, it is rather trivial to build a conflict table that contains the run-time conflict information among features (see Figure 5.5f). This table is then used in parts 2 and 3 by the evaluation mechanism to determine whether a given feature configuration violates any run-time conflicts.

The core path extraction described in Algorithm 5.3 is called with the start node of the workflow model and returns as output the set of paths (`WorkflowPathsSet`) that may be taken at run-time (more precisely, it returns the group of features for each distinct run-time path).

`PathExtraction` is a recursive function that calls three methods of the `WorkflowPathsSet` class. `addNodeToAllPaths` takes a workflow node as input and appends the given node to each of the paths of the object. `addPaths` is a set union operation that takes a set of paths as input and adds it to the path set of the object.

**Example 5.** Calling the method `addPaths` on the path set  $\{a-b-c, d-e-f\}$  with the input set  $\{x-y-z, q-r-s\}$  results in the following set of paths  $\{a-b-c, d-e-f, x-y-z, q-r-s\}$ .

The method `combinePaths` is a cross product operation that takes a set of paths as input and creates new path combinations for each possible tuples by appending each path in the input set to each path of the object.

**Example 6.** Calling the method `combinePaths` on the path set  $\{a-b-c, d-e-f\}$  with the input set  $\{x-y-z, q-r-s\}$  results in the following set of paths  $\{a-b-c-x-y-z, a-b-c-q-r-s, d-e-f-x-y-z, d-e-f-q-r-s\}$ .

Concurrent branches are executed at the same time and, as a result, the paths extracted from those branches are combined with `combinePaths` (i.e., the concurrent path elements

---

**Algorithm 5.3** Core Path Extraction Algorithm [24]

---

```
1  Algorithm PathExtraction
2  Input startNode:WorkflowNode
3  Outputs executablePaths:WorkflowPathsSet, nextNode:WorkflowNode
4
5  currentNode:WorkflowNode = startNode
6  nextNode:WorkflowNode =  $\emptyset$ 
7  executablePaths:WorkflowPathsSet =  $\emptyset$ 
8
9  while ((currentNode.getType() != ALTERNATIVE_END)
10     && (currentNode.getType() != CONCURRENCY_END))
11  {
12     if (currentNode.getType() == ALTERNATIVE_START)
13     {
14         innerPaths:WorkflowPathsSet =  $\emptyset$ 
15         for each pathStart:WorkflowNode in currentNode.getNextBranches()
16         {
17             branchPaths, currentNode = PathExtraction(pathStart)
18             innerPaths.addPaths(branchPaths)
19         }
20         executablePaths.combinePaths(innerPaths)
21     }
22     else if (currentNode.getType() == CONCURRENCY_START)
23     {
24         innerPaths:WorkflowPathsSet =  $\emptyset$ 
25         for each pathStart:WorkflowNode in currentNode.getNextBranches()
26         {
27             branchPaths, currentNode = PathExtraction(pathStart)
28             innerPaths.combinePaths(branchPaths)
29         }
30         executablePaths.combinePaths(innerPaths)
31     }
32     else
33     {
34         executablePaths.addNodeToAllPaths(currentNode)
35         if (currentNode.getType() == END_NODE)
36             break
37         currentNode = currentNode.getNextNode()
38     }
39 }
40 nextNode = currentNode.getNextNode()
41 return executablePaths, nextNode
```

---

appear in the same operational scenario and are therefore not in conflict). On the other hand, alternative branches do not occur at the same time and therefore the paths extracted from those branches have to remain separate (i.e., they are not combined with each other but only added to the same list with `addPaths`). In both cases, the resulting path set for the concurrent or alternative branches (i.e., `innerPaths`) is combined with the rest of the workflow (see lines 20 and 30 in Algorithm 5.3).

The presented algorithm visits each node in the workflow model exactly once. However, many permutations of path segments may still have to be considered, depending on the structure of the workflow model. Therefore, path extraction and conflict identification is done only when the design of the workflow and feature models is completed (i.e., they are saved), hence separating this potentially time consuming operation from the evaluation of the goal model, which is to be done on-the-fly.

### Part 2: Revision of scale factor and offset calculation

As motivated in Chapter 4, relative contributions must be used for reusable goal models.

For a reusable goal model to be used in reuse hierarchies, it is essential to have the same possible maximal and minimal satisfaction values for all goal model elements. Therefore, the satisfaction value of each goal model element is normalized with the help of a scale factor and an offset to ensure that the normalized maximal/minimal satisfaction values are always 100/0.

The normalization step requires the non-normalized minimal and maximal satisfaction values for goal model elements to be calculated first. The normalization step must be performed whenever the feature, workflow, or goal model changes.

The lazy recursive algorithm described in Section 5.3 takes the constraints imposed by the feature model on tasks in the goal model into account when calculating the scale factors

and offsets. For the purpose of handling the additional constraints introduced by workflow models, an additional check is introduced before the verification of a candidate minimal/maximal feature configuration against feature constraints. This additional constraint check takes the suggested feature configuration as input and compares it with the conflict table obtained in the first part to reveal any run-time conflicts that might exist in the feature configuration.

A run-time conflict, if it exists, may only occur among the features in an OR feature group. Therefore, the candidate feature configuration is first filtered to contain only features in OR feature groups to reduce the computational complexity of the operation. If a run-time conflict is found, the scale factor and offset calculation algorithm continues its iteration through possible configurations with the next candidate. The additional check for run-time conflicts is therefore expected to increase the number of examined candidate feature configuration. However, these additional candidates are discarded before running the now redundant check for any violations of feature model constraints, which requires a SAT solver. Thus, the performance penalties that come with the use of a SAT solver are avoided.

At the end of this part, the resultant non-normalized minimal and maximal goal satisfaction values (and consequently the scaling factors and offsets) correspond to feature configurations that do not violate both the configuration constraints introduced by the feature model and the run-time constraints expressed by the workflow model.

### **Part 3: Changes to goal model evaluation**

The last part of the algorithm handles the necessary adjustments to the initialization and propagation phases of the goal model evaluation mechanism to accurately represent the overall satisfaction value for a goal model element when constraints introduced by the feature model and the workflow model are both taken into account.

When a software engineer wants to use the reusable artifact, the software engineer selects a valid feature configuration and then determines based on the goal model evaluation whether the chosen feature configuration is appropriate with respect to the high-level goals and the expectation of the software engineer. Hence, this part of the evaluation mechanism must be determined on the fly and interactively with the software engineer. The choice of the software engineer (i.e., the feature configuration and the importance values of high-level goals) is recorded as a design decision, which may later be revisited if the reuse context changes (e.g., new features may be available or importance values of goals have changed).

The algorithm described in Section 5.3 simply initializes the tasks corresponding to the features in the chosen feature configuration with the maximal satisfaction value of 100. However, this is not possible anymore. Recall the example from Figure 5.4. Even though the feature configuration may contain **Retinal Scan** and **Fingerprint** at the same time, these two features conflict at run-time and therefore cannot be selected at the same time when the goal model is evaluated. The adapted evaluation mechanism therefore has to decide whether the satisfaction value of the task corresponding to **Retinal Scan** or **Fingerprint** should be set to 100, but not both.

If any run-time conflict is detected in the feature configuration to be evaluated, the decision of this third part to select one conflicting feature over another may either be based on an optimistic, pessimistic, or probabilistic evaluation. The remainder of this section focuses mostly on the optimistic evaluation, and describes the other two options briefly at the end of the section.

Essentially, the optimistic evaluation selects the features that yields the best evaluation, while the pessimistic evaluation selects the features that yields the worst evaluation. For the probabilistic evaluation, all conflicting features remain in the feature configuration, but their impacts are modulated by the likelihood of occurrence in the operational scenarios.

For the optimistic option, the prior bottom-up evaluation of satisfaction values (i.e., the

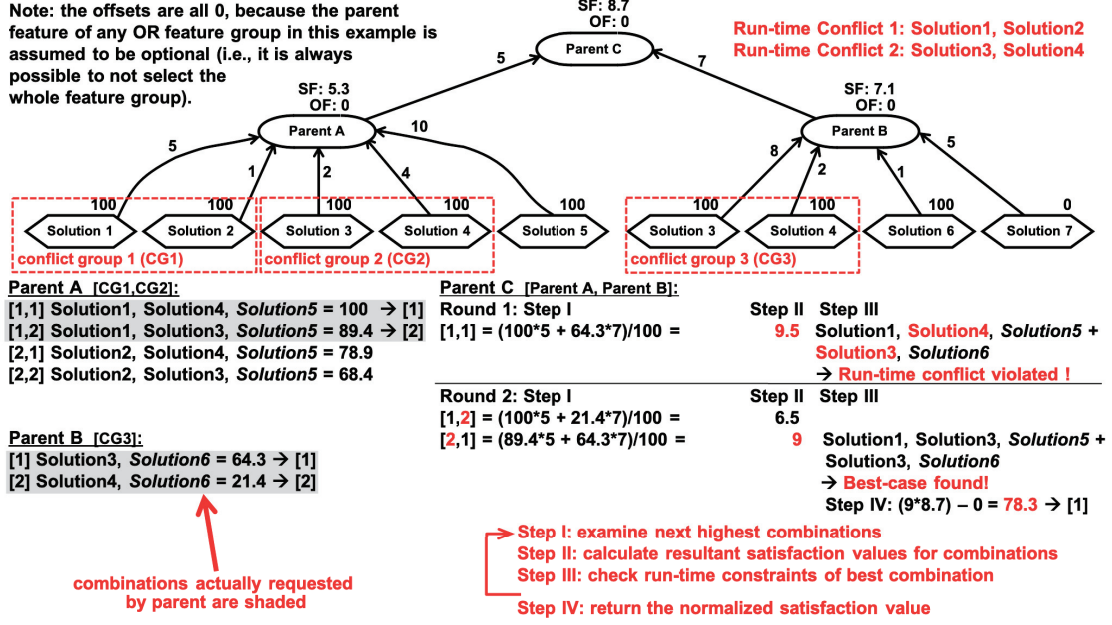


Figure 5.6: Optimistic option for the evaluation algorithm [24]

satisfaction value of a parent is calculated as soon as the satisfaction values of its children are available) is replaced by a recursive evaluation function to be able to react to possible run-time conflicts in the feature configuration that needs to be evaluated. The recursive evaluation function employs a similar lazy backtracking algorithm as described in Algorithm 5.1-5.2 and used in Section 5.3 for the calculation of scale factors and offsets. The main difference is that run-time conflicts now have to be considered instead of feature model constraints imposed on the feature configurations (i.e., in the *checkConfigurationValidity* method of Algorithm 5.1-5.2, line 40). That is, when the chosen feature configuration is already a valid configuration in terms of feature model constraints. Consequently, the need for the SAT solver is eliminated.

Figure 5.6 gives an example of the optimistic selection of features from a group of features with two run-time conflicts. Run-time conflict groups (illustrated with dashed rectangles in

Figure 5.6) are handled at the leaf goal level (e.g., **Parent A** or **Parent B**). A leaf goal may have zero or more conflict groups.

Leaf goals are the first ones to be evaluated. Their satisfaction level is determined based on the best possible contribution from each conflict group in addition to the constant contributions from features without conflicts (e.g., **Solution 5**).

Consider **Parent A** in Figure 5.6, index  $[1,1]$  represents the best possible contributions that could be received from two conflict groups (**Solution 1** is chosen over **Solution 2**, because it contributes 5 instead of 1; **Solution 4** is chosen over **Solution 3**, because it contributes 4 instead of 2). Index  $[1,1]$  for **Parent A** is a two-tuple where each index term corresponds to a contributing conflict group (i.e., **conflict group 1 (CG1)** and **conflict group 2 (CG2)**, respectively). Since the contributions from the features without conflicts are constant (if they are selected), an index term for such contributors is not necessary (e.g., to emphasize that **Solution 5** and **Solution 6** provide constant contributions in Figure 5.6, they are shown in *italic* in the combinations). Consequently, since **Parent B** has only one conflict group contributing to it (i.e., **CG3**), its indexes are one-tuples.

The  $[1,1]$  index of **Parent A** is initially propagated upwards as the best combination for **Parent A**, but higher-level goals might have to consider other run-time conflicts from one of their other children, which in turn may require the selection for **Parent A** to be changed. If this is the case, the next best combination is considered. In the example, this is the index  $[1,2]$ , which selects the best option from the group with **Solution 1** and **Solution 2** (i.e., still **Solution 1**) and the second best option from the group with **Solution 3** and **Solution 4** (i.e., now **Solution 3**).

Figure 5.6 gives an example evaluation for the case where **Solution 1** to **Solution 6** are selected in the configuration. The satisfaction values of selected features are set to 100. The satisfaction value of **Solution 7** is set to 0, because it is not selected.

As illustrated in Figure 5.6, when **Parent C** gets the best combinations giving the highest

contribution (indexed [1,1]) from its children **Parent A** and **Parent B** in the first round (i.e., combination [1] with index [1,1] from **Parent A** and combination [1] with index [1] from **Parent B**), a run-time conflict between **Solution 3** and **Solution 4** occurs within the combined configuration. Hence, a second round is necessary to obtain a configuration that yields the best achievable satisfaction and does not contain any conflicts.

Upon receiving the next best values from both children for the next indexes to be considered ([1,2] and [2,1] are the next best options after [1,1]), the satisfaction values are calculated for these new combinations and the highest one is checked for run-time conflicts. At this point, since the combination with index [2,1] is the highest and it does not contain any conflicts, the resultant satisfaction value for **Parent C** can be obtained by scaling the total contribution of index [2,1] (i.e., combination [2] with index [1,2] from **Parent A** and combination [1] with index [1] from **Parent B**). Therefore in Step IV, the contribution of 9 is scaled by 8.7 to yield 78.3 as the overall satisfaction value and best combination for **Parent C**.

To summarize, the optimistic option of the adapted evaluation for satisfaction values is a lazy algorithm that recursively determines the best-case achievable satisfaction values for each node with the given feature configuration.

If the goal model were bigger and **Parent C**'s parent had requested the next best satisfaction value, the algorithm would continue with further iterations of Steps I-III to determine the next best-case satisfaction value. Theoretically in the worst case, all possible combinations of all goals may have to be examined, but in practice this is rarely the case. For example, only two out of the four possible combinations of **Parent A** were needed to calculate the satisfaction value of **Parent C**. Therefore, the combinations are calculated in a recursive manner on demand.

The pessimistic option requires only a small change to the algorithm for the optimistic option. The pessimistic option only reverses the order of the combinations and ranks the



one that yields the lowest satisfaction value at the top.

The probabilistic option, on the other hand, does not even need the recursive algorithm, but works with the existing, bottom-up approach with an additional pre-processing stage. During that pre-processing stage, the contributions of a feature with a conflict are multiplied by the probability of occurrence in the operational scenario. This, however, requires the path extraction algorithm to be extended to first annotate each path node in the workflow model with a probability of occurrence, which may either be estimated or may be derived from observation of the running system.

## 5.5 Proof-of-Concept Implementations

A proof-of-concept implementation of the novel goal model evaluation mechanism as described in this chapter, including the revised scale factor and offset calculation algorithm, has been implemented in the TouchCORE tool [27], which combines feature and goal modeling to build reusable artifacts called concerns in support of CORE [20].

Since the TouchCORE tool does not yet support workflow modeling, the determination of run-time conflicts through path extraction as described in Section 5.4.2 is done on workflow models (i.e., Use Case Maps) that are created with the jUCMNav tool [227]. The resulting conflict table is made available to the TouchCORE tool.

### 5.5.1 Implementation Workflow

Figure 5.7 illustrates the complete implementation workflow. The *Path Extraction* component reads the Use Case Map file from jUCMNav (e.g., Figure 5.8) and the *Feature Model* file from TouchCORE (e.g., Figure 5.9) and determines the *Conflict Table*.

**Example 7.** When the path extraction is applied on the workflow model in Figure 5.8, the

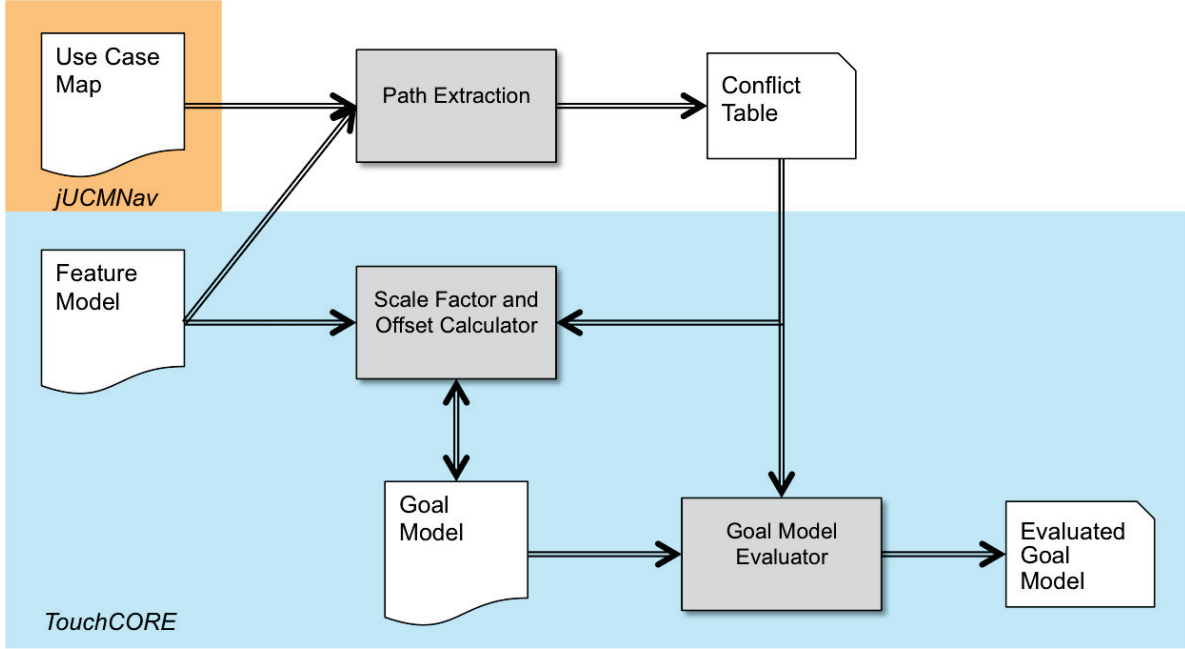


Figure 5.7: Implementation workflow [24]

resultant conflict table identifies run-time conflicts between F10 and F11, F23 and F25, and F24 and F25 as they reside on alternative branches. Other action pairs that lie on alternative paths in Figure 5.8 are not listed as they are not actions that are linked to features in an OR feature group. In this example, F10 and F11 are OR children of `newChild8` whereas F23, F24, and F25 are OR children of F16 as expressed in the feature model in Figure 5.9.

The *Scale Factor and Offset Calculator* considers the *Feature Model* (e.g., Figure 5.9) and the *Conflict Table* as well as the current *Goal Model* (e.g., Figure 5.10) for its calculations, and then updates the *Goal Model* accordingly. The *Scale Factor and Offset Calculator* uses the constraint check mechanism to verify that the feature configurations for the minimal/maximal cases do not violate the run-time constraints imposed by the workflow model in addition to feature model constraints. Because of the additional constraints, it is expected that the revised algorithm will need to examine more combinations, which is indeed generally

the case as will be discussed in Section 5.5.3.

Finally, the *Goal Model Evaluator* analyzes the goal model, now taking the *Conflict Table* into consideration.

Figures 5.8-5.10 show one of the test cases (TC7+F+R) used for the performance evaluation later in this section. When the modeler makes a feature selection, the TouchCORE tool shows the selected features in green and with a checkmark as shown in Figure 5.9. Features that are not selected are shown in grey and without a checkmark.

The tool also shows the results of the goal model evaluation both numerically and with color-coding as can be observed in Figure 5.10 (for features, green means it is selected and grey means it is not selected; whereas for goals, bright green represents the best case (100) and bright red represents the worst case (0)). Note that the feature selection shown in Figure 5.9 and Figure 5.10 is the same, i.e., the goal model is evaluated according to the feature selection in Figure 5.9.

Figure 5.11 zooms into the evaluation of a single goal (GoalF) to highlight the changes to the goal model evaluation based on run-time conflicts.

**Example 8.** In Figure 5.11, without considering run-time constraints, a maximal satisfaction of 8 for GoalF is achieved when features F10, F11, and F19 are selected together, which yields a scale factor of 12.5 (100/8). Note that F4 is in an XOR relationship with F10 and F11 as imposed by the feature model in Figure 5.9 (an OR feature group is shown with a filled triangle, while an XOR feature group is shown with a non-filled triangle in TouchCORE).

When run-time constraints are taken into account, F10 and F11 have a run-time conflict due to their appearance on alternative branches as shown in Figure 5.8, which reduces the maximal satisfaction for GoalF to 7 (3 from F11 and 4 from F19) and yields a scale factor of 14.286 (100/7). When the goal model is evaluated with the given feature configuration, although both F10 and F11 are selected, their impact cannot be additive due to the run-time

## 5.5. Proof-of-Concept Implementations

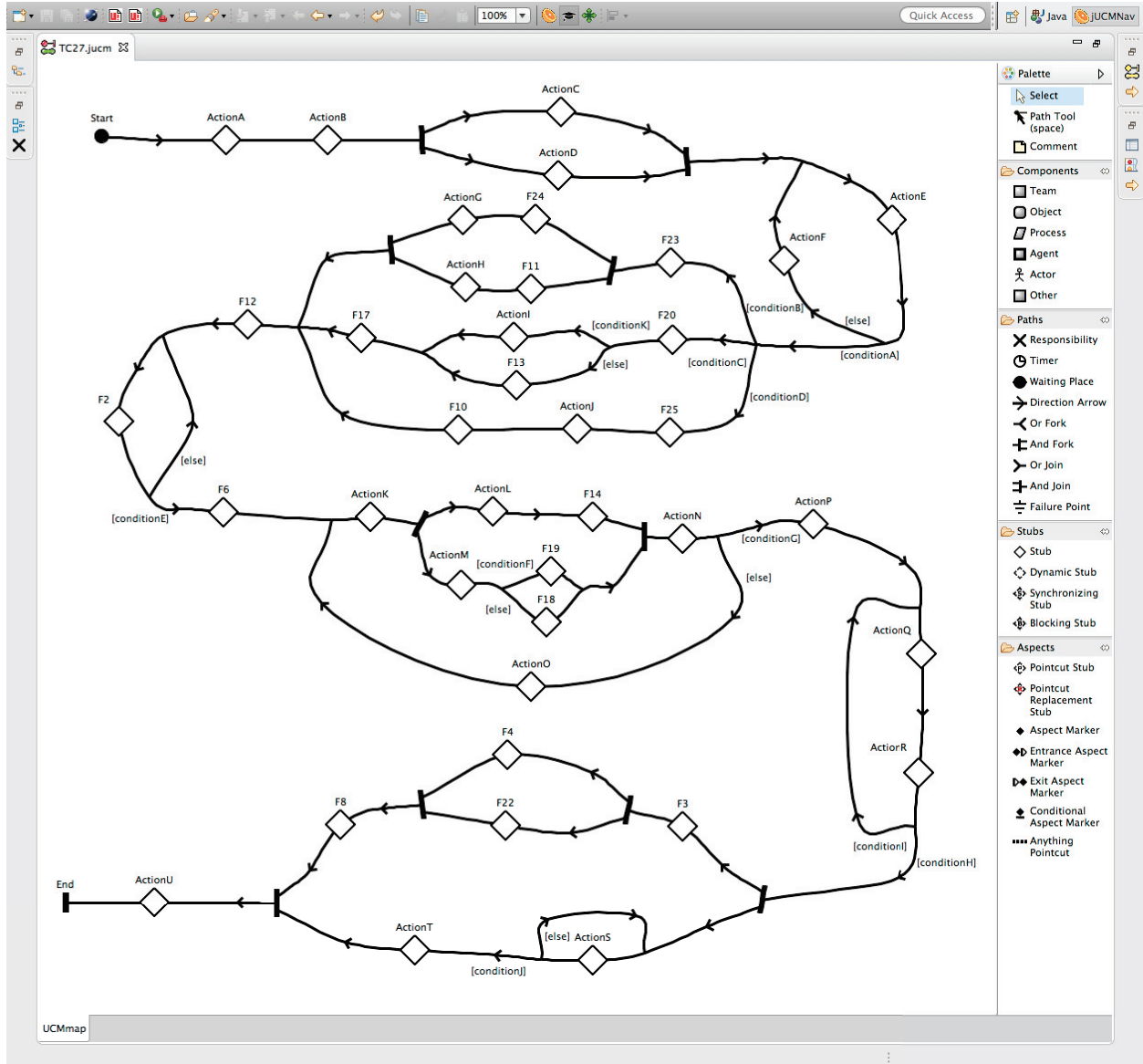


Figure 5.8: Use Case Map for “TC7 + F + R” in the jUCMNav tool [24]

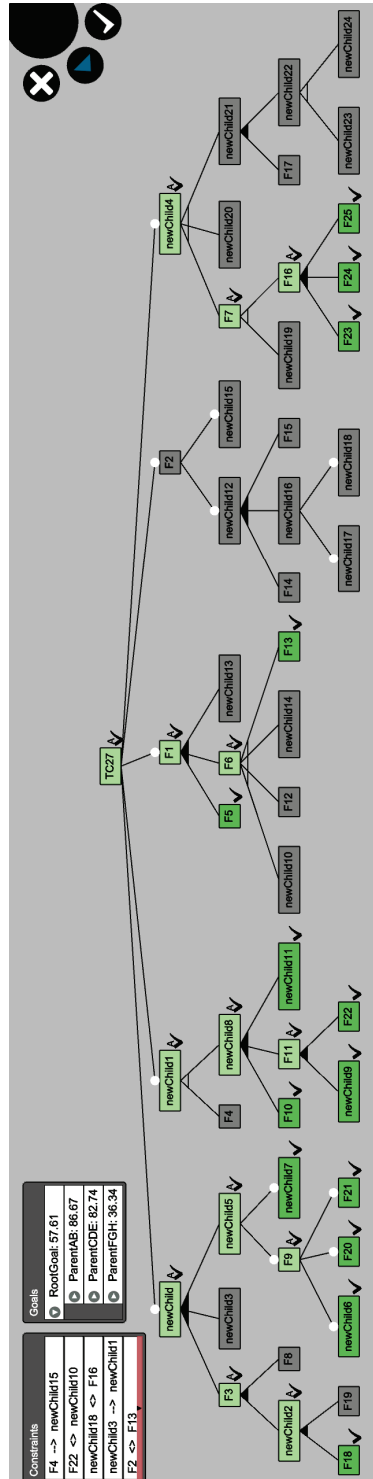


Figure 5.9: Feature Model for “TC7 + F + R” in the TouchCORE tool [24]

conflict between them. Consequently, the evaluated satisfaction value for **GoalF** is 42.86 ( $14.286 * 3$ ) instead of 50.00 ( $12.5 * 4$ ).

The performance evaluation of the proof-of-concept implementation in this section focuses on the second part of the improved algorithm, because the first part responsible for handling the path extraction is done independently when the feature or workflow model changes, i.e., at design-time of the reusable artifact and not when the artifact is reused. In fact, the first part could only be run once just before shipping the reusable artifact.

