2013

A framework to support the maintenance of formal goal models

Konstantin Hoesch-Klohe
University of Wollongong

2013

A framework to support the maintenance of formal goal models

Konstantin Hoesch-Klohe
University of Wollongong

University of Wollongong 

A FRAMEWORK TO SUPPORT THE MAINTENANCE OF FORMAL GOAL MODELS

A Thesis Submitted in Partial Fulfilment of
the Requirements for the Award of the Degree of

PhD in Computer Science and Software Engineering

from

UNIVERSITY OF WOLLONGONG

by

Konstantin Hoesch-Klohe

School of Computer Science and Software Engineering
Faculty of Informatics

2013

CERTIFICATION

I, Konstantin Hoesch-Klohe, declare that this thesis, submitted in partial fulfilment of the requirements for the award of PhD in Computer Science and Software Engineering, in the School of Computer Science and Software Engineering, Faculty of Informatics, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

(Signature Required)

Konstantin Hoesch-Klohe
30 August 2013

Dedicated to

my wife.

Table of Contents

List of Figures

ABSTRACT

A goal model is an important artifact in an organisational context. It encodes organisational intent and serves as a motivation for and justification of system requirements. However, the ever changing business environment requires organisations to constantly reposition and reinvent themselves. This involves adopting new goals or giving up goals that have become undesirable or infeasible to realize. In short, it requires the goal model to be maintained.

While there is a large body of literature on software maintenance, little has been done to