Additionally, the performance of the first part solely depends on the number of alternative paths, as each node in the workflow model is always visited only once during path extraction. Therefore, the performance of the path extraction algorithm – even though path extraction may be time-consuming for some workflow models – is not expected to have a drastic impact on the modeling experience and consequently is not the focus of this analysis.

The third part, on the other hand, is important to the modeling experience, because it affects the reuse of an artifact as opposed to the design of the reusable artifact. The third part handles the goal model evaluation based on a feature configuration chosen by the software engineer who wants to reuse the artifact.

The second and the third part both operate on the same feature, workflow, and goal model with the same feature model and run-time constraints (i.e., conflict table). Furthermore, both parts employ a top-down, recursive, back-tracking algorithm that is based on the lazy evaluation of maximal and minimal sums. The key difference between the second and the third part is that the search for the most appropriate combination is simpler for the third part than it is for second part. The second part has to consider that each feature is either selected or not selected (see the two options for each feature in Figure 5.3). The third part, however, knows exactly which feature is selected and which one is not (see the satisfaction values of the features in Figure 5.6), and therefore there is only one contribution value for

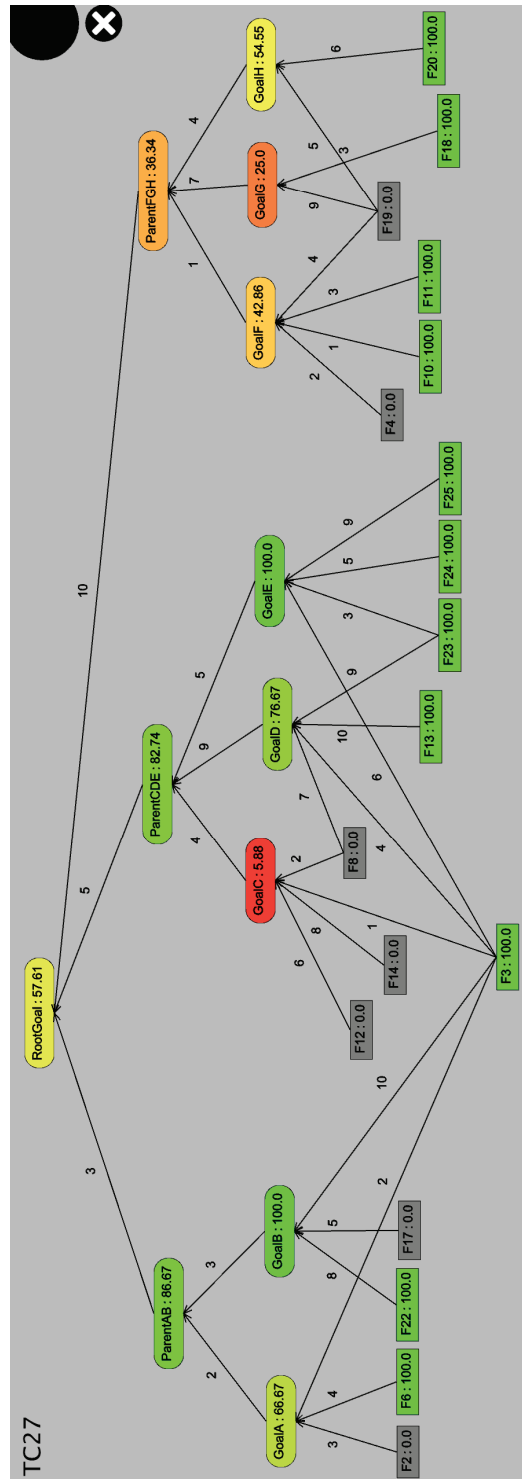


Figure 5.10: Goal Model evaluation for “TC7 + F + R” in the TouchCORE tool [24]

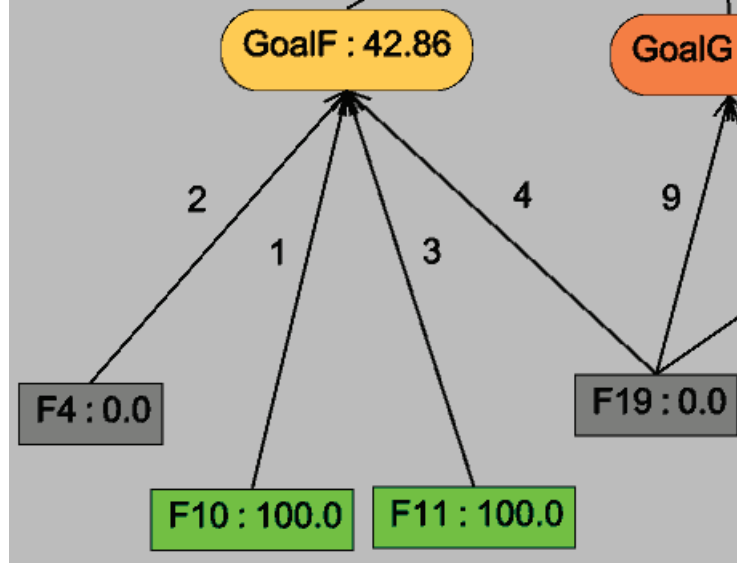


Figure 5.11: Evaluation of a single goal in the presence of run-time constraints [24]

each feature that needs to be considered instead of two. This effectively cuts the search space in half. Furthermore, feature constraints do not need to be considered anymore for the third part, because the feature configuration chosen by the software engineer is valid. Only run-time constraints have to be considered for both parts.

Consequently, the performance of the second part is a clear upper bound for the performance of the third part, especially since there is no need for a SAT solver for the third part. Therefore, the performance results of the second part of the algorithm are reported here, even though the second part is only needed at design-time of the reusable artifact and not when the artifact is actually reused.

### 5.5.2 Design of the Test Cases

The test cases for the performance evaluation are aimed to extend typical models we encountered when building reusable artifacts (e.g., *Authentication*, *Authorization*, *Resource*



*Management*, and *Transactions*) in size. Therefore, to work with models that are larger than all of the models we typically use to build reusable artifacts, test cases are designed to have a feature model with 50 features and a goal model with one root goal and a maximum depth of 3. A goal model used in a test case contains a total of 30 goal model elements; 12 goals that are all labelled *<<run-time>>* and 18 tasks. Furthermore, 8 additional constraints are introduced to goal models as includes/excludes constraints to further increase the complexity.

For the representation of the run-time interactions among features, workflow models were implemented as Use Case Maps consisting of 8 alternative forks ( $\neg$ ), 5 concurrency forks ( $\vdash$ ), and 5 loops. 18 stubs ( $\diamond$ ) representing the contributing features from the goal model are used in the Use Case Maps in addition to twenty additional action stubs.

The test cases reported here go beyond the expected model size of typical reusable artifacts we had encountered in terms of the number of features, goals, feature constraints, as well as run-time constraints.

Tests were run on a total of 27 test cases that were generated as follows. 8 different feature models are evaluated (TC1-8):

**TC1** - all links are optional links

**TC2** - all links are OR links

**TC3** - all links are XOR links

**TC4-8** - random link types out of optional, OR, and XOR links for each link

7 additional feature models are derived by adding 8 random includes/excludes constraints to 7 of the first 8 test cases (TC1+F to TC8+F, with the exception of TC3 where all links are XOR links, because additional constraints do not make sense in this case).

6 more feature models (TC2+R, TC4+R to TC8+R) are derived by introducing as many run-time conflicts as possible to the five test cases with random links and the test case with all OR links. Due to the structure of the feature model in those test cases (since run-time conflicts appear only in OR feature groups) and the random occurrence of features in the goal models, it is possible to add between 1 and 4 run-time constraints (i.e., 4 constraints to TC2, 1 to TC4, 4 to TC5, 1 to TC6, 3 to TC7, and 1 to TC8).

Finally, 6 more feature models (TC2+F+R, TC4+F+R to TC8+F+R) are derived by introducing the 8 includes/excludes constraints to the test cases with run-time conflicts (TC2+R, TC4+R to TC8+R).

### 5.5.3 Performance Evaluation Results

The results of the performance evaluation on a computer with a 2.5 GHz processor and 16GB, 1600 MHz, DDR3 memory are reported in Table 5.1 with java execution time given in milliseconds (see *2nd* column; note that the time reported is the average of the last 100 out of 110 runs to discount for program startup time).

Test cases without feature constraints and run-time constraints were generally handled in a few 100 milliseconds (TC1-8).

Once constraints have to be considered (TC1+F, TC2+F, TC4+F to TC8+F, TC2+R, TC4+R to TC8+R), execution time is more unpredictable, depending heavily on the particularities of a test model. Whilst the execution time for the test cases with only run-time constraints (TC2+R, TC4+R to TC8+R) remains similar to the test cases TC1-8, it certainly increases for the test cases with feature constraints (TC1+F to TC8+F and TC2+F+R, TC4+F+R to TC8+F+R), while still staying within acceptable margins.

Our measurements show that the number of calls to the SAT solver (see *4th* column) has the most significant impact on the execution time. Therefore, a check for simple conflicts is

## 5.5. Proof-of-Concept Implementations

Table 5.1: Performance Results for Scale Factor and Offset Calculation

	Measurements											
	Exec.	Detected	Calls to	Calculated		Unique req.		Total req.		Ratio		Detected
	time	simple FM	SAT	combinations		combinations		combinations				run-time
	(ms)	conflicts	solver	Goal	Feature	Goal	Feature	Goal	Feature	Goal	Feature	conflicts
TC1	94	0	24	24	72	24	36	46	52	1.0	2.0	0
TC2	131	0	24	24	72	24	36	46	52	1.0	2.0	0
TC3	629	7135	24	9227	72	7358	54	34663	1302	1.3	1.3	0
TC4	111	38	24	220	72	107	54	506	309	2.1	1.3	0
TC5	107	1	24	28	72	25	40	47	68	1.1	1.8	0
TC6	145	1481	24	3053	72	1765	54	10375	571	1.7	1.3	0
TC7	67	181	24	606	72	299	54	1774	420	2.0	1.3	0
TC8	102	329	24	862	72	469	54	2666	464	1.8	1.3	0
TC1+F	380	174	46	656	72	325	54	1932	420	2.0	1.3	0
TC2+F	623	174	46	656	72	325	54	1932	420	2.0	1.3	0
TC4+F	345	187	37	659	72	327	54	1945	420	2.0	1.3	0
TC5+F	1026	185	39	659	72	327	54	1945	420	2.0	1.3	0
TC6+F	1357	6659	43	11195	72	7248	54	40002	699	1.5	1.3	0
TC7+F	320	79	27	219	72	131	50	566	340	1.7	1.4	0
TC8+F	3826	11430	78	17092	72	12116	54	62491	706	1.4	1.3	0
TC2+R	97	0	24	134	72	79	48	299	269	1.7	1.5	40
TC4+R	138	38	24	221	72	108	54	507	312	2.0	1.3	1
TC5+R	103	1	24	289	72	188	48	838	371	1.5	1.5	134
TC6+R	92	1081	24	3063	72	1776	54	10419	571	1.7	1.3	413
TC7+R	57	92	24	844	72	426	54	2583	464	2.0	1.3	190
TC8+R	55	222	24	779	72	423	54	2375	464	1.8	1.3	65
TC2+F+R	1021	127	46	2296	72	1235	54	7604	573	1.9	1.3	811
TC4+F+R	770	187	37	660	72	328	54	1946	423	2.0	1.3	1
TC5+F+R	440	12	27	616	72	448	50	2064	465	1.4	1.4	374
TC6+F+R	2267	6898	38	13918	72	9224	54	50143	704	1.5	1.3	1741
TC7+F+R	206	52	26	292	72	187	50	819	410	1.6	1.4	80
TC8+F+R	3195	7739	68	15223	72	10622	54	55394	706	1.4	1.3	2211

TC1: only optional links in feature model;

TC2: only OR links in feature model;

TC3: only XOR links in feature model;

TC4–8: randomly assigned links in feature model;

F: feature constraints added;

R: run-time constraints added;

FM: Feature Model

done before using the SAT solver (see *3rd* column; e.g., if the nearest parent of two selected features is the parent of an XOR group, the two features cannot be selected together). This simple check, in turn, decreases the number of calls to the SAT solver significantly.

The test data does not allow concluding whether feature constraints or run-time constraints contribute more to longer executions times, but if both types of constraints exist, the resulting execution time is significantly longer than for the individual test cases.

The table reports on requested combinations (both unique and total requests) and calculated combinations (see *5th* to *10th* columns). As expected, the results for the latter are never lower than the results for the former, because several combinations have to be calculated to determine one requested combination. For the evaluation of features, two possible combinations (i.e., feature being selected or not-selected) are calculated up front at the beginning of the algorithm. Therefore, the number of calculated feature combinations (see *6th* column) remains the same for all test cases (18 contributing features, 2 combinations per feature, 2 evaluations (maximal/minimal) results in  $18 * 2 * 2 = 72$ ).

The example in Figure 5.3 illustrates that the number of calculated combinations is higher than the number of requested combinations, i.e., 6 combinations have to be calculated for 3 requested combinations.

The example in Figure 5.6 shows the difference between unique and total requested combinations. In Round 1 and Round 2 together, **Parent C** requests 3 combinations from its two children **Parent A** and **Parent B** ( $[1,1]$ ,  $[2,1]$ ,  $[1,2]$ ), i.e., 6 requested combinations in total (i.e., combination  $[1]$  is requested twice from **Parent A**, combination  $[2]$  once from **Parent A**, combination  $[1]$  twice from **Parent B**, and combination  $[2]$  once from **Parent B**). These 6 requested combinations include 4 unique requested combinations, i.e.,  $[1]$  and  $[2]$  from **Parent A** and also  $[1]$  and  $[2]$  from **Parent B**).

When the requested and calculated combinations are compared among different test cases, it can be observed that introducing constraints (feature and/or run-time) usually

increases the number of combinations as each additional conflict that is detected results in requesting and calculating more combinations. When TC7+F+R is compared to TC7, it can be seen that despite the introduced constraints, the number of requested and calculated combinations remains lower for TC7+F+R due to the decrease in total number of detected conflicts (see 3rd and 13th columns).

The columns for the combinations requested by the parent and the combinations that had to be calculated show that they are significantly less than the combinations of an exponential brute-force approach, hence indicating the effectiveness of the lazy, recursive algorithm. In any case, the figures for the requested/calculated combinations, detected run-time conflicts (see last column), and the actual calls to the SAT solver indicate the reduction in calls to the SAT solver, i.e., not all calculated combinations require a call to the SAT solver.

Finally, the ratio of calculated vs. unique requested combinations (see 11th and 12th columns) gives information about the extent of additional combinations calculated to find the desired combination that satisfies the feature and run-time constraints. A larger ratio indicates that on average more combinations had to be examined to report one requested combination. In other words, the algorithm may have had to go through many combinations that violated run-time constraints to find one combination that does not and can be reported as the requested combination.

Considering that the chosen size of feature and goal models and the number of constraints go beyond the common and typical models we have encountered, the results indicate that the proposed goal model evaluation mechanism is feasible. In general, the (in some cases) longer execution time is a trade-off against a more accurate evaluation of the goal model, and hence a more accurate representation of the impact of a reusable artifact on high-level stakeholder/system goals.

## 5.6 Summary

Software reuse relies heavily on the ability to determine the most appropriate artifact given a reuse context and to assemble reusable artifacts in reuse hierarchies. Since software systems are often defined with the help of many different modeling languages, the decision to reuse an artifact should use all relevant information available about the artifact. Additionally, the decision to reuse a candidate artifact, the context in which the artifact is reused, and why the artifact is reused need to be captured in case the decision needs to be revisited. In this chapter, a novel goal model evaluation mechanism for the selection of the most appropriate candidate is presented, which allows a history of design decisions about reusable artifacts to be kept.

The proposed approach uses goal models, which are used to characterize different candidate solutions according to the impacts they have on high-level stakeholder/system goals. The reasoning about high-level goals, non-functional requirements, system qualities, and candidate solutions is provided by an evaluation mechanism that uses relative and quantitative contribution values and determines normalized satisfaction values for high-level goals.

For the representation of various relationships between candidate artifacts, different types of modeling notations such as feature models and workflow models are often used in collaboration with goal models that assess these candidate solutions. The proposed novel goal model evaluation mechanism therefore takes the constraints imposed by feature and workflow models on goal models into account to assess the impact of reusable artifacts on high-level goals more accurately. While feature models express which reusable artifacts may exist together in an application, workflow models capture whether artifacts occur together with other artifacts in the same operational scenario. Considering the constraints of both models in the goal model evaluation increases the accuracy of the evaluation result, i.e., the decision which reusable artifact to choose.

Combined with feature models, goal models allow reuse decisions to be captured by a software engineer: (i) a feature configuration captures the choice of reusable artifact (e.g., which authentication means should be used in the application under development); (ii) the goal model captures the importance of various high-level goals (i.e., the reuse context) and the actual impact of the chosen reusable artifact on high-level goals (i.e., why, for example, one authentication means was chosen over another). These reuse decisions may therefore be revisited, if either the reusable artifacts or the importance of high-level goals changes.

On account of the proposed approach presented in this chapter, our approach fully satisfies the third requirement (R.3) of taking constraints imposed by other modeling notations into account when evaluating reusable goal models. Furthermore, the lazy, recursive algorithm allows flexible integration of additional constraints that may be imposed on goal models by other modeling formalisms in the future.

The performance evaluation of the proof-of-concept implementation for the novel evaluation mechanism demonstrates feasibility and shows that the introduced performance overhead is affordable considering the greater accuracy of the proposed approach.

While this chapter focused on dealing with external constraints imposed on the goal model, the following chapter presents our solutions that allow the evaluation mechanism to be applied to reuse hierarchies of goal models.

# Chapter 6

## Goal Models in Reuse Hierarchies

In order to benefit from the trade-off capabilities of goal modeling in the context of reuse to its fullest, it is essential to make the evaluation mechanism compliant with reuse hierarchies. This chapter first gives an overview of Concern-Oriented, which is used in this work to build the desired hierarchies from reusable artifacts containing goal models, in Section 6.1. Section 6.2, then, presents the novel modeling constructs introduced to achieve the desired modularity through a clearly defined reuse interface in goal models. Section 6.3 describes how the goal model evaluation can be done with the help of this newly introduced reuse interface, followed by the two novel goal model evaluation algorithms that (i) handle delayed decisions throughout the hierarchy with a range evaluation in Section 6.4, and (ii) provide the means to find the optimal set of solutions at the top of the reuse hierarchy with a top-down evaluation in Section 6.5. A discussion on how contextual information can be modeled in reusable goal models and how our range evaluation provides the required support for the analysis of contextual information in goal model reuse hierarchies is presented in Section 6.6. Section 6.7 summarizes the chapters contributions and concludes the chapter.

### 6.1 Reuse Hierarchies with Concern-Oriented

Concern-Oriented [20] is a software reuse paradigm that builds on the ideas of Model-Driven Engineering [228], Software Product Lines (SPLs) [229], and advanced Separation of Concerns (SoC) [230].



Concern-Orientation aims to build large reuse hierarchies with the help of smaller reusable artifacts called *concerns*. A concern is a broad, generic reusable artifact that contains all models required to fully understand the domain of the concern, from requirements and analysis models to design models to testing models and eventual implementations. Thus, a concern often spans multiple phases of software development and levels of abstraction.

In Concern-Orientation, a concern has three interfaces: the variation interface, the customization interface, and the usage interface. The three-part interface of a concern promotes modularity and reuse.

The variation interface of a concern presents - in a feature model - the available design choices and functional variants offered by the concern. Also as part of the variation interface, the impact of alternative solutions (i.e., features) on system qualities and high-level stakeholder goals is shown in a goal model called impact model [20][26] using relative contributions as introduced in Chapter 4. Features provided by the concern are represented as tasks in its goal model as explained in Section 5.3.

The variation interface allows the modeler to select the most appropriate features among those encapsulated within the concern, and reason about the impact of the selection on high-level goals and system properties. Inspired by the URN [7], Concern-Orientation uses propagation-based reasoning for the goal model of its variation interface. Reasoning about the appropriateness of a reusable artifact is important, because an incorrect understanding of a reusable artifact may lead to catastrophic failures [231].

For example, a security concern may offer various means of authentication, from password-based to biometrics-based solutions, each with differing impacts on the level of security as well as cost and end-user convenience. These qualities have to be weighed, when determining which authentication feature is most appropriate in the context where the security concern is reused. Therefore, the goal model describing the variation interface must be analyzed with an evaluation mechanism to perform the required trade-off analysis.

The remaining two interfaces are only needed once the most appropriate features of a concern have been selected with the help of the variation interface, and are therefore irrelevant for this thesis. Briefly, the customization interface allows a generic concern to be adapted to the concrete application context, while the usage interface finally provides access to the structure and behavior provided by the concern.

While the creation of a concern is a non-trivial, time-consuming endeavor because of the required extensive domain expertise, the reuse of the concern is a straightforward three-step process.

First, the features with the best impact on system qualities are selected with the help of the variation interface. Based on the selection, a tailored concern is created that includes only those models that are relevant for the selected features.

Second, the customization interface is used to adapt the tailored models to the application context by specifying which generic elements of the concern are related to concrete elements of the application under development.

In the final third step, the structure and behavior of the tailored and customized concern can now be accessed through the usage interface by the application under development.

Because a concern is a modular reusable artifact that is assembled in concern hierarchies, the goal model that expresses the variation interface of the concern also needs to be modular with clearly defined interfaces.

Furthermore, it must be possible to apply the evaluation mechanism of the goal model to modular goal models in concern hierarchies. As the reuse hierarchy grows and goals of lower level goal models impact those of higher levels, the computational complexity of the evaluation potentially increases.

However, propagation-based evaluation algorithms of existing goal modeling notations [5][6][7] assume a monolithic goal model, and hence are currently not capable of supporting goal-oriented reasoning in the context of concern-oriented reuse. Since evaluating a flattened,

monolithic goal model leads to performance problems especially when external constraints are imposed on goal models, it becomes important to take advantage of modularity and reuse boundaries to manage computational complexity.

Another issue with designing a reusable artifact is that it requires keeping the artifact generic as the context in which the artifact will be used is not known at design time. When reusing a goal model at a lower level in the goal model reuse hierarchy, designers may choose not to make decisions on selecting some tasks. The remaining task selections are called *delayed decisions* because they are postponed to a higher level in the reuse hierarchy when more information is available about the system under development [23]. However, not knowing the state of some tasks (i.e., selected or not selected) conflicts with existing propagation-based evaluation algorithms for goal models [22][225]. Therefore, for goal modeling to be a viable tool in the context of reuse, goal model evaluation needs to be updated to the presence of delayed decisions for continued trade off analysis capability.

Decisions are delayed because the designer does not have sufficient context dependent information available when the reusable artifact is built. For the analysis and evaluation of reusable goal models, considering delayed decisions means considering the full range of uncertainty that exists because of missing contextual information. As a consequence of this, providing support for goal model evaluation with delayed decisions also implies providing support to analyze context dependent information in a goal model reuse hierarchy.

## 6.2 Reuse Interface

Goal catalogues [16] are not appropriate for concern hierarchies because of three reasons.

First, when a goal catalogue is reused, its complete goal model is imported into the current goal model, leading to extremely large and unwieldy goal models as lower-level goal models are copied into higher-level goal models. This is because the goal model of the lowest-

level concern would first be copied into the goal model of the next-higher concern, and then the combined goal model from both concerns would have to be copied into the goal model of the next-next-higher concern, and so on until the concern at the top of the concern hierarchy contains the goal models of all lower-level concerns in one monolithic goal model. Clearly, this is hard to maintain, use, and analyze.

Second, reuse techniques based on goal catalogues cannot differentiate goal model elements of the reused lower-level concern from goal model elements of the current concern. A lower-level goal model element looks just like a goal model element defined by the current concern once the lower-level goal model elements have been copied into the current concern. Furthermore, these goal model elements are also not differentiated at the metamodel level. Therefore, it is not possible to clearly define a reuse interface for the goal model, which would allow hiding lower-level goal model elements behind the interface of the current concern from the next-higher concern wanting to reuse the current concern. Instead, the next-higher concern has access to all goal model elements from all goal models that reside below it in the concern hierarchy. In Concern-Oriented, however, the variation interface of the concern is supposed to be the interface that hides lower-level goal model elements to properly encapsulate goal models.

The third reason will be explained after introducing the novel goal modeling constructs, which enable modular reuse of lower-level goal models in the context of Concern-Oriented, with the help of the example models in Figure 6.1.

The first of the two new modeling constructs is the *reuse link* ( $\text{---}\text{R}\text{---}$ ). At the top of Figure 6.1, the use of the reuse link is shown for a GRL feature model [225]. In GRL, a GRL feature model is a special case of a goal model, because essentially a feature model is an AND/OR tree while a goal model is an AND/OR graph. Links in GRL feature models are interpreted as contribution links. While GRL feature models do not contain any goal model-specific elements, a goal model may include features as tasks to represent their impact on

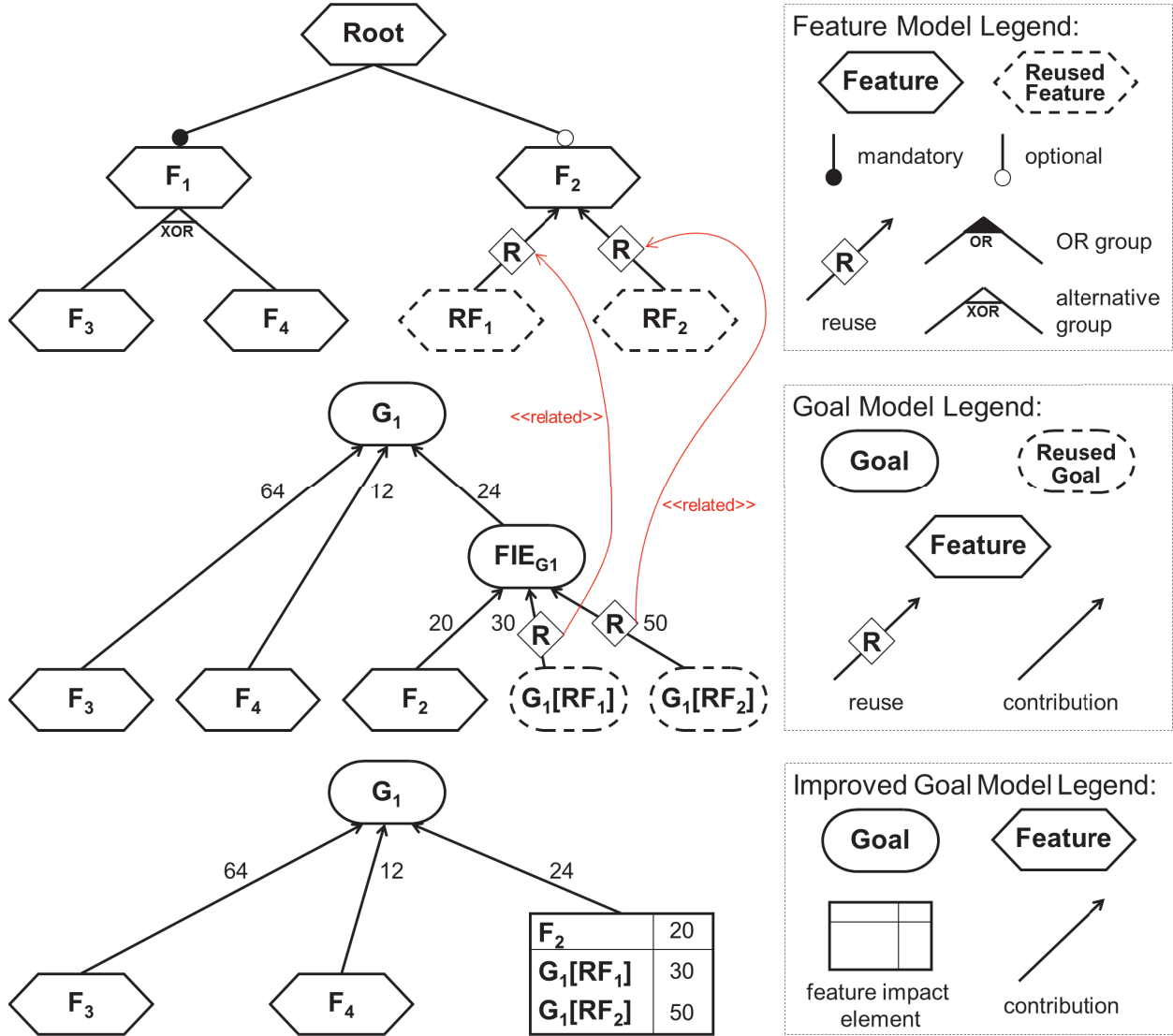


Figure 6.1: Reuse hierarchy in feature (top) and goal (middle and bottom) models [29]

some goal model elements [225]. The GRL feature model in Figure 6.1 has two child features of the **Root** feature (a mandatory  $F_1$  and an optional  $F_2$ ) and features  $F_3$  and  $F_4$  in an alternative feature group for parent feature  $F_1$ .

A reuse link connects a reused feature (the source of the link) with the feature from the current feature model that is reusing the reused feature (the target of the link). Therefore, the feature model in Figure 6.1 states that feature  $F_2$  is reusing two features  $RF_1$  and  $RF_2$  from two different lower-level concerns (i.e.,  $RF_1$  and  $RF_2$  are typically the root features of their respective concerns).

A reuse link in the feature model is similar to a mandatory link, because as long as the parent feature  $F_2$  is selected, the child features  $RF_1$  and  $RF_2$  representing reused elements must be selected, too. Consequently, if  $F_2$  has a choice on whether to use the reused elements  $RF_1$  and  $RF_2$ , then this choice is to be modeled with two sub-features of  $F_2$ , one reusing  $RF_1$  and  $RF_2$  with reuse links and the other one not.

Reused goal model elements only have to be considered when a reuse link is specified in the feature model, i.e., a reuse link in the goal model is always related to a reuse link in the feature model.

The GRL goal model (middle of Figure 6.1) shows the use of the reuse link for GRL goal models. The goal model shows the impact of features on one particular system quality (i.e.,  $G_1$ ). One such impact model exists for each system quality  $G_n$  of interest. In this case,  $F_3$  impacts  $G_1$  more than five times as much as  $F_4$  (64 vs 12), whereas the intermediate goal  $FIE_{G_1}$  impacts  $G_1$  twice as much as  $F_4$  (24 vs 12).

The interesting part of this goal model is the sub-graph below the  $FIE_{G_1}$  goal, which makes use of the new reuse link concept. Because  $F_2$  reuses  $RF_1$  and  $RF_2$  in the GRL feature model, the impacts of the two lower-level concerns on goal  $G_1$  have to be taken into account when reusing the concerns. Therefore, the corresponding goals from the lower-level concerns impact the intermediate goal  $FIE_{G_1}$ . In addition, feature  $F_2$  itself may also impact goal  $G_1$

and hence also contributes to the intermediate goal. In the example, the concern of  $RF_2$  has two and a half times as much impact as feature  $F_2$  (50 vs 20), whereas the concern of  $RF_1$  has one and a half times as much impact as  $F_2$  (30 vs 20).

As the weights on the reuse link in the example indicate, a reuse link in a GRL goal model propagates the satisfaction value from the child to the parent just like a contribution link and it is treated as such by the extended evaluation mechanism. The intermediate goal  $FIE_{G_1}$  is used to be able to clearly differentiate the impact of the reused elements on the feature itself for a particular goal and the impact of the whole feature (including the reused parts) on the overall system. The latter specifies the contribution of the whole feature (i.e., 24 in Figure 6.1) to the overall goal compared to other features of the system (i.e., 64 and 12). The former specifies how much reused elements (i.e., 30 for  $G_1[RF_1]$  and 50 for  $G_1[RF_2]$ ) vs. non-reused parts (i.e., 20 from  $F_2$ ) contribute to the feature itself. This clear differentiation between the contribution of the feature and its reused parts allows reused parts to evolve over time and their contributions to be updated accordingly.

Similar goal models are specified for each other system quality  $G_n$  of interest, i.e., the goal model elements  $G_n[RF_m]$  related to the system quality of interest from the reused concerns are connected with reuse links to an intermediate goal  $FIE_{G_n}$ , which is also connected to the reusing feature  $F_x$ , and then propagates the combined result of the whole feature including reused elements and non-reused parts to the goal model element  $G_n$  at the top. While the reuse links in the goal model can be automatically added given the intermediate goal  $FIE_{G_n}$  with its feature  $F_x$  and system quality  $G_n$ , the weights on the reuse links need to be specified by the modeler who can assess the impact of the reused parts.

To summarize, a reuse link connects either a reusable feature model from a lower-level concern with the current feature model or a reusable goal model from a lower-level concern with the current goal model without having to import the complete reusable model into the current model. Consequently, feature and goal models do not get unnecessarily large in the

concern hierarchy. This is possible, because the model elements of a feature model diagram or goal model diagram are actually references to the definitions of the elements. In the case of a reused model element, this definition does not exist in the current model, but in the model of the reused concern (i.e., the reference points to an element in a different model instead of the current model).

An intermediate goal  $FIE_{G_1}$  is used to aggregate the impacts of the reused concerns and the impact of the reusing feature, because the reusing feature may or may not be selected by the modeler. A reuse link in the goal model is clearly related to a reuse link in the feature model in that only if a reuse is indicated in the feature model does it make sense to consider the impact of the reused concern on a system quality.

For example in Figure 6.1, the reuse link of  $G_1[RF_1]$  in the GRL goal model is related to the reuse link of  $RF_1$  in the GRL feature model and, similarly, the reuse link of  $G_1[RF_2]$  is related to the reuse link of  $RF_2$ . Consequently, the intermediate goal  $FIE_{G_1}$  only needs to be taken into account if the corresponding reuse link in the feature model is active (i.e., the reusing feature is selected).

On the other hand, if a reuse link in the feature model is not active (e.g.,  $F_2$  is not selected), then the corresponding reuse links in the goal model are also not active. Because  $F_2$  is automatically not active in the goal model if it is not selected in the feature model, none of the children of the intermediate goal  $FIE_{G_1}$  are active, which means the intermediate goal is also not active and does not contribute to the satisfaction of the system quality (i.e., it contributes 0).

Therefore, it is possible to disregard parts of the goal model related to a feature that is not selected. Goal catalogues do not allow this, which is the third reason for why goal catalogues are not appropriate for concern hierarchies.

Because the intermediate goal  $FIE_{G_1}$  behaves differently from other GRL intentional elements, it is represented by its own modeling construct called a *feature impact element*



(FIE). This is the second of the two new modeling constructs to support concern-oriented reuse in GRL models.

Several constraints exist for the elements in the sub-graph below a  $FIE_{Gn}$  goal and the reuse links in the GRL model.

1. Only one reuse link exists per reused feature or goal model element.
2. The source of a reuse link is a reused feature or reused goal model element.
3. In a GRL feature model, the target of a reuse link is a feature.
4. In a GRL goal model, the target of a reuse link is a feature impact element.
5. A reuse link in the GRL feature model is not related to another reuse link (see direction of  $\langle\langle related \rangle\rangle$  relationship in Figure 6.1).
6. A reuse link in the GRL goal model is related to exactly one reuse link in the GRL feature model (see direction of  $\langle\langle related \rangle\rangle$  relationship in Figure 6.1).
7. In a GRL goal model, a feature impact element  $FIE$  is the target of exactly one contribution from a feature  $F$ . If a reuse link  $RLG$  in the GRL goal model has  $FIE$  as its target, then  $F$  is the same feature as the feature of the reuse link  $RLF$  in the GRL feature model to which  $RLG$  is related. For example in Figure 6.1, the feature connected to the reuse links in the feature model and the one connected to the feature impact element is the same ( $F_2$ ), because the reuse links in both models are related to each other.

The sub-graph below the  $FIE_{G1}$  goal is visualized by the new feature impact element shown at the bottom of Figure 6.1, (i) because of these constraints, (ii) because the sub-graph requires considerable space on the diagram, (iii) because the links can be automatically

generated but the weights cannot, and (iv) because the whole subgraph is only active if the corresponding feature is selected.

The new feature impact element shows the one feature related to it in the first row and each reused goal model element in the other rows. The column to the right indicates the weights of the respective contribution/reuse links. The new feature impact element is semantically equivalent to the sub-graph with the  $FIE_{G_1}$  goal as its root element.

The new feature impact element modeling construct makes it possible to clearly define the reuse interface for GRL goal models in the context of concern-oriented reuse. The reuse interface contains all intentional elements of the goal model, which are not features or feature impact elements.

For example, the GRL goal model in Figure 6.1 only exposes goal  $G_1$  in its interface, while hiding all features, the feature impact element, and all reused model elements. This is appropriate, because a concern wanting to reuse this goal model should not be aware of the internal structure of the goal model and which parts of it have been reused from another concern, but rather should only be interested in the impact on the system qualities expressed by goal model elements at the top of the goal model.

Figure 6.2 presents the GRL metamodel changes required to realize the two new modeling constructs: **ReuseLink** and **FeatureImpactElement**. A **ReuseLink** is a new subclass of a **Contribution** link, which allows the **ReuseLink** to be treated as a mandatory link in a feature model and as a contribution link in a goal model. The directed reflexive association of **ReuseLink** (**reuseLinkInFM**) captures the relationship of reuse links on goal models and feature models (see constraints 5 and 6 listed earlier). To obtain the reused feature from a **ReuseLink** in a GRL goal model, the source element (**src**) of its **reusedLinkInFM** is retrieved, and the reusing feature is obtained from the target element (**dest**) of its **reusedLinkInFM**. Last but not least, a **FeatureImpactElement** is a new subclass of **IntentionalElement**, allowing it to be used as a node in the GRL goal model and analyzed by existing evaluation



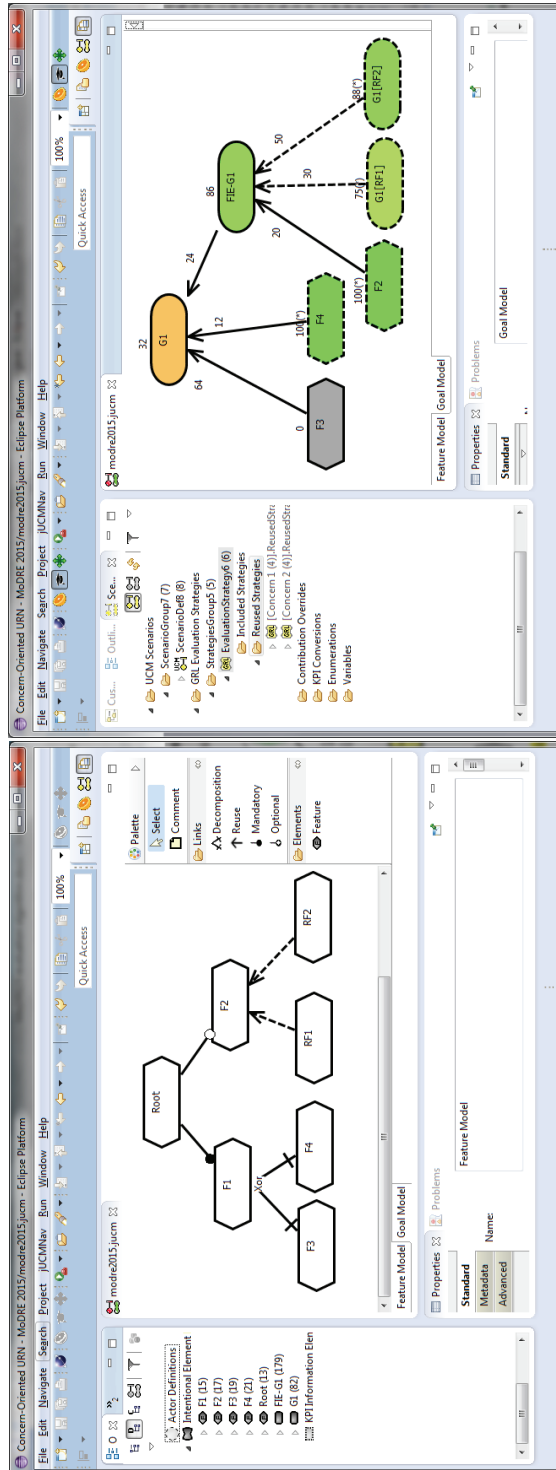


Figure 6.3: Feature Model (left) and evaluated Goal Model with reused strategies (right) in jUCMNav [29]

constructs with the suggested metamodel extensions described in this section address this problem and answer the question of what happens to the visibility of modeling elements of a reused goal model in the reusing model.

## 6.3 Goal Model Evaluation in the Reuse Hierarchy

As the results of the systematic literature review presented in Chapter 3 also point out, current evaluation mechanisms developed for goal models do not work in the context of reuse hierarchies mainly because the mechanisms often expect a monolithic goal model.

As explained previously, the goal model for a reusing concern is now modular, i.e., the goal model is split into several smaller goal models – one for the reusing concern and one for each reused concern. These goal models are connected by reuse links, which need to be taken into account by a new evaluation mechanism for reusable goal models in concern hierarchies. The new evaluation mechanism must consider whether a reuse link is active or not (i.e., whether the feature related to the reuse link is selected or not).

Consider the goal model in Figure 6.1. It can only be fully evaluated once the satisfaction values of the reused goal model elements ( $G_1[RF_1]$ ,  $G_1[RF_2]$ ) are known for each active reuse link. Consequently, the goal model of each active reused element needs to be evaluated first. The fact that a reused concern may itself also reuse other lower-level concerns leads to a recursive evaluation mechanism that first evaluates all the goal models of the reused concerns before evaluating the goal model of a reusing concern.

In Figure 6.2, URN strategies (`EvaluationStrategy`) already allow initial satisfaction values (`Evaluations`) to be specified for a single goal model (i.e., a concern without reuses). To also handle the specification of strategies for the reused goal models (`reusedStrategies`), the GRL metamodel is extended with the concept of `ReusedStrategy`. A `ReusedStrategy` is a subclass of the existing `EvaluationStrategy`, specifying in addition for which `reus-`

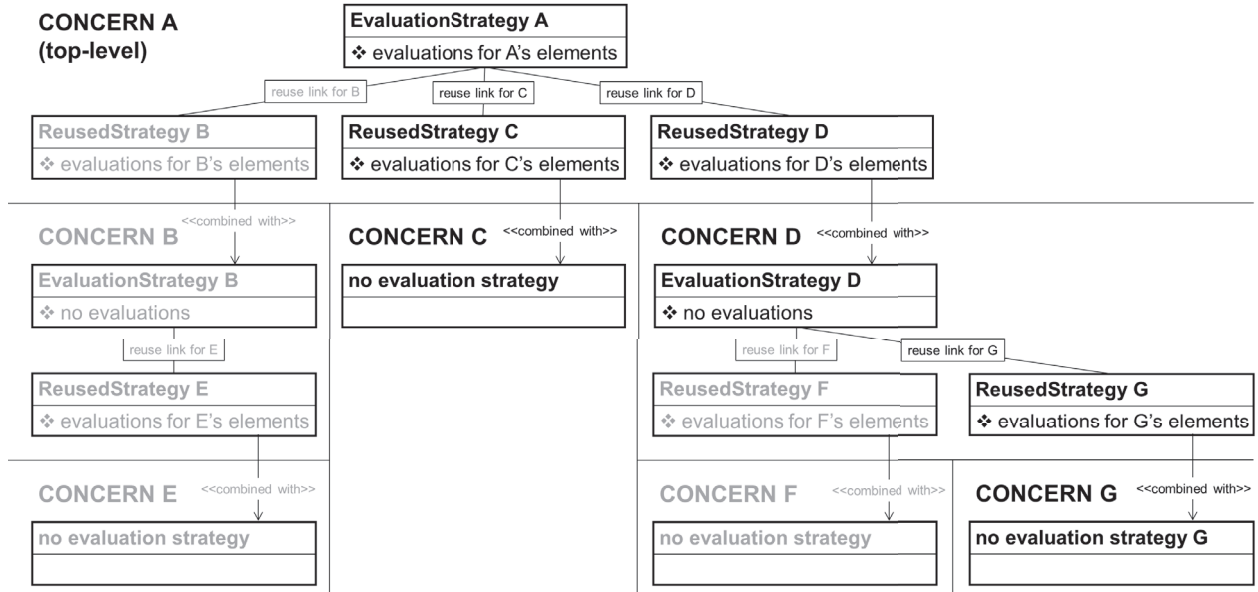


Figure 6.4: Evaluation of Goal Models in a concern hierarchy [29]

edLinkInFM the ReusedStrategy is defined. To obtain the reused concern of a ReusedStrategy, the concern of the source element (src) of its reuseLinkInFM is retrieved. A ReusedStrategy may also be ignored, if the reusedLinkInFM is not active (i.e., its feature (dest) is not selected).

Figure 6.4 illustrates the use of evaluation strategies and reused strategies for a concern hierarchy with seven concerns (A to G). The top-level concern A reuses three concerns (B, C, and D), i.e., a ReuseLink and a ReusedStrategy exist for each of these concerns.

The modeler of top-level concern A defines an EvaluationStrategy that (i) knows about the three reused strategies (reusedStrategies) and (ii) defines the initial satisfaction values (Evaluations) for A's feature and goal model elements.

For example, assuming that the goal model in Figure 6.1 is the top-level concern A, the evaluation strategy of concern A would specify evaluations for all feature and goal model elements except for reused elements and feature impact elements, i.e.,  $F_2$ ,  $F_3$ , and  $F_4$ . The

reused strategies associated with the evaluation strategy of concern A, on the other hand, would specify the evaluations of the feature and goal elements of the reused concerns identified by features  $RF_1$  and  $RF_2$ .

Given the evaluations in the evaluation strategy of concern A in Figure 6.4, some reuse links may be active and some may be inactive. To discuss these two cases, some reuse links are assumed to be inactive in Figure 6.4. Inactive reuse links and their corresponding concerns and strategies are shown in gray font in Figure 6.4 and do not need to be taken into account by the new evaluation mechanism.

An intermediate-level concern (B and D) also has an **EvaluationStrategy**, which also knows about the **ReusedStrategy** for each of the **ReuseLinks** in the intermediate-level concern (i.e., reuse links for E, F, and G). Such a **ReusedStrategy** is defined by the modeler of the intermediate-level concern and not by the modeler of the top-level concern, because the fact that an intermediate concern reuses another concern must be hidden from the modeler of the top-level concern in a proper reuse hierarchy.

The evaluation strategy of an intermediate-level concern does not define **Evaluations** for its feature and goal model elements, because the reusing concern decides on these evaluations. This is the case because only the reusing concern knows which features and goal model elements are relevant in the context of the reusing concern. Therefore, a **ReusedStrategy** specifies **Evaluations** for the reused concern (e.g., **ReusedStrategyB** in concern A defines evaluations for feature and goal model elements in concern B).

A bottom-level concern (C, E, F, and G), on the other hand, does not have an **EvaluationStrategy**, because a bottom-level concern does not reuse any other concern and the **Evaluations** for its feature and goal model element are defined by the reused strategy for the bottom-level concern in the concern reusing the bottom-level concern (e.g., the evaluations for concern C are defined by **ReusedStrategyC** in concern A).

The new evaluation mechanism hence starts at the top-level concern in the concern hier-

archy with evaluation strategy **A**. However, the goal model of concern **A** cannot be evaluated at this point, because it contains active reused elements from concerns **C** and **D**. Therefore, the new evaluation mechanism recursively evaluates the goal models of concerns **C** and **D** using the reused strategies for **C** and **D**, respectively.

The recursion continues as long as the next-lower concern has an active reuse for an even lower-level concern and stops when the next concern does not reuse another concern (e.g., concern **G**), i.e., the concern is at the lowest level in the concern hierarchy.

In this case, the evaluations from the reused strategy **G** in the next-higher concern **D** are applied and the goal model for concern **G** is fully evaluated, which in turn determines the satisfaction values of the reused elements from **G** in the next-higher concern **D**.

At this point, the satisfaction values of all reused elements in concern **D** have been determined, because the reuse of concern **F** is not active. The satisfaction values are therefore combined with the evaluations of the feature and goal model elements of concern **D**, which are defined by reused strategy **D** in the next-higher concern **A**.

Now, the goal model of concern **D** can be fully evaluated, which in turn determines the satisfaction values of the reused elements from **D** in concern **A**. Similarly, the satisfaction values of the reused elements from **C** are determined, and eventually, it is possible to fully evaluate the goal model of the top-level concern **A**, using the satisfaction values of the reused elements from concern **C** and **D** in concern **A** and **A**'s evaluation strategy.

Note that the evaluation of a goal model for which the satisfaction values of all reused elements have been determined may use any of the existing propagation-based evaluation mechanisms, because only a single goal model has to be evaluated at that point. Hence, the new recursive evaluation mechanism works with any of the existing evaluation mechanisms. In other words, the new recursive evaluation mechanism simply ensures that an existing evaluation mechanism is applied to the goal models of the concern in the concern hierarchy in the correct order, so that satisfaction values of reused model elements are determined



before the goal model in which they reside is evaluated.

A description of the new recursive evaluation mechanism for reusable goal models in reuse hierarchies is given in Algorithm 6.1 and shows how the existing evaluation mechanism (**evaluate**) described over the previous chapters is incorporated into the new evaluation mechanism.

Although out of scope for this evaluation mechanism, it is worth mentioning that the presented extensions to the GRL metamodel are capable of handling additional uses of evaluation strategies in concern hierarchies. For example, the concerns in Figure 6.4 which do not have an evaluation strategy or have an evaluation strategy that does not specify evaluations of its own feature/goal model elements (i.e., all concerns except the top-level one) could indeed specify evaluations as part of their evaluation strategies, which then could act as a default strategy for the concern. It is even possible to specify several default strategies, from which a reusing concern can choose.

A reused strategy could then decide to define its own evaluations as shown in Figure 6.4, or include a default strategy from the reused concern as is (**includedStrategies**), or include a default strategy and override or add evaluations. This capability would also allow a concern to define partial evaluation strategies with delayed decisions which are expected to be completed by the reusing concern when more contextual information is available. Furthermore, the extended metamodel also allows such evaluation strategies with delayed decisions to be extended not by the reusing concern immediately above the reused concern in the concern hierarchy, but by any reusing concern that is above the reused concern in the concern hierarchy, since missing contextual information may become available at any point in the reuse hierarchy.

Finally, it is also possible for one feature or concern to reuse the same concern twice, i.e., two reuse links and two reused strategies exist for the same concern. The new evaluation mechanism actually already takes this into account, by evaluating a separate instance of

**Algorithm 6.1** Evaluation Mechanism for Goal Models in Concern Hierarchies [29]

---

```

1  // called for the top-level concern
2  EvaluationResult evaluateTop(EvaluationStrategy e) {
3      // first evaluate all reused concerns
4      initial = evaluateReused(e)
5      // combine the results of the reused elements in the top-level concern with
6      // the initial satisfaction values (evaluations) of the top-level concern
7      initial.combine(e.getEvaluations())
8      // evaluate the top-level goal model based on its initial satisfaction values
9      return evaluate(initial)
10 }
11
12 // called recursively
13 EvaluationResult evaluateReused(EvaluationStrategy e) {
14     result = empty
15     // handle each reused concern of the reusing concern
16     // (only if reuse link is active)
17     for each rs in e.getReusedStrategies() {
18         if rs.getReuseLinkInFM.isActive() {
19             // get the evaluation strategy of the reused concern
20             es = rs.getReuseLinkInFM.getSrc().getConcern().getEvaluationStrategy()
21             // recursively evaluate all reused concerns of the reused concern
22             initial = evaluateReused(es)
23             // combine the results of the reused elements in the already reused
24             // concern with the initial satisfaction values (evaluations)
25             // in the reused strategy for the reused concern
26             initial.combine(rs.getEvaluations())
27             // evaluate the goal model of the reused concern based on its initial
28             // satisfaction values and combine the result with the results from
29             // other reused concerns at this level
30             result.combine(evaluate(initial))
31         }
32     }
33     return result
34 }
35
36 // existing evaluation mechanism
37 EvaluationResult evaluate(EvaluationResult er)
38 { ... }

```

---

a goal model for each reused strategy of a reused concern with the aim of determining a specific set of reused elements (those related to a specific reuse link/reuse strategy) in the reusing concern.

The screen shot from the jUCMNav tool [227] on the right side of Figure 6.3 shows the goal model in Figure 6.1 after evaluation with two reused strategies (shown to the left of the goal model) for the two reused concerns of the two reused goal model elements  $G_1[RF_1]$  and  $G_1[RF_2]$ . Again, there are slight differences for the visualizations of reuse links and reused goals.

## 6.4 Handling Delayed Decisions

In the context of Concern-Oriented, initial task satisfactions that are to be propagated by goal model evaluation are determined by the corresponding selections done on the feature model that contains features representing the mentioned tasks.

A feature can be *selected* (i.e., included in the configuration and is assigned to 100), *unselected* (i.e., excluded from the configuration and is assigned to 0) or *re-exposed* (i.e., decision has not yet been made on its selection) [23]. A task representing a feature is typically a leaf node in the goal model.

As discussed so far in the thesis, the evaluation of reusable goal models results in a single satisfaction value for each goal when the designer of the reusable artifact has a complete task selection in mind (i.e., all of the features are either selected or unselected). In return, these results can be used for a trade-off analysis among different goals to land on an optimal solution variant depending on the project's needs. However, due to the generic nature of reusable artifacts, designers may not always have sufficient information to decide on whether to select or unselect a feature.

When building reuse hierarchies from smaller reusable artifacts, we call the remaining

task selections *delayed decisions* because they are postponed to a higher level in the reuse hierarchy when more is known about the system being developed.

For trade-off analysis to be viable in the context of reuse, it is necessary to take these delayed decisions into account while conducting the goal model evaluation. The extended algorithm proposed in this section takes into account these delayed decisions and evaluates the range from best to worst possible result that can be obtained given the task selections that have been made in the entire reuse hierarchy.

The algorithm leverages the distinct levels in the reuse hierarchy to manage the computational complexity of the evaluation as it spans the entire reuse hierarchy.

At the higher levels of the reuse hierarchy, focus of the evaluation is kept on the decisions that have yet to be made instead of the decisions that have been made in the lower levels (i.e., task selections that cannot be changed). Consequently, when moving on to the next higher level artifact, feature and goal models can be pruned with the decisions that have already been made to deal with the computational complexity of the evaluation. In other words, the evaluation only takes the re-exposed features into account while evaluating the reused lower level goal models. Pruning has been discussed in the context of feature models [232], but not for goal models.

Contributions of this thesis for handling delayed decisions in goal model reuse hierarchies can be summarized into four main points, for which a proof-of-concept implementation is provided:

- A *range evaluation* mechanism to handle delayed decisions is introduced.
- For the evaluation to span the entire reuse hierarchy, *normalization of ranges* at the reuse boundaries is implemented.
- *Goal models are pruned* with the fixed decisions that have been made at the lower levels of the reuse hierarchy to present and guide the designers with only the decisions

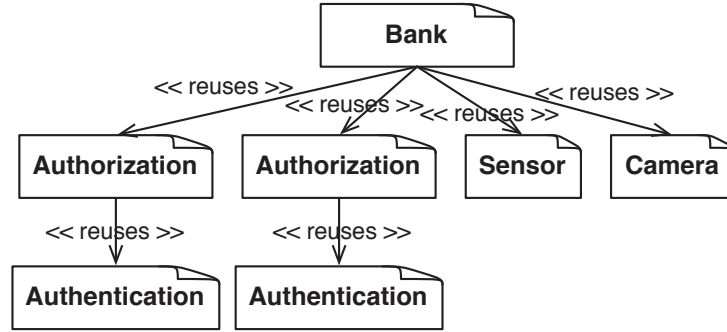


Figure 6.5: Concern hierarchy of the Bank application [30]

that can still be made.

- Taking advantage of the reuse boundaries between goal models at different levels in the hierarchy, *computational complexity of this reuse hierarchy-wide evaluation is managed*.

As discussed in Section 3.2.3 as the third research theme, none of the existing goal modeling approaches provided a complete mechanism to delay decisions in the reuse hierarchy. The range evaluation algorithm introduced in this section addresses this problem in the literature.

For the description of the extended goal model evaluation, a simplified example reuse hierarchy of a Bank Application is presented in the remainder of this section.

#### 6.4.1 Range Evaluation in Goal Model Reuse Hierarchies

A Bank Application is built using 5 different concerns (i.e., Bank, Authorization, Sensor, Camera, and Authentication) in a three level reuse hierarchy as shown in Figure 6.5.

The Bank Concern reuses Authorization twice with different configurations for different purposes (once for Internet Banking and once for ATMs) in addition to Sensor and Camera concerns (needed for the bank’s alarm system for its vault). Furthermore, the Authorization Concern reuses Authentication.

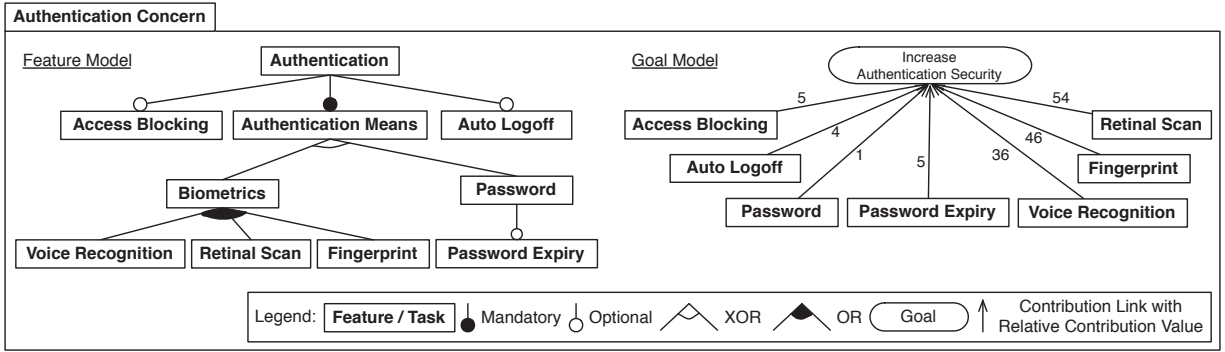


Figure 6.6: Variation interface of the Authentication concern [30]

The reuse scenario is described in this section starting from the lowest level with Authentication Concern. Without loss of generality, we focus on the Authentication and Authorization concerns in the remainder of this section to demonstrate the proposed algorithm.

### Level 1: Authentication Concern

In the reuse hierarchy of the example Bank Application, the Authentication Concern resides at the bottom and is reused by the Authorization Concern. The feature model of the Authentication is shown in Figure 6.6 side by side with one of the goals in the Authentication Concern's goal model.

As previously explained in Section 5.3, features in the feature model are represented as tasks in the goal model to capture their impacts on system goals and qualities.

Numerical values shown on the contribution links in the goal model represent the relative quantitative contribution values that are assigned locally by the designer of this reusable artifact. The designer, being the expert of her domain, ranks and weighs the solution alternatives (i.e., features) provided by the reusable artifact and assigns the relative contribution values. The normalization algorithm described in Chapter 5, then calculates the best and worst possible configurations that yield the maximum and minimum achievable satisfaction

values for each intentional element (i.e., goals) and finds the scaling factor and offset values that will be used in the goal model evaluation to make sure that the evaluation results always fall within the designated minimum/maximum range (i.e.,  $[0, 100]$ ).

For the Authentication Concern, the best possible configuration that maximizes the **Increase Authentication Security** goal is achieved by selecting **Retinal Scan**, **Fingerprint**, **Voice Recognition**, **Access Blocking**, and **Auto Logoff** (i.e.,  $54 + 46 + 36 + 5 + 4 = 145$ ). Features **Password** and **Password Expiry** cannot be in the same configuration as **Retinal Scan**, **Fingerprint**, and **Voice Recognition** due to the alternative (i.e., XOR) relationship between **Password** and **Biometrics** in the feature model of the Authentication.

On the other hand, the worst possible configuration that minimizes the **Increase Authentication Security** goal is achieved by selecting **Password** only (i.e., 1).

When the design of the Authentication Concern is complete, the normalization algorithm calculates the scale factor and offset values for all of its goals in the goal model and stores this information for the goal model evaluation that will be conducted when this concern is reused by another.

**Example 9.** In Figure 6.6, the satisfaction range of **Increase Authentication Security** is scaled from  $[1, 145]$  to the designated range of  $[0, 100]$  with a scaling factor of 0.69444 (i.e.,  $(100 - 0)/(145 - 1) = 0.69444$ ) and an offset of  $-0.69444$  (i.e.,  $((0 * 145) - (100 * 1))/(145 - 1) = -0.69444$ ).

Consequently, if the sum of the contributions is 1, the normalized satisfaction value is 0 ( $[1 * 0.69444] - 0.69444 = 0$ ), and if the sum is 145, then the normalized satisfaction value is 100 ( $[145 * 0.69444] - 0.69444 = 100$ ).

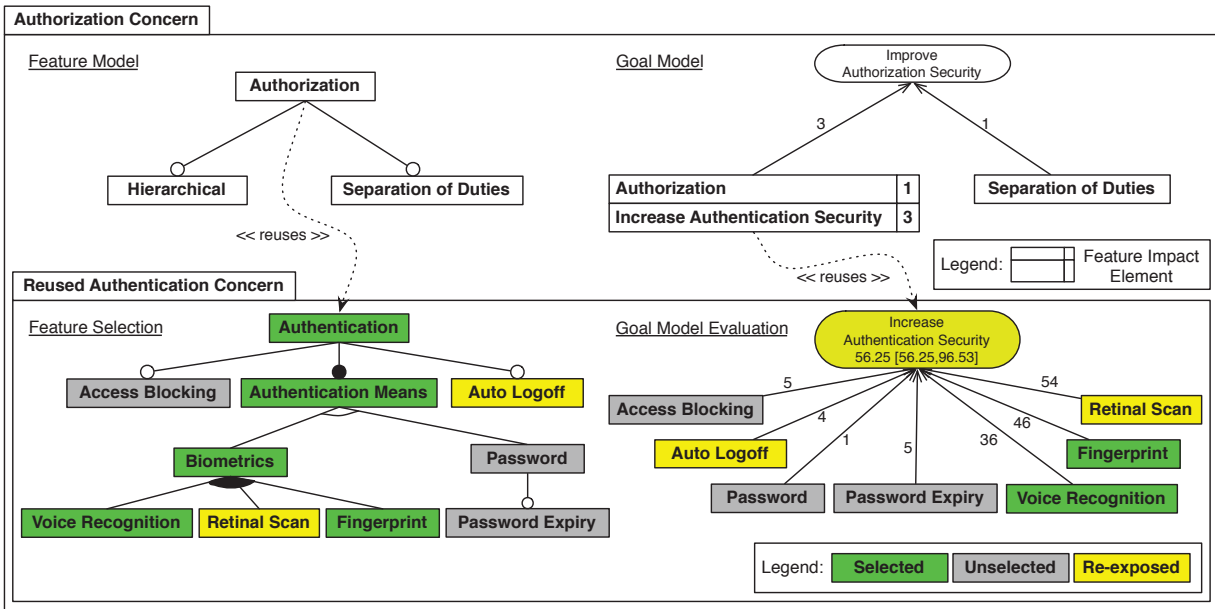


Figure 6.7: Authorization concern reusing Authentication [30]

## Level 2: Authorization Concern

In the second level of the reuse hierarchy, the Authorization Concern is built by reusing the Authentication Concern. As shown in Figure 6.7, the Authorization Concern contains two optional features, Hierarchical and Separation of Duties.

When the designer of the Authorization Concern decides to reuse the Authentication Concern, the root feature (i.e., Authorization) is connected with the current feature model. The designer makes a selection from the features of the reused concern. While doing the feature selection, the designer may select, unselect, or re-expose the features according to their needs for their own reusable artifact. Goal model evaluation is used to guide this feature selection with the help of the trade-off analysis among different goals and solution alternatives.

When the feature selection for the reused concern (i.e., Authentication Concern) is done,



a *reuse link* is formed between the reused feature (i.e., **Authentication**) and the reusing feature (i.e., **Authorization**). A reuse link in the feature model is similar to a mandatory link, because as long as the reusing feature is selected, the reused features must be selected, too.

The impact of the reused artifact on the **Improve Authorization Security** goal of the reusing artifact is represented with a *feature impact element* that constitutes the reuse interface for reusable goal models as described in Section 6.2. The feature impact element depicts the relationship between **Authorization** and the reused **Increase Authentication Security** in the goal model of the Authorization Concern as shown in Figure 6.7. The impacts defined internally for the feature impact element (i.e., 1 for **Authorization** and 3 for **Increase Authentication Security**) represent the locally assigned weights among the reusing feature and reused goals (i.e., how much the feature itself and its reused parts are individually contributing).

The contribution link from the feature impact element represents the combined impact of the feature and its reuses on the goal **Improve Authorization Security** relative to other impacts on the same goal. The goal model of the Authorization Concern shows that the designer of the Authorization Concern decided that **Hierarchical** does not have an impact on authorization security whereas **Separation of Duties** and the root feature (i.e., **Authorization**) do. The contributions of **Separation of Duties** and **Authorization** are 1 and 3, respectively.

Upon selecting **Voice Recognition**, **Fingerprint**, **Biometrics**, **Authentication Means** and **Authentication**, and unselecting **Access Blocking**, **Password**, and **Password Expiry**, the extended goal model evaluation algorithm assigns the initial satisfaction values of 0 and 100 for unselected and selected features, respectively, and then conducts the range evaluation considering the remaining selections due to the re-exposed features in addition to the evaluation of the current level of satisfaction.

**Example 10.** For the goal model evaluation example shown in Figure 6.7, the current satisfaction value for **Increase Authentication Security** is calculated as 56.25 via summation and scaling of the contributions from **Fingerprint** and **Voice Recognition** (i.e.,  $[(36 + 46) * 0.69444] - 0.69444 = 56.25$ ). The possible range of best and worst achievable satisfactions for the goal **Increase Authentication Security** is then calculated as  $[56.25, 96.53]$  considering the re-exposed features **Auto Logoff** and **Retinal Scan**.

The worst achievable value (i.e., 56.25) is equal to the current satisfaction value as unselecting both **Auto Logoff** and **Retinal Scan** would yield the lowest outcome.

The best achievable value at this point (i.e., 96.53) would be achieved with the selection of both **Auto Logoff** and **Retinal Scan** (i.e.,  $[(36 + 46 + 54 + 4) * 0.69444] - 0.69444 = 96.53$ ) and is lower than the best case value of 100 due to **Access Blocking** being unselected and excluded from the feature configuration.

The algorithm presented in Chapter 5 that determines the best and worst configurations to calculate scale factor and offset values is being reused for the evaluation of the possible ranges, this time to answer different questions with more information on feature selections. Instead of assuming each feature can be selected or unselected as in the case of scale factor and offset calculation, the range evaluation algorithm assigns the known fixed values (i.e., 100 if selected and 0 if unselected) to some of the features and finds out the possible ranges based on these selections.

To summarize, depending on the decisions that will be made at the later stages on whether or not to select or unselect the features **Auto Logoff** and **Retinal Scan**, the resultant satisfaction value for **Increase Authentication Security** will be anywhere within the range of  $[56.25, 96.53]$ .

The designer of the Authorization Concern makes a feature selection based on her needs for her own reusable artifact and delays some of the decisions. While making the feature

selection, although only one of the goals from the Authentication goal model is shown in this example, the designer considers all of the goals (e.g., **Increase User Convenience**, **Decrease Cost**, etc.) and the trade-offs among them.

### Level 3: The Bank Application

At the top level, the final application is being built as the Bank Concern. An important aspect to note here is that, with each next-higher level in the reuse hierarchy, the number of decisions that have been delayed in the lower levels either decreases or stays the same (i.e., the remaining decisions from lower levels cannot increase).

To illustrate on the example at hand, when the Bank Application reuses the Authorization Concern, the only decisions to be made for the lower level Authentication Concern are for the previously re-exposed features **Auto Logoff** and **Retinal Scan**. This information is important in two aspects.

First, the designer of the Bank Application does not need to be presented with the complete feature models of the lower level reused concerns as the decisions that have already been made in lower levels are not relevant for this artifact. The same applies to goal models. Hence, feature and goal models can be pruned to only cover the remaining decisions due to the re-exposed features. Furthermore, only the goals that are designated to have an impact on the reusing goal model (e.g., **Increase Authentication Security** from the reused Authentication Goal Model) are relevant to the designer of the higher-level artifact.

Consequently, only the goals that are used in the reuse interface (i.e., feature impact element) and their re-exposed children are kept visible at higher-levels. Any goal model element is pruned from the goal model if it does not have a range of satisfaction values given the already made decisions from lower levels in the reuse hierarchy.

As shown in Figure 6.8, features **Access Blocking**, **Voice Recognition**, **Fingerprint**,

**Password**, and **Password Expiry** are pruned from both the Authentication feature and goal models.

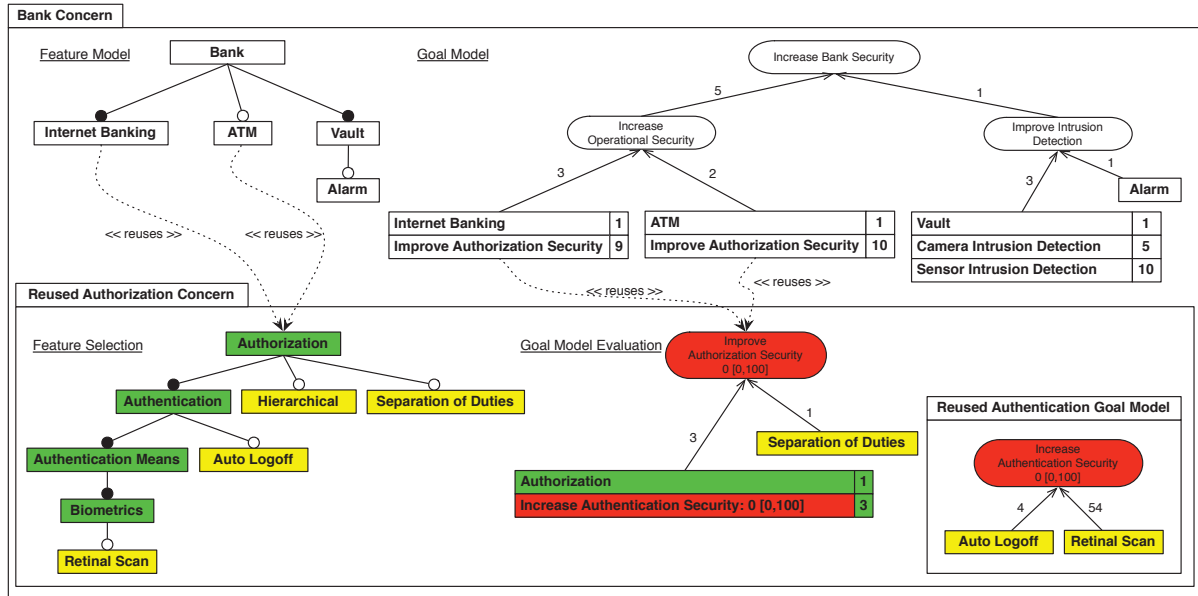
Second, the previously calculated range of  $[56.25, 96.53]$  for the **Increase Authentication Security** goal of Authentication needs to be re-visited now due to the fact that (i) the only decisions left to the Bank Application are whether or not to select the remaining two re-exposed features and that (ii) the Bank Application has no control over the decisions that have already been made by the Authorization Concern which in the end yielded the range of  $[56.25, 96.53]$ . The designer of the Bank Application only has a choice from this range. To be consistent with the philosophy of our approach for reusable goal models that always maps the worst choice to 0 and the best choice to 100, the range needs to be normalized.

Therefore, the evaluation range for the lower level goal **Increase Authentication Security** is normalized to  $[0, 100]$  from the previously calculated  $[56.25, 96.53]$  to properly guide the decision making for the remaining selections as shown in Figure 6.8a for the Reused Authentication Goal Model. With this additional normalization, the evaluation remains viable throughout the entire reuse hierarchy as it presents the designer consistent information for all decisions anywhere in the entire reuse hierarchy.

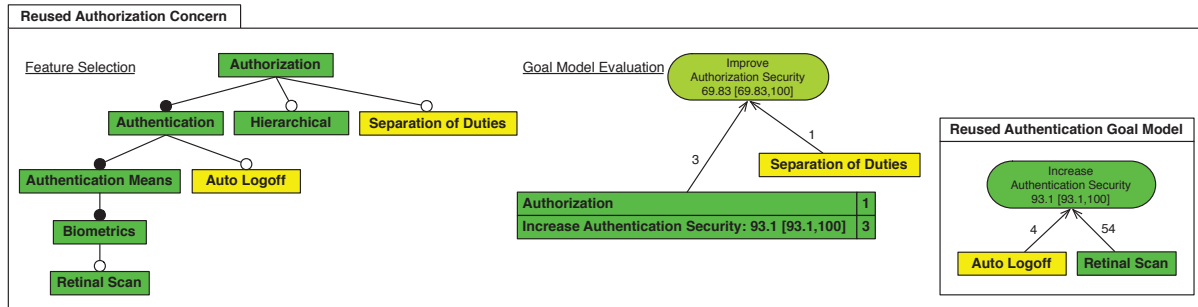
As explained in Figure 6.9, given the target goals, goal model, and the feature model, the range evaluation algorithm traces down to the goal model at the lowest level of the hierarchy (i.e., recursion through Step 1 to Step 2b) through the active feature impact elements (i.e., the reusing feature is selected) and evaluates each goal model for the current, minimum, and maximum possible satisfaction values for the desired goals (i.e., Step 2a to Step 5). The existing scale factor and offset calculation algorithm is called twice, once with the fixed initializations for previous selections (i.e., Step 3 in Figure 6.9), and once with the fixed initializations for current selections (i.e., Step 4 in Figure 6.9) to determine the on-the-fly scale factor and offsets for each goal for the normalization of the ranges of lower-level goals.

As previously described in Chapter 5, the existing normalization algorithm works on a

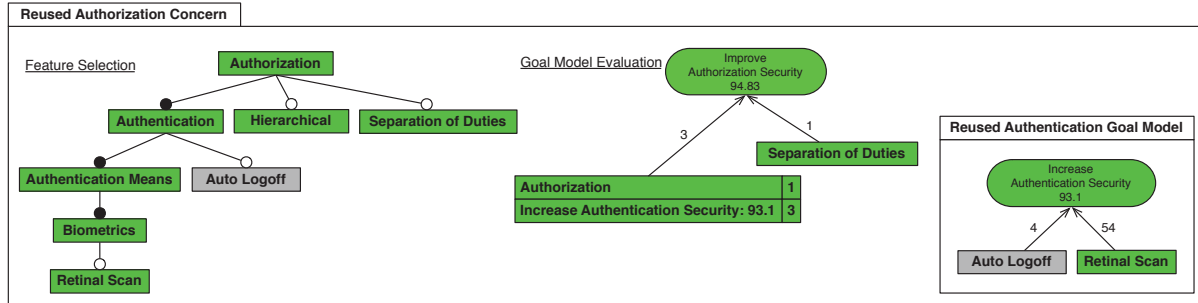
## 6.4. Handling Delayed Decisions



(a) Step I - Starting to make decisions on re-exposed features



(b) Step II - *Retinal Scan* and *Hierarchical* selected



(c) Step III - Complete selection

Figure 6.8: Bank concern reusing Authorization [30]

## 6.4. Handling Delayed Decisions

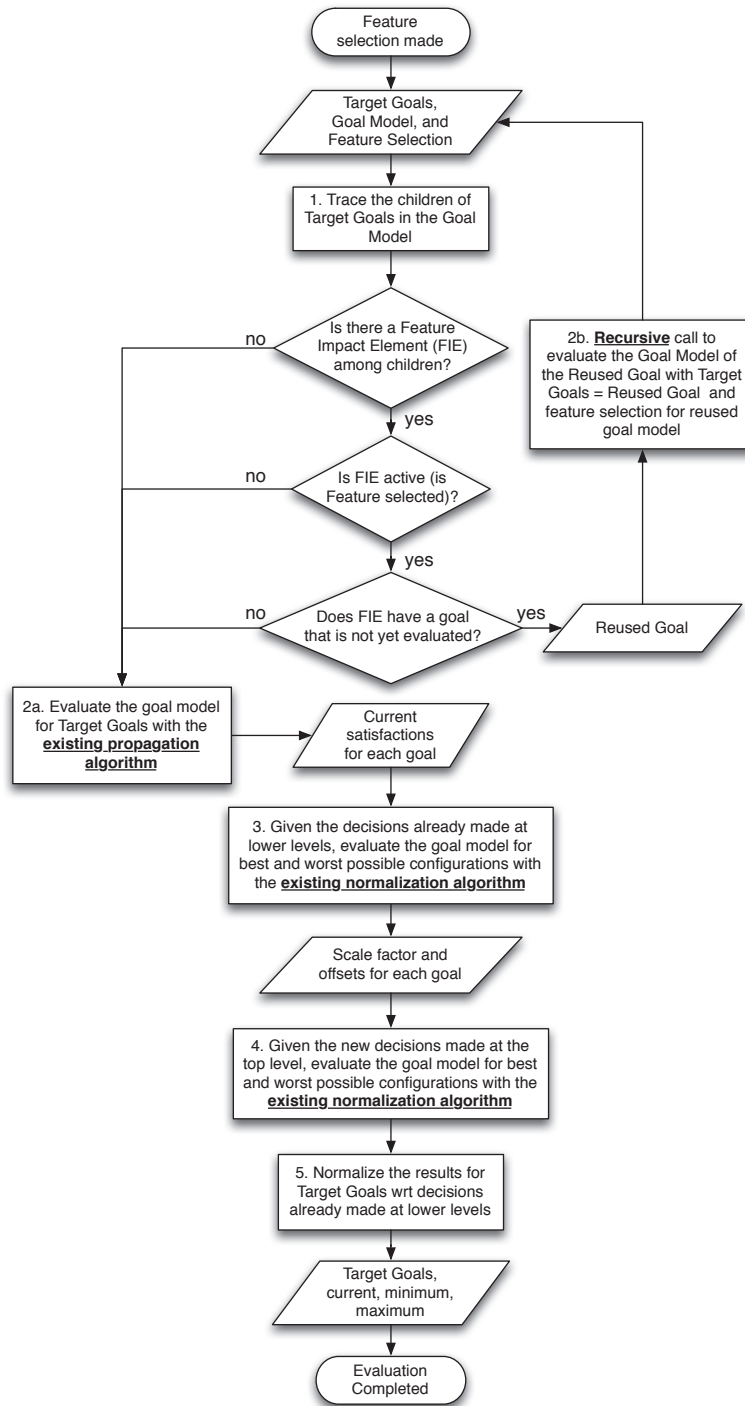


Figure 6.9: Range evaluation flowchart [30]

single level goal model and calculates and saves the scale factor and offset values for each intentional element in the model once the design of the goal model is complete. For a given goal model element, the algorithm asks all of its contributors recursively for their best/worst possible contribution values with the corresponding feature configurations and combines the returned configurations and contribution values. Only if there is a conflict in the combined configuration, the algorithm continues to ask for the next best/worst contributions from the contributors and goes on until it finds a contribution value that is yielded by a valid feature configuration. For the scale factor and offset calculation, each feature can only have two possible values, 100 when it is selected and 0 when it is unselected. When this normalization algorithm is reused for the range evaluation, we limit these possible values according to the decisions that have already been made and consequently run the same backtracking in a smaller solution space.

There are a number of reasons why this backtracking algorithm continues to operate at a single level for range evaluations and does not backtrack across the reuse boundaries. Given a feature selection, the backtracking algorithm goes through the goal model. When it reaches a feature impact element, there are two options. Either the feature corresponding to the feature impact element is selected or it is not selected. If it is not selected, then there is no need to consider the reused goals of the feature impact element. Therefore, the backtracking algorithm stops at the feature impact element. If the feature is selected, then the best/worst contribution of each reused goal is going to be 100/0, because the range at the reuse boundary was normalized. Furthermore, there is a need to ask for the next best/worst contribution, only if there is a conflict. However, there cannot be a conflict, because each reuse is exclusive to its reusing feature. Consequently, the backtracking algorithm stops at each reused goal after evaluating it once for its best/worst contribution.

Eliminating the need to go across reuse boundaries helps with computational complexity of this reuse hierarchy-wide evaluation as we do not need to consider a large monolithic goal

model for the calculation of scale factor and offset values for normalization during the design as well as the calculation of ranges and range normalization during the evaluation of reusable goal models.

Furthermore, when we consider the overall feature selection for the whole reuse hierarchy, while the number of features in the feature selection naturally increases with each level in the reuse hierarchy, the number of re-exposed features decreases. This means that the range evaluation works with a progressively smaller set of possible solutions which also helps manage the complexity.

To give a concrete example, consider the designer of the Bank Concern who decides to reuse Authorization. A three-step incremental feature selection scenario is depicted in Figure 6.8.

The first step shown in Figure 6.8a starts with the designer's decision to reuse the Authorization Concern. The designer is presented with the base feature selection for the concern being reused (i.e., Authorization) in addition to the re-exposed features from lower levels (i.e., Authentication). Goal model evaluation with the now normalized range values guides this reuse process with its support for trade-off analysis. Note that in this example, the designer chooses the same features for each of the two reuses of the Authorization Concern. However, the two reuses are independent from each other and the designer could have selected different features for each of the reuses.

The designer of the Bank Concern proceeds with the feature selection in the second step shown in Figure 6.8b. The designer selects **Hierarchical** from the Authorization Concern and **Retinal Scan** from the Authentication Concern at the lower level. At this point, the goal **Improve Authorization Security** evaluates to 69.83 and its range evaluation shows that with the remaining decisions, it will end up somewhere within the range [69.83, 100].

In the third and last step, shown in Figure 6.8c, the designer makes a complete selection by selecting **Separation of Duties** from the Authorization Concern and unselecting **Auto**



**Logoff** from the Authentication Concern. At this point, since there are no more decisions to be made (i.e., no re-exposed features left), the goal **Improve Authorization Security** evaluates to the fixed value of 94.83. Similarly, it can be seen that the goal **Increase Authentication Security** evaluates to the fixed value of 93.1 at the lower level.

In the end, the Bank Concern and its goal model can be used for trade-off analysis on the potential features of the Bank to select a product variant and deploy the final application.

### 6.4.2 Performance Discussion and Tool Support

The extended evaluation algorithm for reusable goal models switches the typical single satisfaction value evaluation to a range of values that are still possible based on the current task selections.

The algorithm that is used for the evaluation and normalization of ranges follows the same logic as the one used for the calculation of scale factor and offset values. In other words, the algorithm to find out best and worst configurations that yield the highest and lowest satisfaction values for the goals is being reused to answer different questions.

The prior algorithm calculates the scale factor and offset values for intentional elements once the design of the goal model is complete and within a solution space where all the possible feature configurations are included. In the case of the range evaluation, however, some decisions have already been made for some features. Consequently, for the range evaluation, the amount of time it would take to find the best and worst possible configurations is always upper-bound by the amount of time it takes the original algorithm to calculate scale factor and offset values, which was shown to have reasonable performance results in Section 5.5. Now that the same algorithm is being used multiple times in the evaluation, the impact of the added functionalities on the existing evaluation algorithm is at the most a multiple of the prior one and, hence, is linear.

Tool support for this extended goal model evaluation as presented in this work as well as full editing capabilities for feature and impact models is also provided in the concern-oriented software design modeling tool TouchCORE [27].

## 6.5 Top-down Evaluation of Reusable Goal Models

In the lower and intermediate levels of reuse hierarchies, reusable goal models represent the local impacts of alternative solutions provided by a reusable artifact on different qualities and non-functional requirements.

When an application is built at the top of the reuse hierarchy with the help of goal modeling, trade-off analysis helps decision making among different reusable solutions that are available to build the application. To drive the decision making process, it is necessary to provide guidelines for questions such as “*Which goals are more important for the stakeholders of the application?*” or “*How much of a goal’s satisfaction can we sacrifice to improve on another goal’s satisfaction?*”.

In the remainder of this section, our proposed method for goal prioritization in support of trade-off analysis at the application level is introduced, followed by the description of how top-down evaluation is achieved with our novel algorithm, both for a single-level goal model (in Section 6.5.1 and 6.5.2) and through the entire hierarchy of reusable goal models (in Section 6.5.3).

### 6.5.1 Goal Prioritization with Thresholds

First, let us consider prioritization of goals for decision making via trade-off analysis in goal models. When the application consists of several top-level goals, it is necessary to determine what these goals mean for the application. For this purpose, our approach uses percentage

*importance* values that are assigned by the application designers at the top level in the reuse hierarchy to express the priority of a goal.

Application designers may introduce an *actor* into the goal model and assign importance values to top-level goals to represent the overall prioritization of goals. While we discuss one actor within the scope of this thesis, the algorithm can be trivially extended to multiple actors by adding another layer of importance links from actors to the system [233].

For some goals such as cost and performance, *importance* values may be based on quantified, real life measurements. However, for the cases when those measurements are not available, and for non-functional requirements (i.e., *softgoals*) such as user convenience, security, and privacy, for which it is hard to define measurement units, application designers shall assign importance values through discussions with stakeholders as well as using their expertise. Using percentage importance values for goal prioritization allows us to address both cases and figure out the best solutions for an application using goal models.

Having decided on how the satisfaction of some high-level goals may influence the satisfaction of their stakeholders, designers can, in turn, limit the variability among the possible solutions to a desired, smaller set of solutions that complies with those decisions. In other words, with the help of goal models and trade-off analysis, designers can decide on the most suitable set of solutions (i.e., *tasks*) to be included in the application based on their impacts on the high-level goals that are prioritized by stakeholders.

In addition to the prioritization through importance values, our approach takes into account *threshold* values that may be assigned by application designers. Setting a threshold on a goal complements the goal prioritization with importances and allows application designers to specify their desired worst-case satisfaction value for that goal so that the algorithm can work to find the solutions sets that do not have the goal's satisfaction drop below the threshold.

### 6.5.2 The Lazy Recursive Algorithm in Top-down Evaluation

As explained in Section 5.2.1, the lazy recursive algorithm is initially designed to determine the possible minimal and maximal satisfaction values for goals and to use those values in the normalization of goal satisfactions in reusable goal models with constraints.

The lazy recursive algorithm avoids combinatorial explosion as it only takes a possible solution set into consideration when necessary (i.e., only when a constraint is violated by a solution set, the next best solution sets are evaluated). Even though in the worst case (i.e., where all possible solution sets violate a constraint)  $2^n$  combinations have to be examined, where  $n$  stands for the number of solutions (i.e., typically *tasks* in the goal model), the actual calculation time is reduced significantly.

Consequently, we propose to reuse and adapt this lazy recursive algorithm to handle additional constraints that may be introduced by stakeholders (i.e., goal importances and threshold satisfactions as explained in Section 6.5.1) and conduct the top-down evaluation.

An *actor* (i.e., stakeholder) is represented as a hidden top-level goal (i.e., dashed ellipse at the top of Figure 6.10) with incoming contribution links from the goals for which an importance value is defined.

An *importance value* that is defined for an actor-goal tuple is represented as the contribution value for its corresponding contribution link (i.e., values labeled “Importance” on dashed arrows at the top of Figure 6.10).

Finally, a *threshold value* is defined for a top-level goal to represent the desired worst-case satisfaction value for that goal (i.e., values labeled “Threshold” on top-level goals in Figure 6.10).

The desired result of the top-down evaluation is one of two possible outcomes:

- Find all combinations of solutions that do not violate any constraints, including the defined threshold values, and that correspond to the same, best possible evaluation

result for the actor.

- If it is not possible to satisfy all of the thresholds at the same time, suggest all solution combinations that violate the least amount of threshold constraints and that correspond to the best possible evaluation result for the actor.

To calculate the maximal solution for a goal, the algorithm *(i)* asks the goal's children (i.e., nodes that have outgoing contribution links with positive/negative contribution values to the goal) for the next highest/lowest solution combinations they can provide, *(ii)* calculates the potential satisfaction values for received combinations, *(iii)* checks the combined solution set for constraint violations, and if there are constraint violations, repeats from step *(i)*, or else, *(iv)* records this solution set and the best result achieved for it and repeats steps *(i)* to *(iv)* until the calculated satisfaction value drops below the recorded best result.

An explanatory model depicting the top-down evaluation process is shown in Figure 6.10. To keep readers' focus on the description of the algorithm iterations, the model size is kept minimal. The goal model contains two goals (i.e., **Goal X** and **Goal Y**) and 5 tasks (i.e., **Solution1-5**) that contribute to them.

Since this goal model is reusable, contribution values are defined relatively and goal satisfaction values must be normalized by scaling factor and offset values calculated for each goal as explained in Chapter 4.

For the sake of simplicity, no external constraints are defined among the five tasks. However, as explained in Chapter 5, it is possible to introduce external constraints for the tasks when using them as features in a feature model. In such scenario, two or more solutions may be conflicting due to numerous reasons (e.g., being XOR children of the same parent feature, having an *excludes* constraint between them, or having a run-time conflict imposed by a workflow model). Those potential conflicts are handled in *Step III* of the algorithm with the constraint checker.

## 6.5. Top-down Evaluation of Reusable Goal Models

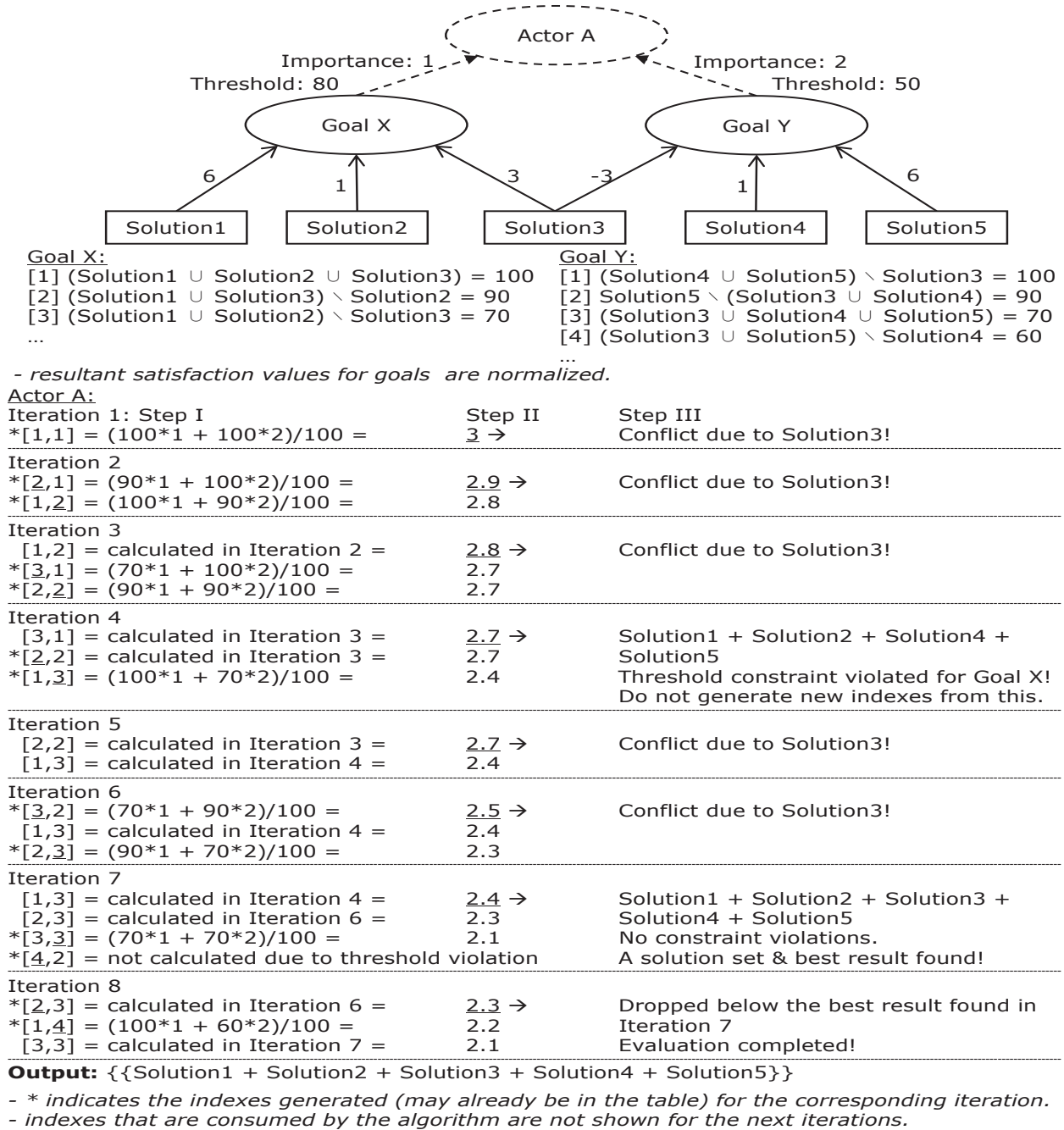


Figure 6.10: Top-down evaluation on a single-level reusable Goal Model [32]

When the application designer decides to find the optimal result for the goal model, they define an actor (e.g., **Actor A**) and set the importance and threshold values for high-level goals. The designer may also choose to include or exclude some solutions (i.e., tasks) from the result as a precondition to the top-down evaluation (e.g., select or unselect a feature that represents a task in the goal model to be in the feature configuration). In that case, the decisions made by the designer as preconditions are reflected on the initializations of the tasks. Setting preconditions reduces the size of the possible solution set for a task from 2 (i.e., task will be either included or excluded in the result) to 1. In Figure 6.10, to not oversimplify the example, no preconditions are set on **Solution1-5**.

The top-down evaluation is done per root element in the goal model (i.e., an actor) with the importance values defined for them. The example in Figure 6.10 illustrates that the algorithm begins with **Actor A**, and examines combination  $[1, 1]$  in *Step I* of the first iteration (i.e., the combination with the highest satisfaction value of each of **Actor A**'s contributors (**Solution1**, **Solution2**, **Solution3** for **Goal X**, and **Solution4** and **Solution5** excluding **Solution3** for **Goal Y**)). The potential actor satisfaction for this combination is calculated as 3 in *Step II* of the same iteration, but because the solution set of **Goal Y** excludes **Solution3**, the constraint check fails in *Step III* and this result is considered invalid.

Therefore, the algorithm moves on to the second iteration to examine the next best combinations where the index of each child is incremented individually. Having classified  $[1, 1]$  as invalid in the previous iteration, the combinations  $[2, 1]$  and  $[1, 2]$  are taken into consideration for the next (*Step I* of the second iteration), because any other remaining combination cannot result in a higher satisfaction value. This results in the satisfaction values of 2.9 and 2.8, respectively, in *Step II* of the second iteration, indicating that the combination  $[2, 1]$  is the next best option. While the algorithm takes the combination  $[2, 1]$  into account (i.e., consumes it in *Step III*), it still keeps the combination  $[1, 2]$  in the table. However, the same conflict due to *Solution3* reappears for combination  $[2, 1]$  and hence it is

again found to be invalid.

In the third iteration, the next best combinations are again brought to the table (i.e., since  $[2,1]$  is found invalid in the previous iteration,  $[3,1]$  and  $[2,2]$  are added). The next highest valued combination that has not been examined in the table is  $[1,2]$ . Up until *Step III* of the fourth iteration, the algorithm behaves the same way as it does for its previous application to find the maximal solution to calculate the scale factor and offset values for normalization as described in Section 5.2.1.

Upon the evaluation of the combination  $[1,2]$  at the last step of the third iteration, the algorithm adds  $[2,2]$  and  $[1,3]$  with their corresponding values 2.7 and 2.4, respectively, to the table. At this point, the already calculated (in third iteration)  $[3,1]$  is now at the top of the table and is examined in *Step III*. The constraint check finds out that, while this combination satisfies all other constraints, the threshold constraint for **Goal X** is violated (i.e., the satisfaction of **Goal X** for its third best combination yields 70, whereas its threshold is 80).

Considering a single level goal model only, the change to the existing lazy recursive algorithm for its adaptation to top-down evaluation is that it does not generate new indexes from those that violate a threshold constraint for a goal (e.g., at *Iteration 5* in Figure 6.10,  $[4,1]$  and  $[3,2]$  are not generated), because all remaining indexes also violate the threshold. Furthermore, the goal whose index caused the threshold violation (e.g., at *Iteration 4*, **Goal X** drops below its threshold for index value 3) is not asked for any index values greater than that (e.g., at *Iteration 7* in Figure 6.10, the combination for index  $[4,2]$  is not calculated).

The algorithm keeps going through the next combinations (i.e., *Iterations 5-7*) and at *Step III* of the seventh iteration, it discovers a solution set with no constraint violations (i.e., **Solution1**, **Solution2**, **Solution3**, **Solution4**, and **Solution5** together results in an evaluation of 2.4 for **Actor A**). At this point, it is important to note that there can still be other valid solution sets with the same evaluation result for the actor (i.e., 2.4). As described



in the first possible outcome of the top-down evaluation, it is desired to find not one but all of the possible solutions that do not violate any constraints and comply with the importances set by the application designer. Therefore, after finding a best case solution, the algorithm still continues until the result examined in *Step III* drops below the best result that was previously found. As the next best evaluation result of 2.3 for the combination [2,3] in *Step III* of *Iteration 8* drops below the previously found best result of 2.4, the output of the top-down evaluation in Figure 6.10 is a single solution: **Solution1**, **Solution2**, **Solution3**, **Solution4**, and **Solution5**.

Note that the first index that causes the evaluation to drop below a threshold is still kept in the table (e.g., 3 for **Goal X**) and only the next index is not kept (e.g., 4 for **Goal X**). The reasoning behind this is being able to provide the application designer the second possible outcome of the top-down evaluation. If the algorithm finds out that there are no possible combinations to satisfy all of the thresholds at the same time, it must find and suggest solution combinations with the highest evaluation result that violate the least amount of threshold constraints. If this is the case, the algorithm goes through the table starting from the top, looks for and returns all combinations with the lowest number of threshold violations, no other constraint violations, and the highest evaluation.

As explained in Chapter 4, use of relative values and normalization implies that an evaluation result of 100 for a goal indicates the best possible solution for that specific goal, whereas an evaluation result of 0 indicates the worst. To apply a threshold, the designer must understand what is sufficient for the application to be built, e.g., through reference selection or benchmarking. At the application level, the designer can focus on one quality area and do a reference selection on the goal model that yields a specific evaluation result for a goal that corresponds to the quality in focus. Later in the top-down evaluation, they can use that reference result as a threshold for that particular goal to enforce the algorithm to find the solution sets that do not yield a value worse than the reference selection.

Alternatively, if the application designers are able to do testing or benchmarking with the potential solutions and relate the results of these activities to a more measurable scale (i.e., goal satisfactions), those results can also be used to define the threshold values for their respective goals.

### 6.5.3 Top-down Evaluation in the Reuse Hierarchy

When reusable goal models are used to build a hierarchy of reusable artifacts, new challenges arise for the top-down evaluation.

In addition to the use of relative contribution values with normalization, we introduced two new modeling constructs (i.e., *reuse link* and *feature impact element*) that allows us to clearly define the reuse interface for goal models in Section 6.2. In this context, goal models and feature models are used in collaboration and tasks in the goal model represent features in the feature model. For its realization, a feature may reuse other artifacts that also make use of reusable goal models and the impacts of reused goals are reflected on the goal model at the higher level through the feature impact elements. Each reused goal in a feature impact element is assigned a weight along with the feature itself (e.g., reused goal **Authorization – Increase Performance** is assigned a weight of 4 whereas the feature **Transfer Funds** is assigned a weight of 1 in the **Bank Goal Model** in Figure 6.11). The feature impact element is evaluated internally based on (i) the satisfaction values of its reused goals, and (ii) whether or not the feature itself has been selected. The reuse link only becomes active if the feature is selected and included in the solution. Hence, the feature impact element’s own satisfaction is evaluated only if the feature is selected and is set to 0 otherwise.

The reuse boundary introduced by the feature impact element enables the evaluation of individual levels and prevents backtracking across levels since the normalization algorithm knows that each reused goal will evaluate to 100 for its best-case solution and 0 for its

worst, regardless of what their corresponding solution sets are (i.e., it is not possible to have conflicts across levels as constraints are defined internally for the models at the same level).

A three-level reuse hierarchy built with the help of reuse links and feature impact elements is illustrated in Figure 6.11.

At the top level (i.e., **Bank Goal Model**), two feature impact elements (i.e., **Transfer Funds, Authorization - Increase Performance** and **Transfer Funds, Authorization - Decrease Cost**) represent that the feature **Transfer Funds** reuses the **Authorization** artifact and separately reflect the impacts of **Increase Performance** and **Decrease Cost** goals of the **Authorization** on the **Bank Goal Model**.

Similarly, at the intermediate level (i.e., **Authorization Goal Model**), the feature **Authorization** reuses the **Authentication** artifact and separately reflects the impacts of **Improve Performance** and **Reduce Cost** goals of the **Authentication** on the **Authorization Goal Model**.

At the top level of the reuse hierarchy, the application designer introduces the actor (i.e., **Bank CTO**) and sets the importance and threshold values for the two high-level goals (i.e., **Increase Performance** and **Decrease Cost**) to conduct the top-down evaluation.

The main difference in running the top-down evaluation in a reuse hierarchy instead of a single level goal model is that it is possible to have conflicting solution sets for different reused goals of the same reused goal model. Consequently, when the algorithm asks a feature impact element for its next best combinations, it needs to go into the lower level reused goal model and find the possible solution sets there. Furthermore, when it is time to evaluate a combination (i.e., *Step III* of an iteration in Figure 6.10) for the actor (i.e., **Bank CTO** in Figure 6.11), the algorithm also needs to combine the solutions provided by those reused goals (i.e., **Increase Performance** and **Decrease Cost** goals of the **Authorization**) and check for the constraint violations in the lower level artifact (i.e., feature model of the **Authorization** artifact).



The trivial way to handle the reuses of a goal model is to disregard the goal model reuse hierarchy and treat it as one big flattened goal model to run the top-down evaluation on it. However, the computational complexity of the top-down evaluation increases as the goal model grows bigger. A better approach considers the reuse boundaries between reusable goal models that are already defined, taking advantage of this modular structure to reduce computational complexity.

For this purpose, we introduce a pre-processing algorithm that creates a surrogate actor for the lower level (i.e., reused) goal model and derives the importance values between reused goals and this surrogate actor. Derivation of the importance values for the surrogate actor is done via calculating the *comparative self-impact factors* of the reused goals (i.e., **Increase Performance** and **Decrease Cost** goals of the **Authorization**) on the actual actor in the goal model doing the reuse (i.e., **Bank CT0** in Figure 6.11). The derivation starts at the feature impact elements that represent the impact of the reused goal (i.e., **Transfer Funds**) and the calculated self-impact factor of the reused goal on the feature impact element is propagated upwards through the contribution links until the self-impact factor of the reused goal on the actor is obtained.

The self-impact factor calculation is illustrated in Figure 6.11 and described in pseudo code in Algorithm 6.2. First, the derived importance values for **Surrogate** of **Bank CT0** is calculated.

In the first step (i.e., *Step i1*), the self-impact factor of the reused goal **Increase Performance** on the feature impact model element is calculated as 0.8 since the possible range (max - min) for the feature impact element is 5 and the weight of the reused goal is 4 (i.e.,  $Sif_{AIPi1} = 4/5 = 0.8$ ).

In the second step (i.e., *Step i2*), the self-impact factor of the feature impact model element on the goal **Decrease Time for Authorization** is calculated as 0.6 since the possible range (max - min) for the goal is 4 and the contribution of the feature impact element is 3

(i.e.,  $Sif_{AIPi2} = 3/4 * 0.8 = 0.6$ ).

Similarly, in the third step (i.e., *Step i3*), the self-impact factor of the goal **Decrease Time for Authorization on Increase Performance** is calculated as 0.2 (i.e.,  $Sif_{AIPi3} = 2/6 * 0.6 = 0.2$ ) and propagated to the actor (i.e., **Bank CTO**) as 0.2 (i.e.,  $Sif_{AIPi4} = 0.2 * 1 = 0.2$ ) in *Step i4*.

When the same process is repeated for the reused goal **Decrease Cost** (shown in steps j1-4 in Figure 6.11), its self-impact factor is calculated as 0.18.

Using the derived comparative importance values (0.2 and 0.18) in the reused **Authorization Goal Model**, the same algorithm then calculates the derived importance values (i.e., 0.12 and 0.15 for goals **Improve Performance** and **Reduce Cost**, respectively) for **Surrogate of Surrogate of Bank CTO** at the bottom level (shown in steps i'1-3 and j'1-3 in Figure 6.11).

Derived comparative importance values of 0.12 and 0.15 for goals **Improve Performance** and **Reduce Cost** in **Authentication Goal Model** imply that regardless of the initialization of the features in the reuse hierarchy, the two example cases shown in Figure 6.11 (i.e., when the satisfaction values of **Improve Performance** and **Reduce Cost** are 100 and 80, respectively, and when the satisfaction values are 75 and 100, respectively) would yield the same satisfaction result for the top actor (i.e., **Bank CTO**) when everything else is kept the same.

The generalized formula to calculate the self-impact factor of a reused goal for the  $i^{th}$  step of forward propagation is as follows:

$$Sif_{RGi} = \frac{C_i}{max_{Gi} - min_{Gi}} * Sif_{RG(i-1)} \quad (6.1)$$

where  $Sif_{RG(i-1)}$  is the self-impact factor calculated for the reused goal in the previous

step,  $C_i$  is the relative contribution value of the link that the self-impact value is being propagated through, and  $max_{G_i}$  and  $min_{G_i}$  are the maximum and minimum achievable values of the goal that the self-impact value is being reflected upon. The values  $max_{G_i}$  and  $min_{G_i}$  do not need to be re-calculated as the range can be deduced from the scale factor (i.e.,  $max_{G_i} - min_{G_i} = 100/ScaleFactor$ ), which is already calculated and saved for each goal in the model when it is created.

At the initial step (i.e., for the feature impact element), the following formula is used:

$$Sif_{RG0} = \frac{W_{RG}}{max_{FIE} - min_{FIE}} \quad (6.2)$$

where  $Sif_{RG0}$  is the self-impact factor of the reused goal on the feature impact element,  $W_{RG}$  is the weight of the reused goal, and  $max_{FIE}$  and  $min_{FIE}$  are the internal maximum and minimum achievable values of the feature impact element.

At the last step of the self-impact factor derivation, the importance value replaces  $C_i$  and the final self-impact is calculated via multiplication of importance and the self-impact factor calculated for the reused goal on the goal for which the importance value is defined.

Calculating the derived importance values for surrogate actors allows the use of the same lazy recursive top-down evaluation algorithm on the reused goal model. The top-down evaluation starts at the application level, finds all reused goal models and if different goals from the same reused goal model are used in the reusing goal model, derives the importance values for the surrogate and moves on to the evaluation of the reused goal model. This process recursively repeats until a goal model with no reuses (i.e., no active feature impact elements) is found. Then, the top-down evaluation is executed at the bottom level and the algorithm starts going back up while collecting the valid, best case solution sets at each lower level goal model.

At higher levels, to calculate the best case solution sets, the algorithm uses the resultant

satisfaction values for the reused goals that are already calculated during the recursion. At the application level, when the top-down evaluation is completed, all the solution sets (i.e., the solution sets for both the lower level models and the top-level goal model) are combined and presented to the application designer. At this point it is important to note that it is not possible to have a conflict across different levels of the reuse hierarchy as the feature selections are independent (i.e., each goal model has their own, distinct feature model that is used by the conflict checker).

Consequently, propagating the importance values down and recursively using the same top-down evaluation algorithm for each level individually allows the top-down evaluation algorithm to benefit from the modularity granted by the reuse boundaries and prevents backtracking through the entire goal model reuse hierarchy for this evaluation. However, threshold values defined at the top level are not reflected in the evaluation of lower level goal models. As a consequence, a solution set from a reused goal may violate threshold constraints. Hence, this algorithm may determine an approximate solution with the best overall actor satisfaction but violated threshold constraints. It does not attempt to find other solutions with lower overall actor satisfactions that satisfy the thresholds.

A major issue with the evaluation of reusable goal models is the increased computational complexity with increasing model size. Rather than running the top-down evaluation on a large, flattened goal model in a sizable reuse hierarchy, our novel approach takes advantage of the reuse boundaries and individually evaluates reusable goal models of smaller size. This makes the top down evaluation feasible as is demonstrated in the next section.

Consequently, with both bottom-up and top-down evaluation, our approach fully satisfies the first requirement (R.1) of providing support for trade-off reasoning through the reuse hierarchy (see Section 1.2) as it allows the goal model evaluation to span the entire reuse hierarchy by incrementally evaluating levels in the reuse hierarchy.



**Algorithm 6.2** Surrogate Importance Calculation Algorithm

---

```

1  Map<Goal, ImportanceValue> calculateSurrogateImportances(GoalModel gm, Set<Goal>
2      reusedGoals, Map<Goal, ImportanceValue> importanceValues) {
3      Map<Goal, ImportanceValue> surrogateImportances;
4      Map<Goal, SelfImpactFactor> evaluations;
5      // For each reused goal in the GM, find the derived surrogate importance.
6      for (aReusedGoal : reusedGoals) {
7          Map<Goal, ContributorsToEvaluate> evaluationCounter;
8          Set<Goal> goalsReadyToEvaluate;
9          Map<Goal, Weight> weightOfGoalToFIE;
10         // Initialization finds the contributor counts for goals, stores the weights
11         // of reused goals on FIEs, finds the leaf nodes to be evaluated based on
12         // the evaluationCounter, and adds a goal that does not have any
13         // not-yet-evaluated contributors to the ready list.
14         initialize(gm, evaluationCounter, goalsReadyToEvaluate, weightOfGoalToFIE);
15
16         while (goalsReadyToEvaluate.size() > 0) {
17             Goal goal = goalsReadyToEvaluate.get(0);
18             evaluateImpactFactorForGoal(goal, evaluations);
19             // When a goal is evaluated, counters for the goals it contributes to
20             // are decremented. Ready list is updated accordingly.
21             updateCounterAndReadyList(goal, evaluationCounter, goalsReadyToEvaluate);
22         }
23         // Finally, go through the results found for the root goals with set
24         // importance values, and scale them with their importance values.
25         for (Goal aGoal : importanceValues.keySet()) {
26             derivedSurrogateImportance = evaluations.get(aGoal)
27                 * importanceValues.get(aGoal);
28             surrogateImportances.put(aReusedGoal, derivedSurrogateImportance);
29         }
30     }
31     return surrogateImportances;
32 }
33 // This method evaluates the self impact factor of a goal & saves it to evaluations.
34 evaluateImpactFactorForGoal(Goal g, Map<Goal, SelfImpactFactor> evaluations) {
35     selfRange = 100 / g.getScalingFactor();
36     if (Goal g contributes to a FIE) {
37         selfContribution = weightOfGoalToFIE.get(g);
38         selfImpactFactor = selfContribution / selfRange;
39     } else {
40         for (each incoming contribution link aLink to Goal g) {
41             selfImpactFactor += evaluations.get(aLink.source()) * aLink.weight();
42         }
43         selfImpactFactor /= selfRange;
44     }
45     evaluations.put(g, selfImpactFactor);
46 }

```

---

### 6.5.4 Proof-of-Concept Implementation

A proof-of-concept implementation of the novel top-down evaluation algorithm for goal model reuse hierarchies as described in Section 6.5.3 has been implemented in the TouchCORE reuse tool [27], which uses feature and goal modeling in collaboration to build *concerns* in support of Concern-oriented Reuse (CORE) [20] and provides support for evaluation of goal models in reuse hierarchies [31].

To argue the feasibility of our algorithm in comparison to other possible ways to conduct a top-down evaluation, we additionally implemented two alternative algorithms.

As explained in Section 6.5.3, our algorithm (labeled “*with thresholds*” in Table 6.1) finds all of the (potentially approximate) best-case solutions, taking the threshold values into account. The algorithm goes through the complete set of solutions from a reused goal model and if any of those solutions satisfies the thresholds at the top level, it finds all of the best-case solutions (not approximations in this case); if not, it offers all of them as approximations.

First alternative algorithm (labeled “*without thresholds*” in Table 6.1) finds all of the best-case solutions, without allowing threshold values. In this case, the complete set of solutions from a reused goal model still needs to be considered for the evaluation of the reusing goal model, however, since thresholds do not exist, this algorithm does find all of the best-case solutions.

The second alternative algorithm (labeled “*without surrogates [with thresholds]*” in Table 6.1) treats the entire goal model reuse hierarchy as one flat goal model, and consequently it is able to find all of the best-case solutions even with thresholds taken into account.

As the complexity of the problem depends on both the size of the solution set (i.e., possible combinations of leaf level tasks) and how the models are structured (e.g., how external constraints are defined, how contribution links are added, and how contribution values are

assigned), for the performance evaluation, 5 different 3-level reuse hierarchies are prepared (i.e., H1-5 in Table 6.1) via randomly selecting 3 different concerns (each with 50 features and 8 cross-tree constraints (i.e., includes or excludes relationships between features) in their feature model and 30 nodes in their goal model) out of the 7 test cases with feature model constraints generated for the performance evaluations presented previously in Section 5.5. Since the goal models of those test cases had a single top-level goal with 3 intermediate goals contributing to it, those top-level goals are removed from the test cases, hence resulting in 3 new top-level goals.

At the top of each reuse hierarchy, an actor is added to the goal model with randomly generated importance values (i.e., a value in the range  $[1, 10]$ ) for each of the 3 top goals. Threshold values used are also determined at random (i.e., a multiple of 10 in the range  $[0, 100]$ ). For the reusing goal models (i.e., goal models at the top and middle levels of the hierarchies), a feature represented as a leaf level task in the goal model is picked at random to represent the impact of a reused goal (i.e., to become a feature impact element) and added once again to the goal model with a different reused goal (in the end, reusing goal models contained 30 nodes, excluding the actor at the top level).

The reused goals are also picked at random, but with one constraint that at each reuse boundary both reuse links are active (i.e., each of the two goals from the bottom level has an impact on the actor at the top level). In other words, 2 out of 3 top-level goals from each reused goal model always have an impact on the reusing goal model. To not decrease the size of the possible solution sets by ruling out some of the possibilities, no preconditions were added to the hierarchies under evaluation (i.e., none of the features were selected or unselected initially).

Table 6.1 reports the results of the performance evaluation on a computer with a 2.5 GHz processor and 16GB, 1600 MHz, DDR3 memory. The execution time is given in seconds and is calculated as the average of 20 runs to discount for program startup time. An algorithm

Table 6.1: Performance Results for Top-down Evaluation

	Execution time per reuse hierarchy (s)				
	H1	H2	H3	H4	H5
with thresholds	<b>9.22</b>	<b>222.47</b>	<b>2.19</b>	<b>5.88</b>	<b>1.58</b>
without thresholds	14.99	421.64	2.20	5.86	1.64
without surrogates [with thresholds]	> 30min	> 30min	> 30min	> 30min	> 30min

is interrupted manually after exceeding 30 minutes for an evaluation.

The performance results show that our proposed approach (i.e., *with thresholds*) makes the top down evaluation feasible compared to the *without surrogates [with thresholds]* algorithm, which does not complete in all five cases. Furthermore, comparing the *without thresholds* algorithm shows that use of threshold values may help the algorithm execute faster, which is the case for reuse hierarchies *H1* and *H2*. This result is expected since the use of a threshold value may stop the algorithm from exploring a subset of solutions further (e.g., possible solutions for **Goal X** are not explored further upon reaching its threshold in Figure 6.10) and, in return, can make it simpler to find the best-case solutions. For the goal model reuse hierarchies where this simplification is not the case (i.e., *H3*, *H4*, and *H5* where threshold values do not affect the evaluation), observed execution times for the *with thresholds* and *without thresholds* algorithms are similar as the execution of the algorithms become similar as well.

The chosen size of feature and goal models used at each level of the reuse hierarchies built for this performance evaluation go beyond the common and typical models we have encountered in the context of CORE (the largest feature and goal models in the “reusable concern library” of TouchCORE contain 26 nodes). Consequently, the results indicate that the proposed top-down evaluation mechanism is feasible in realistic settings.

The possibility of the algorithm providing approximations when the thresholds cannot be met with the best-case solutions from reused goal models is a trade-off for having higher performance in some cases.

## 6.6 Contextual Information

As previously explained in Section 1.2 as one of the five requirements for the reuse of goal models, designing a generic software artifact requires taking into consideration a wide range of scenarios in which the artifact could be reused. However, it often results in the use of placeholders rather than concrete model elements to represent unknown contextual information to keep the reusable artifact generic.

Dealing with contextual information in a reusable goal model brings two important aspects into consideration, (i) the ability to model context using the provided explicit modeling constructs in the goal model, and (ii) the ability to evaluate and analyze the goal model and the entire goal model reuse hierarchy in the presence of unknown, contextual information.

As discussed in Section 3.2.2, the systematic literature review points to lack of context representation in goal modeling approaches predominantly focusing on reuse. Although there is substantial body of work on the representation of context in a goal model (hence addressing the first aspect), the gap between the two concepts, reuse and context, creates a need for mechanisms to analyze context dependent information in a goal model reuse hierarchy.

The three approaches in Tables 3.5-3.7 that fully satisfy the fourth requirement (R.4) for contextual information and partially satisfy at least three out of four remaining requirements are Security Patterns [62][63][113][114], Contextual Goal Models [97][115][116][117][118][119][120][121][122][123][124][125], and URN [145][146][147]. These three approaches describe a reasoning mechanism without dealing with complexity emerging from the growing reuse hierarchy and provide the means to delay decisions without incorporating them into evaluation.

The Security Patterns approach does not capture external constraints and Contextual Goal Models and URN do not describe clear module boundaries.

While Security Patterns extend and use Contextual Goal Models for their context representation and specify context in terms of domain properties (i.e., facts related to a particular domain that are used to evaluate values of context tags assigned to model elements), Contextual Goal Models identify variation points in their goal models (e.g., as or-decomposition, means-end, actor dependency, root goals, and-decomposed goals, and contribution to soft-goals). On the other hand, URN makes use of KPIs and beliefs for the representation of context. Domain properties and variation points provide a lot of flexibility in the representation of contextual information. However, having a lot of flexibility and possible variations conflicts with goal model evaluation and analysis across the reuse hierarchy especially in the presence of external constraints.

Delayed decisions that are discussed in Section 6.4 originate from the lack of contextual information at certain levels within the reuse hierarchy. Considering the example three-level reuse hierarchy of Authentication, Authorization, and Bank concerns, the three respective designers of the concerns are in possession of different types of information. As a result of this, they may decide to postpone some decision making to a later point in the reuse hierarchy when more information about the context is known.

Taking the delayed decisions into consideration for the analysis and evaluation of reusable goal models implies taking into account the full range of uncertainty (i.e.,  $[0, 100]$  satisfaction range for an intentional element) imposed by the missing contextual information. Consequently, with the help of the extended evaluation algorithm we are also providing the required support to conduct an analysis of context dependent information in a goal model reuse hierarchy to a certain extent, as the information being introduced to the model at later stages is contextual.

The support is limited because it considers the contextual uncertainty and possible range

for the satisfaction values of intentional elements. This type of support allows the use of KPIs and beliefs because KPIs and beliefs can be treated in a similar way as other intentional elements in normalization and range evaluation. Furthermore, variability of intentional elements and decomposition and dependency links can also be supported. For example, an intentional element being present in the model based on a variation point and a domain property is similar to it being selected or re-exposed because in both cases it can take a satisfaction value in  $[0,100]$  range. Similar to intentional elements, presence of decomposition links and dependencies based on a variation point and a domain property (assuming it is a binary variation and not a change in the type of the link) can also be trivially handled by the evaluation. At this point, it is also important to note that while we currently support contribution links in reusable goal models, our novel evaluation mechanism could be trivially extended to support the evaluation of stakeholders, KPIs, domain properties, and decomposition and dependency links.

However, support for modeling contextual information through delaying decisions cannot handle varying contribution values based on contextual information because it conflicts with normalization. The normalization algorithm can only be run if all contribution values are known. This means that, if one of the contribution values is unknown (partial), then the potentially expensive lazy, recursive algorithm needs to be run during reuse as well, and not only during design.

Consequently, having the extended range evaluation mechanism presented in the previous section addresses not only the second requirement of goal model reuse (R.2), but also the fourth requirement (R.4) to a certain extent (i.e., partially satisfied based on the evaluation criteria in Table 3.11), as the information being introduced to the model at later stages is contextual. Based on the evaluation criteria in Table 3.9 of the systematic literature review, our approach fully satisfies the second requirement (R.2) as it makes full use of re-exposition and the associated uncertainty in re-exposed features that can be included in or excluded

from the system at a later time in the reuse hierarchy.

## 6.7 Summary

This chapter demonstrates how goal models in GRL extended with feature models can be reused in the context of Concern-Oriented, where feature and goal models are used together to specify the variation interface of a concern.

The chapter discusses extensions to the GRL metamodel enabling reuse and a novel recursive propagation-based evaluation mechanism. The new concepts incorporated into the GRL metamodel are the `ReuseLink`, the `ReusedStrategy`, and the `FeatureImpactElement`. The novel evaluation mechanism is capable of handling reusable goal models in concern hierarchies, i.e., systems are built by assembling small reusable concerns into larger reusable concerns in a concern hierarchy, while avoiding the creation of extremely large and unwieldy monolithic goal models.

With the help of the metamodel extensions, a clear reuse interface is defined for goal models, explicitly stating which elements of a reusable goal model may be reused by a reusing goal model. The interface exposes all goal model elements in a goal model except for features, the newly introduced feature impact element, and any reused elements from other lower-level concerns. These other elements are hidden from the reusing goal model. Consequently, our approach fully satisfies the fifth requirement (R.5) of providing a well-defined interface for reuse (see Section 6.2).

While GRL is used in this chapter to demonstrate reusable goal models, the results can be applied to all goal modeling notations with propagation-based evaluation mechanisms. The novel recursive evaluation mechanism is also applicable to reusable goal models that are not used in conjunction with feature models, in which case all reuse links are always active. In this case, the `FeatureImpactElement` is also not needed.



Building on top of our reusable goal model evaluation approach, we also introduce a new, reuse hierarchy-wide evaluation approach that allows the evaluation of reusable goal models in the presence of delayed decisions. The term delayed decisions refers to the remaining task selections that are postponed to a higher level in the reuse hierarchy in the context of Concern-Orientedness. With this new evaluation, our approach fully satisfies the second requirement (R.2) as it makes full use of re-exposition and the associated uncertainty in re-exposed features.

Delayed decisions originate from missing contextual information at certain levels within the reuse hierarchy. Taking these delayed decisions into consideration for the analysis and evaluation of reusable goal models signifies that our reuse hierarchy-wide evaluation approach is taking into account the full range of uncertainty imposed by this missing contextual information, and hence, is providing the required support to analyze context dependent information in a goal model reuse hierarchy to partially satisfy the fourth requirement (R.4) based on the evaluation criteria in Table 3.11.

Last, but not the least, we introduce the novel top-down evaluation algorithm for reusable goal models that allows designers to find solution sets to satisfy the stakeholders of their system and their goals, even when reuse hierarchies grow larger and external constraints are imposed on the goal models. For this novel top-down evaluation, we use three modeling constructs (i.e., actor, importance, and threshold) for goal prioritization and an adaptation of the existing bottom-up evaluation algorithm to top-down evaluation of goal model reuse hierarchies. Secondly, to benefit from reuse boundaries and allow the goal model of each reuse level to be evaluated individually in the top-down evaluation, we introduce an algorithm that propagates top-level importance values down the reuse hierarchy to surrogate actors and finds the derived importance values for reused goals at the lower level goal models. Finally, we show the feasibility of this novel top-down evaluation with a proof-of-concept implementation in the TouchCORE reuse tool.

By allowing the goal model evaluation to span the entire reuse hierarchy and incrementally evaluating levels in the reuse hierarchy (not only for bottom-up, but also for top-down evaluation), our approach fully satisfies the first requirement (R.1) of providing support for trade-off reasoning through the reuse hierarchy as stated in Section 1.2.

The following chapter summarizes the thesis and states future research directions to build on this work.

# Chapter 7

## Conclusions

This chapter first summarizes the thesis' contributions in Section 7.1, followed by a discussion of future work and research directions in Section 7.2.

### 7.1 Contributions

In the early requirements phase, goal models are used to capture hierarchical representations of stakeholder objectives, system qualities, potential solutions, and their relationships. With the help of goal models, requirements engineers can understand the goals of stakeholders and explore the alternative solutions with their impacts on the goals. Reuse of goal models, however, has received limited attention.

This thesis investigates the requirements for goal model reuse and proposes an approach that addresses the identified five requirements: providing trade-off reasoning capability via goal model evaluation through reuse hierarchies while taking delayed decisions and constraints imposed by other models into account, allowing context dependent information to be modeled and analyzed (to a certain extent), and providing an interface for reuse. To address these five requirements, we introduce novel goal model evaluation mechanisms for bottom-up and top-down evaluation based on a lazy, recursive algorithm, which in return, allow the design, modularization, evaluation, and analysis of reusable goal models in reuse hierarchies in the presence of external constraints taking delayed decisions and contextual information into account.

## 7.1. Contributions

Table 7.1: Analysis of our approach with respect to the SLR evaluation criteria

Approach	Search String	R.1	R.2	R.3	R.4	R.5	Context Representation	Supplementary Models	Tool Support
Reusable Goal Models [17] [24][28][29][30][31][32]	1*	●	●	●	○	●	delayed decisions	feature models & workflow models	●

\* only [28][29][31] found by the search

The hypotheses of this research are defined as: *(a)* current goal modeling technologies do not fully satisfy the five requirements, and *(b)* it is feasible to determine the required goal model concepts and tool-supported algorithms for goal model reuse that satisfy the five requirements.

This research is assessed via a systematic literature review to evaluate the findings qualitatively (hence answering the first hypothesis) and proof-of-concept implementations of proposed improvements as well as the experiments and performance and scalability assessments to evaluate the feasibility of the proposed approach quantitatively (hence answering the second hypothesis). As a result, the two hypotheses of this work hold.

Looking back at the five requirements identified in Chapter 1 (see Table 7.1), our approach fully satisfies the first requirement (R.1) of providing support for trade-off reasoning through the reuse hierarchy by allowing the goal model evaluation to span the entire reuse hierarchy and incrementally evaluating levels in the reuse hierarchy (see Section 4.3). Furthermore, the incremental evaluation mechanism prevents backtracking through the whole reuse hierarchy leading to reduced complexity in both bottom-up and top-down evaluation of goal models (see Section 6.5).

In order to fully satisfy the second requirement (R.2) of providing the means to delay decisions to a later point in the reuse hierarchy when more complete information is available, while still allowing the reusable artifact to be analyzed, our approach makes full use of re-

exposition and the associated uncertainty in re-exposed features that can be included in or excluded from the system at a later time in the reuse hierarchy (see Section 6.4). As shown by the results of the systematic literature review, none of the existing goal modeling approaches provided a complete mechanism to delay decisions in the reuse hierarchy. The range evaluation algorithm introduced in this work addresses this problem in the literature.

Our approach fully satisfies the third requirement (R.3) of taking constraints imposed by other modeling notations into account when evaluating reusable goal models with an updated evaluation mechanism that handles external constraints imposed by feature models and workflow models (see Chapter 5). Moreover, the lazy, recursive algorithm introduced for this purpose allows flexible integration of additional constraints that may be imposed on goal models by other modeling formalisms in the future and therefore paves the way for the development of a general framework that allows general constraints on goal models to be expressed and linked to modeling constructs in other modeling formalisms.

For the representation and analysis of contextual information as required by the fourth requirement (R.4), partial support is already made available by our approach (see Section 6.4) but it is limited. Incorporating delayed decisions into the analysis and evaluation of reusable goal models means that the full range of uncertainty (i.e.,  $[0, 100]$  range) imposed by the missing contextual information is taken into account for the satisfaction value of an intentional element. Consequently, with the help of the extended evaluation algorithm our approach provides the required support to analyze context dependent information in a goal model reuse hierarchy to a certain extent, as the information being introduced to the model at later stages is contextual. However, support is partial because context is not explicitly modeled and while contextual changes to satisfaction values of intentional elements can be dealt with, contextual changes to contribution values conflict with the normalization algorithm.

Since there already is substantial body of work on the various modeling constructs to represent context in goal models, our focus is on facilitating the evaluation and analysis

mechanisms to support contextual analysis while a full treatment of context is not in the scope of this thesis and hence future work.

The fifth requirement (R.5) of providing a well-defined interface for reuse is fully satisfied by our approach as it provides a clearly defined reuse interface with the necessary metamodel extensions, and the model boundaries are maintained after reuse (see Section 6.2). Looking at the state of the art, it was deemed necessary to determine which goal model constructs should be visible to a reusing party and which should not be visible to develop techniques to manage the visibility of a modeling element and under which circumstances it is allowed to change (see Section 3.2.3). Newly introduced modeling constructs and metamodel extensions address this problem and answer the question of what happens to the visibility of modeling elements of a reused goal model in the reusing model.

Overall, our proposed approach is a substantial improvement over existing goal modeling approaches surveyed in Chapter 3 by the systematic literature review. While our approach fully satisfies four out of five requirements and partially satisfies the remaining requirement, the systematic literature review shows that existing goal modeling approaches at most satisfy one requirement and partially satisfy the other four requirements.

Even though the changes mentioned to goal model evaluation and GRL notation have been applied mostly on connected goal models, which are used in combination with other modeling notations, the contributions are also applicable to standalone goal models. It is also important to note that standalone goal models do not need to fulfill the requirement for handling external constraints (R.2) as they are used in isolation.

## 7.2 Future Work and Research Directions

Future work and research directions for this work can be broadly grouped into four categories:

- Our approach is currently tailored to specific modeling formalisms (e.g., feature models

and workflow models) in the way it takes external constraints imposed by other modeling notations into account during the evaluation of reusable goal models. It does not necessarily apply to the constraints imposed by yet another modeling formalism. Techniques have not yet been developed for other modeling formalisms connected with goal models (e.g., see organizational models, structural models, and state-based models in the “*Supplementary Models*” columns of Tables 3.5-3.7). Consequently, building on top of our lazy, recursive algorithm that allows flexible integration of additional constraints in goal model analysis and evaluation, instead of developing individual solutions for each modeling formalism that may impose a constraint on reusable goal models, a general framework could be developed that allows general constraints on goal models to be expressed and linked to modeling constructs in other modeling formalisms.

- In addition to the existing support for analysis and evaluation of contextual information in the form of delayed decisions, to further ensure the compliance of our goal modeling approach with the fourth requirement (R.4) (i.e., to fully satisfy the fourth requirement), the GRL metamodel could be extended with the identified concepts from the literature review to consequently support explicit modeling of context dependent information.
- The evaluation of reusable goal models could be trivially extended to support the evaluation of stakeholders, KPIs, and decomposition and dependency links in addition to already supported contribution links.
- Last but not the least, empirical studies need to be performed to investigate how easy it is to use our proposed approach with relative contribution values (e.g., how they would impact the process of decision-making in assigning the contributions), in reuse hierarchies (e.g., how modelers working at different levels of the hierarchy make

use of the range evaluation and re-exposition and how it would influence the design of generic artifacts), and while building an application at the top-level of a reuse hierarchy (e.g., performance and usability of the top-down evaluation with real-life, reusable goal models).



# Bibliography

- [1] P. Zave, “Classification of research efforts in requirements engineering,” *ACM Comput. Surv.*, vol. 29, no. 4, pp. 315–321, 1997. [Online]. Available: <http://doi.acm.org/10.1145/267580.267581>
- [2] K. Czarnecki, M. Antkiewicz, C. H. P. Kim, S. Lau, and K. Pietroszek, “Model-driven software product lines,” in *Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, October 16-20, 2005, San Diego, CA, USA*, 2005, pp. 126–127. [Online]. Available: <http://doi.acm.org/10.1145/1094855.1094896>
- [3] I. A. ElSayed, Z. Ezz, and E. Nasr, “Goal modeling techniques in requirements engineering: A systematic literature review,” *JCS*, vol. 13, no. 9, pp. 430–439, 2017. [Online]. Available: <https://doi.org/10.3844/jcssp.2017.430.439>
- [4] E. S. K. Yu and J. Mylopoulos, “Why goal-oriented requirements engineering,” in *REFSQ 1998, Italy*, 1998, pp. 15–22.
- [5] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, ser. International Series in Software Engineering. Springer, October 1999, vol. 5.

- [6] E. S.-K. Yu, “Modelling strategic relationships for process reengineering,” Ph.D. dissertation, Toronto, Ont., Canada, Canada, 1996, uMI Order No. GAXNN-02887 (Canadian dissertation).
- [7] International Telecommunication Union (ITU-T), “Recommendation Z.151 (10/12): User Requirements Notation (URN) - Language Definition,” approved October 2012.
- [8] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, “Tropos: An agent-oriented software development methodology,” *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, May 2004. [Online]. Available: <http://dx.doi.org/10.1023/B:AGNT.0000018806.20944.ef>
- [9] A. Dardenne, A. van Lamsweerde, and S. Fickas, “Goal-directed requirements acquisition,” *Sci. Comput. Program.*, vol. 20, no. 1-2, pp. 3–50, Apr. 1993. [Online]. Available: [http://dx.doi.org/10.1016/0167-6423\(93\)90021-G](http://dx.doi.org/10.1016/0167-6423(93)90021-G)
- [10] C. W. Krueger, “Software reuse,” *ACM Comput. Surv.*, vol. 24, no. 2, pp. 131–183, Jun. 1992. [Online]. Available: <http://doi.acm.org/10.1145/130844.130856>
- [11] NATO, “Standard for the Development of Reusable Software Components,” GTE Government Systems, Chantilly, VA, USA, Tech. Rep., 1994.
- [12] J. A. Lewis, S. M. Henry, D. G. Kafura, and R. S. Schulman, “An empirical study of the object-oriented paradigm and software reuse,” in *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA’91), Sixth Annual Conference, Phoenix, Arizona, USA, October 6-11, 1991, Proceedings.*, 1991, pp. 184–196. [Online]. Available: <http://doi.acm.org/10.1145/117954.117969>
- [13] V. R. Basili, L. C. Briand, and W. L. Melo, “How reuse influences productivity in object-oriented systems,” *Commun. ACM*, vol. 39, no. 10, pp. 104–116, 1996.

- [14] W. C. Lim, “Effects of reuse on quality, productivity, and economics,” *IEEE Softw.*, vol. 11, no. 5, pp. 23–30, Sep. 1994.
- [15] A. van Lamsweerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009. [Online]. Available: <http://eu.wiley.com/WileyCDA/WileyTitle/productCd-EHEP000863.html>
- [16] L. M. Cysneiros, V. Werneck, and A. Kushniruk, “Reusable knowledge for satisficing usability requirements,” in *13th IEEE International Conference on Requirements Engineering (RE 2005), 29 August - 2 September 2005, Paris, France*. IEEE Computer Society, 2005, pp. 463–464. [Online]. Available: <http://dx.doi.org/10.1109/RE.2005.60>
- [17] M. B. Duran, “Reusable goal models.” in *RE*. IEEE Computer Society, 2017, pp. 532–537. [Online]. Available: <http://dblp.uni-trier.de/db/conf/re/re2017.html#Duran17>
- [18] G. Mussbacher and J. Kienzle, “A vision for generic concern-oriented requirements reuse<sup>re@21</sup>,” in *21st IEEE International Requirements Engineering Conference, RE 2013, Rio de Janeiro-RJ, Brazil, July 15-19, 2013*. IEEE Computer Society, 2013, pp. 238–249. [Online]. Available: <http://dx.doi.org/10.1109/RE.2013.6636724>
- [19] J. Kienzle, W. A. Abed, and J. Klein, “Aspect-oriented multi-view modeling,” in *AOSD 2009, Virginia, USA*, 2009, pp. 87–98.
- [20] O. Alam, J. Kienzle, and G. Mussbacher, “Concern-oriented software design,” in *Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 - October 4, 2013. Proceedings*, ser. Lecture Notes in Computer Science, A. Moreira, B. Schätz, J. Gray, A. Vallecillo, and P. J. Clarke, Eds., vol. 8107. Springer, 2013, pp. 604–621. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-41533-3\\_37](http://dx.doi.org/10.1007/978-3-642-41533-3_37)

- [21] A. van Lamsweerde, “Goal-oriented requirements engineering: A guided tour,” in *5th IEEE International Symposium on Requirements Engineering (RE 2001)*, 27-31 August 2001, Toronto, Canada, 2001, p. 249. [Online]. Available: <https://doi.org/10.1109/ISRE.2001.948567>
- [22] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, and E. Yu, “Evaluating goal models within the goal-oriented requirement language,” *Int. J. Intell. Syst.*, vol. 25, no. 8, pp. 841–877, Aug. 2010. [Online]. Available: <http://dx.doi.org/10.1002/int.v25:8>
- [23] J. Kienzle, G. Mussbacher, P. Collet, and O. Alam, “Delaying decisions in variable concern hierarchies,” in *2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, ser. GPCE 2016. New York, NY, USA: ACM, 2016, pp. 93–103.
- [24] M. B. Duran and G. Mussbacher, “Investigation of feature run-time conflicts on goal model-based reuse,” *Information Systems Frontiers*, pp. 1–21, 2016. [Online]. Available: <http://dx.doi.org/10.1007/s10796-016-9657-7>
- [25] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.
- [26] J. Kienzle, G. Mussbacher, O. Alam, M. Schöttle, N. Belloir, P. Collet, B. Combemale, J. DeAntoni, J. Klein, and B. Rumpe, “VCU: the three dimensions of reuse,” in *Software Reuse: Bridging with Social-Awareness - 15th International Conference, ICSR 2016, Limassol, Cyprus, June 5-7, 2016, Proceedings*, ser. Lecture Notes in Computer Science, G. M. Kapitsaki and E. S. de Almeida, Eds., vol. 9679. Springer, 2016, pp. 122–137. [Online]. Available: [http://dx.doi.org/10.1007/978-3-319-35122-3\\_9](http://dx.doi.org/10.1007/978-3-319-35122-3_9)

- [27] TouchCORE tool, “(version 7.0.2),” 2018, <http://touchcore.cs.mcgill.ca>.
- [28] M. B. Duran, G. Mussbacher, N. Thimmegowda, and J. Kienzle, “On the reuse of goal models.” in *SDL Forum*, ser. Lecture Notes in Computer Science, J. Fischer, M. Scheidgen, I. Schieferdecker, and R. Reed, Eds., vol. 9369. Springer, 2015, pp. 141–158. [Online]. Available: <http://dblp.uni-trier.de/db/conf/sdl/sdl2015.html#DuranMTK15>
- [29] M. B. Duran, A. N. Pina, and G. Mussbacher, “Evaluation of reusable concern-oriented goal models,” in *2015 IEEE International Model-Driven Requirements Engineering Workshop, MoDRE, Ottawa, ON, Canada, August 24, 2015*. IEEE Computer Society, 2015, pp. 53–62. [Online]. Available: <http://dx.doi.org/10.1109/MoDRE.2015.7343876>
- [30] M. B. Duran and G. Mussbacher, “Evaluation of goal models in reuse hierarchies with delayed decisions.” in *RE Workshops*. IEEE Computer Society, 2017, pp. 6–15. [Online]. Available: <http://dblp.uni-trier.de/db/conf/re/re2017w.html#DuranM17>
- [31] R. Alexandre, C. Camillieri, M. Duran, A. Pina, M. Schöttle, J. Kienzle, and G. Mussbacher, “Support for evaluation of impact models in reuse hierarchies with jUCMNav and TouchCORE,” in *CEUR Workshop Proceedings*, Kulkarni V. and Badreddin O., Eds., vol. 1554. CEUR-WS, 2015, pp. 28–31.
- [32] M. B. Duran and G. Mussbacher, “Top-down evaluation of reusable goal models,” in *New Opportunities for Software Reuse - 17th International Conference, ICSR 2018, Madrid, Spain, May 21-23, 2018, Proceedings*, 2018, pp. 76–92. [Online]. Available: [https://doi.org/10.1007/978-3-319-90421-4\\_5](https://doi.org/10.1007/978-3-319-90421-4_5)
- [33] J. Horkoff and E. Yu, “Analyzing goal models: Different approaches and how to choose among them,” in *Proceedings of the 2011 ACM Symposium on Applied*

- Computing*, ser. SAC '11. New York, NY, USA: ACM, 2011, pp. 675–682. [Online]. Available: <http://doi.acm.org/10.1145/1982185.1982334>
- [34] A. Pourshahid, G. Richards, and D. Amyot, “Toward a goal-oriented, business intelligence decision-making framework,” in *E-Technologies: Transformation in a Connected World - 5th International Conference, MCETECH 2011, Les Diablerets, Switzerland, January 23-26, 2011, Revised Selected Papers*, ser. Lecture Notes in Business Information Processing, G. Babin, K. Stanoevska-Slabeva, and P. Kropf, Eds., vol. 78. Springer, 2011, pp. 100–115. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-20862-1\\_7](http://dx.doi.org/10.1007/978-3-642-20862-1_7)
- [35] J. Horkoff and E. Yu, “Evaluating goal achievement in enterprise modeling – an interactive procedure and experiences,” in *The Practice of Enterprise Modeling*, A. Persson and J. Stirna, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 145–160.
- [36] A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, and A. J. Forster, “Business process management with the user requirements notation,” *Electronic Commerce Research*, vol. 9, no. 4, pp. 269–316, 2009.
- [37] P. Giorgini, J. Mylopoulos, and R. Sebastiani, “Goal-oriented requirements analysis and reasoning in the tropos methodology,” *Eng. Appl. of AI*, vol. 18, no. 2, pp. 159–171, 2005.
- [38] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, “Reasoning with goal models,” in *Proceedings of the 21st International Conference on Conceptual Modeling*, ser. ER '02. London, UK, UK: Springer-Verlag, 2002, pp. 167–181.

- [39] R. Sebastiani, P. Giorgini, and J. Mylopoulos, *Simple and Minimum-Cost Satisfiability for Goal Models*, 2004, pp. 20–35. [Online]. Available: [https://doi.org/10.1007/978-3-540-25975-6\\_4](https://doi.org/10.1007/978-3-540-25975-6_4)
- [40] E. Letier and A. van Lamsweerde, “Reasoning about partial goal satisfaction for requirements and design engineering,” in *12th ACM SIGSOFT Intl. Symposium on Foundations of Software Engineering, USA*, 2004, pp. 53–62.
- [41] B. Kitchenham, “Procedures for performing systematic reviews,” Department of Computer Science, Keele University, UK, Keele University. Technical Report TR/SE-0401, 2004.
- [42] Roy Rosenzweig Center for History and New Media at George Mason University and the Corporation for Digital Scholarship, “Zotero,” software available from <https://www.zotero.org/>.
- [43] J.-F. Roy, “Requirement engineering with URN: Integrating goals and scenarios,” Master’s thesis, University of Ottawa, Canada, 2007.
- [44] D. Colomer and J. Franch Gutiérrez, “i\* modules: a jUCMNav implementation,” in *iStar 5th International i\* workshop: 29-30th August, 2011, Trento, Italy*. CEUR Workshop Proceedings, 2011, pp. 178–180.
- [45] J. Castro, M. Lucena, C. Silva, F. Alencar, E. Santos, and J. Pimentel, “Changing attitudes towards the generation of architectural models,” *Journal of Systems and Software*, vol. 85, no. 3, pp. 463–479, 2012.
- [46] J. Pimentel, M. Lucena, J. Castro, C. Silva, E. Santos, and F. Alencar, “Deriving software architectural models from requirements models for adaptive systems: the

- STREAM-A approach,” in *Requirements Engineering*. Springer-Verlag, Jun. 2011, vol. 17, pp. 259–281, doi: 10.1007/s00766-011-0126-z.
- [47] G. Mussbacher, D. Amyot, J. Araújo, A. Moreira, and M. Weiss, “Visualizing Aspect-Oriented Goal Models with AoGRL,” in *Requirements Engineering Visualization, 2007. REV 2007. Second International Workshop on*. IEEE, Oct. 2007, pp. 1–1.
- [48] G. Mussbacher, “Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models,” in *Models in Software Engineering*, ser. Lecture Notes in Computer Science, H. Giese, Ed. Springer Berlin Heidelberg, Sep. 2007, no. 5002, pp. 305–316, doi: 10.1007/978-3-540-69073-3\_32.
- [49] G. Mussbacher and D. Amyot, “Extending the User Requirements Notation with Aspect-Oriented Concepts,” in *SDL 2009: Design for Motes and Mobiles*, ser. Lecture Notes in Computer Science, R. Reed, A. Bilgic, and R. Gotzhein, Eds. Springer Berlin Heidelberg, Sep. 2009, no. 5719, pp. 115–132, doi: 10.1007/978-3-642-04554-7\_8.
- [50] G. Mussbacher, D. Amyot, J. Araújo, and A. Moreira, “Modeling Software Product Lines with AoURN,” in *AOSD Workshop on Early Aspects*, ser. EA ’08. New York, NY, USA: ACM, 2008, pp. 2:1–2:8.
- [51] G. Mussbacher, D. Amyot, and J. Whittle, “Composing Goal and Scenario Models with the Aspect-Oriented User Requirements Notation Based on Syntax and Semantics,” in *Aspect-Oriented Requirements Engineering*, A. Moreira, R. Chitchyan, J. Araújo, and A. Rashid, Eds. Springer, 2013, pp. 77–99, doi: 10.1007/978-3-642-38640-4\_5.
- [52] J. C. S. d. P. Leite, Y. Yu, L. Liu, E. S. K. Yu, and J. Mylopoulos, “Quality-Based Software Reuse,” in *Advanced Information Systems Engineering*, ser. Lecture Notes in



- Computer Science, O. Pastor and J. F. e. Cunha, Eds. Springer Berlin Heidelberg, Jun. 2005, no. 3520, pp. 535–550, doi: 10.1007/11431855\_37.
- [53] N. Niu, Y. Yu, B. González-Baixauli, N. Ernst, J. C. S. d. P. Leite, and J. Mylopoulos, “Aspects across Software Life Cycle: A Goal-Driven Approach,” in *Transactions on Aspect-Oriented Software Development VI*, ser. Lecture Notes in Computer Science, S. Katz, H. Ossher, R. France, and J.-M. Jézéquel, Eds. Springer Berlin Heidelberg, 2009, no. 5560, pp. 83–110, doi: 10.1007/978-3-642-03764-1\_3.
- [54] J. Castro, M. Kolp, L. Liu, and A. Perini, “Dealing with Complexity Using Conceptual Models Based on Tropos,” in *Conceptual Modeling: Foundations and Applications*, ser. Lecture Notes in Computer Science, A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, Eds. Springer, 2009, no. 5600, pp. 335–362, doi: 10.1007/978-3-642-02463-4\_18.
- [55] F. Alencar, J. Castro, A. Moreira, J. Araújo, C. Silva, R. Ramos, and J. Mylopoulos, “Integration of Aspects with i\* Models,” in *Agent-Oriented Information Systems IV*, ser. Lecture Notes in Computer Science, M. Kolp, B. Henderson-Sellers, H. Mouratidis, A. Garcia, A. K. Ghose, and P. Bresciani, Eds. Springer Berlin Heidelberg, 2008, no. 4898, pp. 183–201, doi: 10.1007/978-3-540-77990-2\_11.
- [56] L. F. da Silva, “An aspect-oriented approach to model requirements,” in *IEEE International Conference on Requirements Engineering, Doctoral Consortium*, 2005, p. 24.
- [57] L. F. d. Silva and J. C. S. d. P. Leite, “Aspect-Oriented Goal Modeling and Composition with AOV-Graph,” in *Aspect-Oriented Requirements Engineering*, A. Moreira, R. Chitchyan, J. Araújo, and A. Rashid, Eds. Springer Berlin Heidelberg, 2013, pp. 101–120, doi: 10.1007/978-3-642-38640-4\_6.

- [58] A. Gil and J. Araujo, “AspectKAOS: integrating early-aspects into KAOS,” in *15th workshop on Early aspects*. ACM, 2009, pp. 31–36.
- [59] A. T. V. Gil, “Integrating early aspects with goal-oriented requirements engineering,” Ph.D. dissertation, FCT-UNL, 2008.
- [60] F. Semmak, C. Gnaho, and R. Laleau, “Extended kaos to support variability for goal oriented requirements reuse,” in *MoDISE-EUS*, ser. CEUR Workshop Proceedings, vol. 341. CEUR-WS.org, 2008, pp. 22–33.
- [61] F. Semmak, R. Laleau, and C. Gnaho, “Supporting variability in goal-based requirements,” in *Proceedings of the Third IEEE International Conference on Research Challenges in Information Science, RCIS 2009, Fès, Morocco, 22-24 April 2009*, A. Flory and M. Collard, Eds. IEEE, 2009, pp. 237–246.
- [62] T. Li, J. Horkoff, and J. Mylopoulos, “Integrating Security Patterns with Security Requirements Analysis Using Contextual Goal Models.” Springer Berlin Heidelberg, 2014, pp. 208–223, doi: 10.1007/978-3-662-45501-2\_15.
- [63] T. Li and J. Mylopoulos, “Modeling and applying security patterns using contextual goal models,” in *Proceedings of the Seventh International i\* Workshop co-located with the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), Thessaloniki, Greece, June 16-17, 2014.*, ser. CEUR Workshop Proceedings, F. Dalpiaz and J. Horkoff, Eds., vol. 1157. CEUR-WS.org, 2014. [Online]. Available: <http://ceur-ws.org/Vol-1157/paper11.pdf>
- [64] S. A. Behnam, D. Amyot, G. Mussbacher, E. Braun, N. Cartwright, and M. Saucier, “Using the Goal-oriented pattern family framework for modelling outcome-based regu-

- lations,” in *Requirements Patterns (RePa), 2012 IEEE 2nd Intl. Workshop on.* IEEE, 2012, pp. 35–40.
- [65] S. A. Behnam and D. Amyot, “Evolution mechanisms for goal-driven pattern families used in business process modelling,” *International Journal of Electronic Business*, vol. 10, no. 3, pp. 254–291, 2013.
- [66] S. A. Behnam, D. Amyot, and G. Mussbacher, “Towards a Pattern-Based Framework for Goal-Driven Business Process Modeling,” in *Software Engineering Research, Management and Applications (SERA), 2010 Eighth ACIS Intl. Conf. on.* IEEE, 2010, pp. 137–145.
- [67] S. A. Behnam, “Goal-oriented Pattern Family Framework for Business Process Modeling,” Ph.D. dissertation, University of Ottawa, 2012.
- [68] W. Cheah, L. Sterling, and K. Taveter, “Task Knowledge Patterns Reuse in Multi-Agent Systems Development,” in *Principles and Practice of Multi-Agent Systems*, ser. Lecture Notes in Computer Science, N. Desai, A. Liu, and M. Winikoff, Eds. Springer Berlin Heidelberg, Nov. 2010, no. 7057, pp. 459–474, doi: 10.1007/978-3-642-25920-3\_33.
- [69] M. Strohmaier, J. Horkoff, E. Yu, J. Aranda, and S. Easterbrook, “Can Patterns Improve i\* Modeling? Two Exploratory Studies,” in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, B. Paech and C. Rolland, Eds. Springer, 2008, no. 5025, pp. 153–167, doi: 10.1007/978-3-540-69062-7\_16.

- [70] L. A. Hermoye, A. Van Lamsweerde, and D. E. Perry, *Attack Patterns for Security Requirements Engineering*, 2006. [Online]. Available: <https://hostdb.ece.utexas.edu/~perry/work/papers/060908-LH-threats.pdf>
- [71] L. Chung and S. Supakkul, “Capturing and reusing functional and non-functional requirements knowledge: a goal-object pattern approach,” in *Information Reuse and Integration, 2006 IEEE International Conference on*. IEEE, 2006, pp. 539–544.
- [72] I. A. M. El-Maddah and T. S. E. Maibaum, “Goal-oriented requirements analysis for process control systems design,” in *1st ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2003), 24-26 June 2003, Mont Saint-Michel, France, Proceedings*, 2003, pp. 45–46. [Online]. Available: <https://doi.org/10.1109/MEMCOD.2003.1210085>
- [73] I. A. M. El-Maddah and T. S. E. Maibaum, “Requirements-Reuse Using GOPCSD: Component-Based Development of Process Control Systems,” in *Software Reuse: Methods, Techniques, and Tools*, ser. Lecture Notes in Computer Science, J. Bosch and C. Krueger, Eds. Springer Berlin Heidelberg, Jul. 2004, no. 3107, pp. 318–328, doi: 10.1007/978-3-540-27799-6\_27.
- [74] B. González-Baixaui, M. A. Laguna, and J. C. S. do Prado Leite, “Using Goal-Models to Analyze Variability.” in *VaMoS*, 2007, pp. 101–107.
- [75] P. Spoletini, A. Ferrari, and S. Gnesi, “Context transformations for goal models,” in *Model-Driven Requirements Engineering Workshop (MoDRE), 2014 IEEE 4th International*. IEEE, 2014, pp. 17–26.
- [76] M. Benaroch, “Knowledge modeling directed by situation-specific models,” *International journal of human-computer studies*, vol. 49, no. 2, pp. 121–157, 1998.

- [77] H. Chengwan and T. Chengmao, “Goal-Based Pointcut,” *International Journal of Digital Content Technology and its Applications*, vol. 7, no. 5, p. 93, 2013.
- [78] K. Cooper, S. P. Abraham, R. S. Unnithan, L. Chung, and S. Courtney, “Integrating visual goal models into the Rational Unified Process,” *Journal of Visual Languages & Computing*, vol. 17, no. 6, pp. 551–583, Dec. 2006.
- [79] R. Girardi and C. Faria, “A generic ontology for the specification of domain models,” in *1st International Workshop on Component Engineering Methodology (WCEM’03) at Second International Conference on Generative Programming and Component Engineering*, Ed. Sven Overhage and Klaus Turowski, 2003, pp. 41–50.
- [80] —, “An ontology-based technique for the specification of domain and user models in multi-agent domain engineering,” *CLEI electronic journal*, vol. 7, no. 1, p. 7, 2004.
- [81] R. Girardi, C. Faria, and L. Marinho, “Ontology-based domain modeling of multi-agent systems,” in *OOPLSA Workshop*, 2004, pp. 295–308.
- [82] R. Girardi and A. Leite, “The Specification of Requirements in the MADAE-Pro Software Process,” *iSys-Revista Brasileira de Sistemas de Informação*, vol. 3, 2011.
- [83] R. Girardi and A. N. Lindoso, “An Ontology-based Methodology for Multiagent Domain Engineering,” in *Artificial intelligence, 2005. epia 2005. portuguese conference on*. IEEE, 2005, pp. 321–324.
- [84] —, “DDEMAS: A domain design technique for multi-agent domain engineering,” pp. 141–150, 2005. [Online]. Available: [https://doi.org/10.1007/11568346\\_16](https://doi.org/10.1007/11568346_16)
- [85] M. A. Laguna and B. González-Baixaui, “Goals and MDA in Product Line Requirements Engineering,” *Departmente of Computer Science, University of Valladolid, Valladolid (Spain) GIRO-2005-01*, 2005.

- [86] M. A. Laguna and B. Gonzalez-Baixauli, “Requirements variability models: meta-model based transformations,” in *symposia on Metainformatics*. ACM, 2005, p. 9.
- [87] A. N. Lindoso and R. Girardi, “The SRAMO Techique for Analysis and Reuse of Requirements in Multi-agent Application Engineering.” in *WER*, 2006, pp. 41–50.
- [88] P. H. Meland, E. A. Gjøere, and S. Paul, “The use and usefulness of threats in goal-oriented modelling,” in *2013 International Conference on Availability, Reliability and Security, ARES 2013, Regensburg, Germany, September 2-6, 2013*, 2013, pp. 428–436.
- [89] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao, “Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation,” in *2014 IEEE 22nd Intl. Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 113–122.
- [90] E. Santos, J. Pimentel, J. Castro, and A. Finkelstein, “On the dynamic configuration of business process models,” in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2012, pp. 331–346.
- [91] M. Schöttle, O. Alam, J. Kienzle, and G. Mussbacher, “On the modularization provided by concern-oriented reuse,” in *Companion Proceedings of the 15th International Conference on Modularity*. ACM, 2016, pp. 184–189.
- [92] D. Stefan and E. Letier, “Supporting sustainability decisions in large organisations,” in *ICT for Sustainability 2014 (ICT4S-14), Stockholm, Sweden, August 25, 2014*. Atlantis Press, 2014. [Online]. Available: <https://doi.org/10.2991/ict4s-14.2014.41>
- [93] V. Sugumaran and S. Park, “A Knowledge-Based Agent Modeling and Design Environment,” *AMCIS 2000 Proceedings*, p. 19, 2000.

- [94] J. Wang, Z. Feng, J. Zhang, P. C. K. Hung, K. He, and L.-J. Zhang, “A Unified RGPS-Based Approach Supporting Service-Oriented Process Customization,” in *Web Services Foundations*, A. Bouguettaya, Q. Z. Sheng, and F. Daniel, Eds. Springer New York, 2014, pp. 657–682, doi: 10.1007/978-1-4614-7518-7\_26.
- [95] P. Donzelli, “A goal-driven and agent-based requirements engineering framework\*,” in *Requirements Engineering*. Springer-Verlag, Jul. 2003, vol. 9, pp. 16–39, doi: 10.1007/s00766-003-0170-4.
- [96] Z. Feng, K. He, R. Peng, J. Wang, and Y. Ma, “Towards merging goal models of networked software,” in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE’2009), Boston, Massachusetts, USA, July 1-3, 2009*, 2009, pp. 178–184.
- [97] R. Ali, F. Dalpiaz, and P. Giorgini, “A goal modeling framework for self-contextualizable software,” in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 326–338.
- [98] K. Hoesch-Klohe, A. Ghose, and H. K. Dam, “A Framework to support the Maintenance of Formal Goal Models (Technical Report),” 2013. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.396.7425&rep=rep1&type=pdf>
- [99] E. Kavakli, “Goal-Oriented Requirements Engineering: A Unifying Framework,” in *Requirements Engineering*. Springer-Verlag London Limited, Jan. 2002, vol. 6, pp. 237–251, doi: 10.1007/PL00010362.

- [100] S. Overbeek, U. Frank, and C. Köhling, “A language for multi-perspective goal modelling: Challenges, requirements and solutions,” *Computer Standards & Interfaces*, vol. 38, pp. 1–16, Feb. 2015.
- [101] L. F. Silva and J. C. S. do Prado Leite, “Generating requirements views: A transformation-driven approach,” *Electronic Communications of the EASST*, vol. 3, 2007.
- [102] R. Stegers, A. Teije, and F. Van Harmelen, “From natural language to formal proof goal - Structured goal formalisation applied to medical guidelines,” in *From natural language to formal proof goal - Structured goal formalisation applied to medical guidelines*, ser. 15th International Conference on Knowledge Engineering and Knowledge Management, EKAW 2006. Pödebrady: Springer Verlag, 2006, vol. 4248 LNAI, no. 4248, pp. 51–58, doi: 10.1007/11891451\_8.
- [103] B. Vazquez, A. Martinez, A. Perini, H. Estrada, and M. Morandini, “Enriching Organizational Models through Semantic Annotation,” *Procedia Technology*, vol. 7, pp. 297–304, 2013.
- [104] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J. Leite, “From stakeholder goals to high-variability software design,” Technical report csrg-509, University of Toronto, Tech. Rep., 2005.
- [105] S. Zhiqi, “Goal-oriented Modeling for Intelligent Agents and their Applications,” Ph.D. dissertation, Nanyang Technological University, 2005.
- [106] E. Letier, “Reasoning about agents in goal-oriented requirements engineering,” Ph.D. dissertation, Université Catholique de Louvain, Belgium, 2001.



- [107] J. M. Morales, E. Navarro, P. Sánchez, and D. Alonso, “A controlled experiment to evaluate the understandability of KAOS and i\* for modeling Teleo-Reactive systems,” *Journal of Systems and Software*, vol. 100, pp. 1–14, Feb. 2015.
- [108] ———, “TRiStar: an i\* extension for teleo-reactive systems requirements specifications,” in *30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 283–288.
- [109] H. S. Al-Subaie and T. S. Maibaum, “Evaluating the effectiveness of a goal-oriented requirements engineering method,” in *Comparative Evaluation in Requirements Engineering, 2006. CERE’06. Fourth International Workshop on*. IEEE, 2006, pp. 8–19.
- [110] O. Daramola, Y. Pan, P. Karpatis, and G. Sindre, “A comparative review of i\*-based and use case-based security modelling initiatives,” in *Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 1–12.
- [111] X. Franch, “Fostering the Adoption of i \* by Practitioners: Some Challenges and Research Directions,” in *Intentional Persp. on Inf. Systems Eng.*, S. Nurcan, C. Salinesi, C. Souveyet, and J. Ralyté, Eds. Springer, 2010, pp. 177–193, doi: 10.1007/978-3-642-12544-7\_10.
- [112] R. Matulevičius and P. Heymans, “Comparing Goal Modelling Languages: An Experiment,” in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, P. Sawyer, B. Paech, and P. Heymans, Eds. Springer Berlin Heidelberg, Jun. 2007, no. 4542, pp. 18–32, doi: 10.1007/978-3-540-73031-6\_2.
- [113] T. Li, J. Horkoff, and J. Mylopoulos, “Analyzing and Enforcing Security Mechanisms on Requirements Specifications.” Springer International Publishing, 2015, pp. 115–131, doi: 10.1007/978-3-319-16101-3\_8.

- [114] T. Li, E. Paja, J. Mylopoulos, J. Horkoff, and K. Beckers, “Security attack analysis using attack patterns,” in *Research Challenges in Information Science (RCIS), 2016 IEEE Tenth International Conference on*. IEEE, 2016, pp. 1–13.
- [115] R. Ali, R. Chitchyan, and P. Giorgini, “Context for goal-level product line derivation,” in *3rd International Workshop on Dynamic Software Product Lines (DSPL09)*, 2009.
- [116] R. Ali, F. Dalpiaz, and P. Giorgini, “Goal-based self-contextualization,” in *Proceedings of the Forum at the CAiSE 2009 Conference, Amsterdam, The Netherlands, 8-12 June 2009*, ser. CEUR Workshop Proceedings, E. S. K. Yu, J. Eder, and C. Rolland, Eds., vol. 453. CEUR-WS.org, 2009. [Online]. Available: <http://ceur-ws.org/Vol-453/paper07.pdf>
- [117] —, “Contextual Goal Models,” Technical report disi-10-020, University of Trento, Tech. Rep., 2010. [Online]. Available: [http://eprints.biblio.unitn.it/1820/1/Contextual\\_Goal\\_Model\\_Ali\\_et\\_al.pdf](http://eprints.biblio.unitn.it/1820/1/Contextual_Goal_Model_Ali_et_al.pdf)
- [118] —, “A goal-based framework for contextual requirements modeling and analysis,” *Requirements Engineering*, vol. 15, pp. 439–458, Nov. 2010, dOI: 10.1007/s00766-010-0110-z.
- [119] —, “Reasoning about Contextual Requirements for Mobile Information Systems: a Goal-based Approach,” Technical report disi-10-029, University of Trento, Tech. Rep., 2010.
- [120] —, “Reasoning with contextual requirements: Detecting inconsistency and conflicts,” *Information and Software Technology*, vol. 55, no. 1, pp. 35–57, Jan. 2013.
- [121] —, “Requirements-driven deployment,” *Software & Systems Modeling*, vol. 13, pp. 433–456, Feb. 2014, dOI: 10.1007/s10270-012-0255-y.

- [122] R. Ali, F. Dalpiaz, P. Giorgini, and V. E. S. Souza, “Requirements evolution: from assumptions to reality,” in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2011, pp. 372–382, doi: 10.1007/978-3-642-21759-3\_27.
- [123] R. Ali, A. Franzen, A. Griggio, and P. Giorgini, “Modeling and Analyzing Contextual Requirements,” Technical report disi-09-019, University of Trento, Tech. Rep., 2009.
- [124] R. Ali, “Modeling and reasoning about contextual requirements: Goal-based framework,” Ph.D. dissertation, University of Trento, 2010.
- [125] R. Ali, Y. Yu, R. Chitchyan, A. Nhlabatsi, and P. Giorgini, “Towards a unified framework for contextual variability in requirements,” in *Software Product Management (IWSPM), 2009 Third International Workshop on*. IEEE, 2009, pp. 31–34.
- [126] L. Baresi and L. Pasquale, “Live goals for adaptive service compositions,” in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2010, pp. 114–123.
- [127] —, “Adaptation Goals for Adaptive Service-Oriented Architectures,” in *Relating Software Requirements and Architectures*, P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrík, Eds. Springer Berlin Heidelberg, 2011, pp. 161–181, doi: 10.1007/978-3-642-21001-3\_10.
- [128] L. Baresi, L. Pasquale, and P. Spoletini, “Fuzzy goals for requirements-driven adaptation,” in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 125–134.
- [129] N. Bencomo, “Requirements for self-adaptation,” in *Generative and Transformational Techniques in Software Engineering IV, International Summer School, GTTSE 2011*,

- Braga, Portugal, July 3-9, 2011. Revised Papers*, 2011, pp. 271–296. [Online]. Available: [https://doi.org/10.1007/978-3-642-35992-7\\_7](https://doi.org/10.1007/978-3-642-35992-7_7)
- [130] N. Bencomo, K. Welsh, P. Sawyer, and J. Whittle, “Self-explanation in adaptive systems,” in *Engineering of Complex Computer Systems (ICECCS), 2012 17th Intl. Conference on*. IEEE, 2012, pp. 157–166.
- [131] K. Welsh, N. Bencomo, P. Sawyer, and J. Whittle, “Self-explanation in adaptive systems based on runtime goal-based models,” *Trans. Computational Collective Intelligence*, vol. 16, pp. 122–145, 2014. [Online]. Available: [https://doi.org/10.1007/978-3-662-44871-7\\_5](https://doi.org/10.1007/978-3-662-44871-7_5)
- [132] G. Chatzikonstantinou and K. Kontogiannis, “Model contextual variability for agents using goals and commitments,” in *Proceedings of the 6th International i\* Workshop 2013, Valencia, Spain, June 17-18, 2013*, 2013, pp. 103–108.
- [133] F. Guimaraes, G. Rodrigues, D. Batista, and R. Ali, “Pragmatic requirements for adaptive systems: A goal-driven modeling and analysis approach,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9381, pp. 50–64, 2015.
- [134] A. Lapouchnian and J. Mylopoulos, “Modeling Domain Variability in Requirements Engineering with Contexts.” Springer, 2009, pp. 115–130, doi: 10.1007/978-3-642-04840-1\_11.
- [135] —, “Capturing contextual variability in  $i^*$  models,” in *Proceedings of the 5<sup>th</sup> International i\* Workshop 2011, Trento, Italy, August 28-29, 2011*, ser. CEUR Workshop Proceedings, J. B. de Castro, X. Franch, J. Mylopoulos, and

- E. S. K. Yu, Eds., vol. 766. CEUR-WS.org, 2011, pp. 96–101. [Online]. Available: <http://ceur-ws.org/Vol-766/paper17.pdf>
- [136] A. Lapouchnian and E. S. K. Yu, “Exploring context sensing in the goal-driven design of business processes,” in *18th IEEE Conference on Business Informatics, CBI 2016, 29th August - 1st September 2016, Paris, France, Volume 1 - Conference Papers*, 2016, pp. 45–54. [Online]. Available: <https://doi.org/10.1109/CBI.2016.14>
- [137] A. Lapouchnian, “Exploiting Requirements Variability for Software Customization and Adaptation,” Ph.D. dissertation, University of Toronto, 2011.
- [138] Souza and Mylopoulos, “Monitoring and diagnosing malicious attacks with autonomic software,” in *International Conference on Conceptual Modeling*, vol. 5829 LNCS, 2009, pp. 84–98.
- [139] Y. Lei, K. Ben, and Z. He, “A Framework for Self-Adaptive Software Based on Extended Tropos Goal Model,” in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2015 7th International Conference on*, vol. 2. IEEE, 2015, pp. 533–536.
- [140] —, “A model driven agent-oriented self-adaptive software development method,” in *Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*. IEEE, 2015, pp. 2242–2246.
- [141] W. Liu and Z. Feng, “Context-based Requirement Modeling for Self-adaptive Service Software,” *Binary Information Press*, 2012. [Online]. Available: <https://www.techrepublic.com/resource-library/whitepapers/context-based-requirement-modeling-for-self-adaptive-service-software/>

- [142] ———, “Requirement uncertainty analysis for service-oriented self-adaptation software,” in *Network Computing and Information Security*. Springer, 2012, pp. 156–163, dOI: 10.1007/978-3-642-35211-9\_20.
- [143] W. Liu, C. W. He, and Z. W. Feng, “Requirement uncertainty modeling for service oriented self-adaptive software,” in *Advanced Materials Research*, vol. 433. Trans Tech Publ, 2012, pp. 4798–4801.
- [144] P. K. Murukannaiah and M. P. Singh, “Xipho: Extending Tropos to engineer context-aware personal agents,” in *International conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 309–316.
- [145] D. Amyot and G. Mussbacher, “Development of Telecommunications Standards and Services with the User Requirements Notation,” in *Workshop on ITU System Design Languages 2008*, 2008, pp. 15–16.
- [146] A. Pourshahid, L. Peyton, S. Ghanavati, D. Amyot, P. Chen, and M. Weiss, “Model-Based Validation of Business Processes,” *Business Process Mgmt.: Concepts, Tech., & Application*, pp. 165–183, 2012.
- [147] A. Pourshahid, “A Framework for Monitoring and Adapting Business Processes Using Aspect-Oriented URN,” Ph.D. dissertation, Université d’Ottawa/University of Ottawa, 2014.
- [148] N. A. Qureshi, I. J. Jureta, and A. Perini, “Towards a Requirements Modeling Language for Self-Adaptive Systems.” Springer Berlin Heidelberg, 2012, pp. 263–279, dOI: 10.1007/978-3-642-28714-5\_24.

- [149] N. A. Qureshi, I. Jureta, and A. Perini, “Adaptive RML: A Requirements Modeling Language for Self-Adaptive Systems,” Technical report, Tech. Rep., 2011.
- [150] N. A. Qureshi, “Requirements Engineering for Self-Adaptive Software: Bridging the Gap between Design-Time and Run-Time,” Ph.D. dissertation, University of Trento, 2011.
- [151] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, “System Identification for Adaptive Software Systems: A Requirements Engineering Perspective.” Springer Berlin Heidelberg, 2011, pp. 346–361, dOI: 10.1007/978-3-642-24606-7\_26.
- [152] V. E. Silva Souza, “Requirements-based software system adaptation,” Ph.D. dissertation, University of Trento, 2012.
- [153] J. Sun Kim, S. Park, and V. Sugumaran, “Contextual problem detection and management during software execution in complex environments,” *Industrial Mgmt. & Data Systems*, vol. 106, no. 4, pp. 540–561, 2006.
- [154] M. Vrbaski, G. Mussbacher, D. Petriu, and D. Amyot, “Goal Models As Run-time Entities in Context-aware Systems,” in *7th Workshop on Models@Run.Time*, ser. MRT ’12. New York, NY, USA: ACM, 2012, pp. 3–8.
- [155] M. Vrbaski, D. Petriu, and D. Amyot, “Tool support for combined rule-based and goal-based reasoning in Context-Aware systems,” in *20th IEEE Intl. Requirements Engineering Conf. (RE)*. IEEE, 2012, pp. 335–336.
- [156] M. Vrbaski, “Domain Independent Context Awareness Framework,” Ph.D. dissertation, Carleton University Ottawa, 2012.

- [157] Z. Xu, H. Zhao, and L. Liu, “User’s Requirements Driven Services Adaptation and Evolution,” in *Computer Software and Applications Conference Workshops (COMP-SACW), 2012 IEEE 36th Annual*. IEEE, 2012, pp. 13–19.
- [158] Z. Yang and Z. Jin, “Modeling and specifying parametric adaptation mechanism for self-adaptive systems,” in *Requirements Engineering*. Springer, 2014, pp. 105–119.
- [159] J. Stirna, J. Grabis, M. Henkel, and J. Zdravkovic, “Capability Driven Development – An Approach to Support Evolving Organizations.” Springer Berlin Heidelberg, 2012, pp. 117–131, doi: 10.1007/978-3-642-34549-4\_9.
- [160] S. Bērziša, G. Bravos, T. C. Gonzalez, U. Czubayko, S. España, J. Grabis, M. Henkel, L. Jokste, J. Kampars, H. Koç, J.-C. Kuhr, C. Llorca, P. Loucopoulos, R. J. Pascual, O. Pastor, K. Sandkuhl, H. Simic, J. Stirna, F. G. Valverde, and J. Zdravkovic, “Capability Driven Development: An Approach to Designing Digital Enterprises,” in *Business & Information Systems Engineering*. Springer Fachmedien Wiesbaden, Feb. 2015, vol. 57, pp. 15–25, doi: 10.1007/s12599-014-0362-0.
- [161] H. Koç, J. Kuhr, K. Sandkuhl, and F. Timm, “Capability-driven development - A novel approach to design enterprise capabilities,” in *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*, ser. Intelligent Systems Reference Library, E. El-Sheikh, A. Zimmermann, and L. C. Jain, Eds., 2016, vol. 111, pp. 151–177. [Online]. Available: [https://doi.org/10.1007/978-3-319-40564-3\\_9](https://doi.org/10.1007/978-3-319-40564-3_9)
- [162] J. Stirna, J. Zdravkovic, M. Henkel, P. Loucopoulos, and C. Stratigaki, “Modeling Organizational Capabilities on a Strategic Level,” in *IFIP Working Conference on The Practice of Enterprise Modeling*. Springer, 2016, pp. 257–271.



- [163] J. Stirna and J. Zdravkovic, “Supporting Perspectives of Business Capabilities by Enterprise Modeling, Context, and Patterns,” in *International Conference on Business Informatics Research*. Springer, 2016, pp. 262–277.
- [164] S. España, J. Grabis, M. Henkel, H. Koç, K. Sandkuhl, J. Stirna, and J. Zdravkovic, “Strategies for capability modelling: Analysis based on initial experiences,” in *Advanced Information Systems Engineering Workshops - CAiSE 2015 International Workshops, Stockholm, Sweden, June 8-9, 2015, Proceedings*, 2015, pp. 40–52.
- [165] C. Menghi, “Contextual, requirements driven, adaptive access control,” Ph.D. dissertation, Politecnico di Milano, 2012.
- [166] L. Pasquale, C. Menghi, M. Salehie, L. Cavallaro, I. Omoronya, and B. Nuseibeh, “Securitas: a tool for engineering adaptive security,” in *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE’12, Cary, NC, USA - November 11 - 16, 2012*, 2012, p. 19.
- [167] L. Dounas, R. Mazo, C. Salinesi, and O. El Beqqali, “Continuous Monitoring of Adaptive e-learning Systems Requirements,” in *12th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2015)*, 2015.
- [168] —, “Runtime Requirements Monitoring Framework for Adaptive e-Learning Systems,” in *International Conference on Software & Systems Engineering and their Applications (ICSSEA’15)*, 2015.
- [169] P. Akiki, “Engineering Adaptive Model-Driven User Interfaces for Enterprise Applications,” Ph.D. dissertation, The Open University, 2014.

- [170] X. Peng, S.-W. Lee, and W.-Y. Zhao, “Feature-Oriented Nonfunctional Requirement Analysis for Software Product Line,” *Journal of Computer Science and Technology*, vol. 24, pp. 319–338, Mar. 2009, doi: 10.1007/s11390-009-9227-2.
- [171] M. Morandini, “Goal-oriented development of self-adaptive systems,” Ph.D. dissertation, University of Trento, 2011.
- [172] T. Zhao, H. Zhao, W. Zhang, and Z. Jin, “User preference based autonomic generation of self-adaptive rules,” in *6th Asia-Pacific Symposium on Internetware on Internetware*. ACM, 2014, pp. 25–34.
- [173] A. Mello Ferreira, “Energy aware service based information systems,” Ph.D. dissertation, Politecnico di Milano, 2013.
- [174] I. Elgedawy, Z. Tari, and M. Winikoff, “Exact functional context matching for web services,” in *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*, 2004, pp. 143–152.
- [175] E. B. d. Santos, “Business process configuration with nfrs and context-awareness,” Ph.D. dissertation, Universidade Federal de Pernambuco, 2013.
- [176] E. Santos, J. Pimentel, D. Dermeval, J. Castro, and O. Pastor, “Using NFR and context to deal with adaptability in business process models,” in *2nd International Workshop on Requirements@Run.Time (RE@RunTime 2011), Trento, Italy, August 30, 2011*, 2011, pp. 43–50.
- [177] R. Ali, A. Griggio, A. Franzén, F. Dalpiaz, and P. Giorgini, “Optimizing monitoring requirements in self-adaptive systems,” in *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2012, pp. 362–377, doi: 10.1007/978-3-642-31072-0\_25.

- [178] R. Ali, C. Solis, I. Omoronyia, M. Salehie, and B. Nuseibeh, “Social adaptation at runtime,” in *International Conference on Evaluation of Novel Approaches to Software Engineering*. Springer, 2012, pp. 110–127.
- [179] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, “An Architecture for Requirements-Driven Self-reconfiguration.” Springer Berlin Heidelberg, 2009, pp. 246–260, doi: 10.1007/978-3-642-02144-2\_22.
- [180] D. F. Mendonça, G. N. Rodrigues, R. Ali, V. Alves, and L. Baresi, “GODA: A goal-oriented requirements engineering framework for runtime dependability analysis,” *Information and Software Technology*, vol. 80, pp. 245–264, 2016.
- [181] D. F. Mendonça, R. Ali, and G. N. Rodrigues, “Modelling and Analysing Contextual Failures for Dependability Requirements,” in *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 55–64.
- [182] D. F. Mendonça, “Dependability Verification for Contextual/Runtime Goal Modelling,” Ph.D. dissertation, Universidade de Brasília, 2015.
- [183] J. Pimentel, E. Santos, and J. Castro, “Conditions for ignoring failures based on a requirements model.” in *SEKE*, 2010, pp. 48–53.
- [184] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao, “Rationalism with a dose of empiricism: combining goal reasoning and case-based reasoning for self-adaptive software systems,” *Requirements Engineering*, vol. 20, pp. 233–252, Sep. 2015, doi: 10.1007/s00766-015-0227-1.
- [185] J. Vilela and J. Castro, “Modeling the monitoring and adaptation of context-sensitive systems,” in *Proceedings of the Eighth International i\*Workshop, iStar 2015, in*

- conjunction with the 23rd International Requirements Engineering Conference (RE 2015), Ottawa, Canada, August 24-25, 2015.*, 2015, pp. 55–60. [Online]. Available: <http://ceur-ws.org/Vol-1402/paper5.pdf>
- [186] J. Vilela, J. Castro, and J. Pimentel, “A systematic process for obtaining the behavior of context-sensitive systems,” *Journal of Software Engineering Research and Development*, vol. 4, p. 2, Dec. 2016, doi: 10.1186/s40411-016-0028-3.
- [187] J. Vilela, J. Castro, J. Pimentel, and P. Lima, “On the behavior of context-sensitive systems,” 2015. [Online]. Available: [http://wer.inf.puc-rio.br/WERpapers/artigos/artigos\\_WER15/WER15-vilela.pdf](http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER15/WER15-vilela.pdf)
- [188] J. Vilela, J. Castro, J. Pimentel, M. Soares, P. Cavalcanti, and M. Lucena, “Deriving the behavior of context-sensitive systems from contextual goal models,” in *30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1397–1400.
- [189] J. F. F. Vilela, “GO2s: a systematic process to derive the behavior of contextsensitive systems from requirements models,” Ph.D. dissertation, Universidade Federal de Pernambuco, 2015.
- [190] M. R. Almaliki, “Engineering an adaptive and socially-aware feedback acquisition.” Ph.D. dissertation, Bournemouth University, 2015.
- [191] I. Johari Shirazi, “Combining Business Intelligence, Indicators, and the User Requirements Notation for Performance Monitoring,” Ph.D. dissertation, Université d’Ottawa/University of Ottawa, 2012.
- [192] A. Knauss, “The Capture and Evolution of Contextual Requirements: The Case of Adaptive Systems,” Ph.D. dissertation, University of Victoria, 2015.

- [193] E. B. Zavala Rodríguez, “Dealing with uncertainty in contextual requirements at run-time: A proof of concept,” Ph.D. dissertation, Universitat Politècnica de Catalunya, 2015.
- [194] S. Nalchigar, E. Yu, and S. Easterbrook, “Towards Actionable Business Intelligence: Can System Dynamics Help?” Springer Berlin Heidelberg, 2014, pp. 246–260, dOI: 10.1007/978-3-662-45501-2\_18.
- [195] J. Zdravkovic, J. Stirna, M. Henkel, and J. Grabis, “Modeling business capabilities and context dependent delivery by cloud services,” in *International Conference on Advanced Information Systems Engineering*. Springer, 2013, pp. 369–383.
- [196] L. O. B. da Silva Santos, L. F. Pires, and M. van Sinderen, “A goal-based framework for dynamic service discovery and composition,” in *ACT4SOC 2008 - Proceedings of the 2nd International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing, Porto, Portugal, July 5-8, 2008*, M. van Sinderen, Ed. INSTICC Press, 2008, pp. 67–78.
- [197] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao, “Self-adaptation through incremental generative model transformations at runtime,” in *36th International Conference on Software Engineering*. ACM, 2014, pp. 676–687.
- [198] G. Neumann and M. Strembeck, “An approach to engineer and enforce context constraints in an RBAC environment,” in *eighth ACM symposium on Access control models and technologies*. ACM, 2003, pp. 65–79.
- [199] M. Rahimi, M. Mirakhorli, and J. Cleland-Huang, “Automated extraction and visualization of quality concerns from requirements specifications,” in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 2014, pp. 253–262.

- [200] U. Aßmann, S. Götz, J.-M. Jézéquel, B. Morin, and M. Trapp, “A Reference Architecture and Roadmap for Models@run.time Systems.” Springer International Publishing, 2014, pp. 1–18, dOI: 10.1007/978-3-319-08915-7\_1.
- [201] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, “Talos: an architecture for self-configuration,” Technical report disi-08-026, University of Trento, Tech. Rep., 2008.
- [202] A. Fayoumi, E. Kavakli, and P. Loucopoulos, “Towards a Unified Meta-model for Goal Oriented Modelling,” in *European, Mediterranean & Middle Eastern Conference on Information Systems 2015*, 2015.
- [203] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, “Requirements-aware systems: A research agenda for re for self-adaptive systems,” in *2010 18th IEEE International Requirements Engineering Conference*. IEEE, 2010, pp. 95–103.
- [204] K. Schmid, “Goal-Based Requirements Modelling as a Basis for Adaptivity to the Service Context,” *Context Awareness for Proactive Systems*, vol. 2006, p. 31, 2006.
- [205] H. Liu, “Integration of model driven engineering and ontology approaches for solving interoperability issues,” Ph.D. dissertation, Ecole Centrale de Lille, 2011.
- [206] L. Mei, “Cognitive Context Elicitation and modeling,” Ph.D. dissertation, University of Toronto, 2011.
- [207] U. Alegre, J. C. Augusto, and T. Clark, “Engineering context-aware systems and applications: A survey,” *Journal of Systems and Software*, vol. 117, pp. 55–83, Jul. 2016.
- [208] M. L. de Jesus Souza, A. R. Santos, and E. S. de Almeida, “Towards the selection of modeling techniques for dynamic software product lines,” in *Fifth International*

- Workshop on Product Line Approaches in Software Engineering*. IEEE Press, 2015, pp. 19–22.
- [209] M. L. de Jesus Souza, A. R. Santos, I. do Carmo Machado, E. S. de Almeida, and G. S. da Silva Gomes, “Evaluating variability modeling techniques for dynamic software product lines: A controlled experiment,” in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2016, Maringá, Brazil, September 19-20, 2016*, 2016, pp. 1–10. [Online]. Available: <https://doi.org/10.1109/SBCARS.2016.15>
- [210] G. Guedes, C. Silva, M. Soares, and J. Castro, “Variability Management in Dynamic Software Product Lines: A systematic mapping,” in *Components, Architectures and Reuse Software (SBCARS), 2015 IX Brazilian Symposium on*. IEEE, 2015, pp. 90–99.
- [211] L. Kolos-Mazuryk, P. Eck, and R. Wieringa, *A survey of requirements engineering methods for pervasive services*. Freeband A-MUSE Deliverable D5.7, 2006. [Online]. Available: <https://ris.utwente.nl/ws/portalfiles/portal/5128046/File-62328.pdf>
- [212] E. Paja, A. Maté, C. C. Woo, and J. Mylopoulos, “Can goal reasoning techniques be used for strategic decision-making?” pp. 530–543, 2016. [Online]. Available: [https://doi.org/10.1007/978-3-319-46397-1\\_41](https://doi.org/10.1007/978-3-319-46397-1_41)
- [213] M. Soares, J. Vilela, G. Guedes, C. Silva, and J. Castro, “Core Ontology to Aid the Goal Oriented Specification for Self-Adaptive Systems,” in *New Advances in Information Systems and Technologies*. Springer, 2016, pp. 609–618.
- [214] K. Petersen and C. Gencel, “Worldviews, research methods, and their relationship to validity in empirical software engineering research,” in *2013 Joint Conf. of the 23rd Intl. Workshop on Software Measurement (IWSM) and the 8th Intl. Conf. on Software*

- Process and Product Measurement*, ser. IWSM-MENSURA '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 81–89.
- [215] J. Maxwell, “Understanding and validity in qualitative research,” *Harvard educational review*, vol. 62, no. 3, pp. 279–301, 1992.
- [216] G. Poels, A. Maes, F. Gailly, and R. Paemeleire, *Measuring the Perceived Semantic Quality of Information Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 376–385. [Online]. Available: [http://dx.doi.org/10.1007/11568346\\_41](http://dx.doi.org/10.1007/11568346_41)
- [217] A. I. Anton, “Goal identification and refinement in the specification of software-based information systems,” Ph.D. dissertation, Atlanta, GA, USA, 1997, uMI Order No. GAX97-35409.
- [218] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, “GRAIL/KAOS: an environment for goal-driven requirements engineering,” in *Pulling Together, 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997.*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 612–613.
- [219] J. Mylopoulos, L. Chung, and E. Yu, “From object-oriented to goal-oriented requirements analysis,” *Commun. ACM*, vol. 42, no. 1, pp. 31–37, Jan. 1999.
- [220] M. Morandini, L. Penserini, and A. Perini, “Towards goal-oriented development of self-adaptive systems,” in *2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, ser. SEAMS '08. New York, NY, USA: ACM, 2008, pp. 9–16.



- [221] J. a. Pimentel, J. Castro, J. Mylopoulos, K. Angelopoulos, and V. E. S. Souza, “From requirements to statecharts via design refinement,” in *29th Annual ACM Symposium on Applied Computing*, ser. SAC '14. New York, NY, USA: ACM, 2014, pp. 995–1000.
- [222] K. Saller, M. Lochau, and I. Reimund, “Context-aware dspls: Model-based runtime adaptation for resource-constrained systems,” in *17th Intl. Software Product Line Conf. Co-located Workshops*, ser. SPLC '13 Workshops. New York, NY, USA: ACM, 2013, pp. 106–113.
- [223] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, “Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling,” in *18th IEEE International Requirements Engineering Conference*, ser. RE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 115–124.
- [224] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, “Feature-oriented domain analysis (foda) feasibility study,” Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.
- [225] Y. Liu, Y. Su, X. Yin, and G. Mussbacher, “Combined propagation-based reasoning with goal and feature models.” in *MoDRE*, A. Moreira, P. Sánchez, G. Mussbacher, and J. a. Araújo, Eds. IEEE, 2014, pp. 27–36. [Online]. Available: <http://dblp.uni-trier.de/db/conf/re/modre2014.html#LiuSYM14>
- [226] M. Acher, P. Collet, P. Lahire, and R. France, “FAMILIAR: A Domain-Specific Language for Large Scale Management of Feature Models,” *Science of Computer Programming*, vol. 78, no. 6, pp. 657 – 681, Jun. 2013. [Online]. Available: <https://hal.inria.fr/hal-00767175>

- [227] jUCMNav tool, “(version 7.0.0),” 2016, <http://jucmnav.softwareengineering.ca/ucm/bin/view/ProjetSEG/JUCMNavRelease700>.
- [228] D. C. Schmidt, “Model-driven engineering,” *IEEE Computer*, vol. 39, no. 2, pp. 41–47, February 2006.
- [229] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering - Foundations, Principles, and Techniques*. Springer, 2005.
- [230] E. W. Dijkstra, *A Discipline of Programming*. Prentice-Hall, 1976. [Online]. Available: <http://www.worldcat.org/oclc/01958445>
- [231] E. S. Agency, “Ariane 5, Flight 501 Failure,” Tech. Rep., 1996. [Online]. Available: <http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf>
- [232] N. Thimmegowda and J. Kienzle, “Visualization algorithms for feature models in concern-driven software development,” in *Companion Proceedings of the 14th International Conference on Modularity, MODULARITY 2015, Fort Collins, CO, USA, March 16 - 19, 2015*, 2015, pp. 39–42.
- [233] Aprajita, “Timedgrl: Specifying goal models over time,” Master’s thesis, Department of Electrical and Computer Engineering, McGill University, Canada, 2017.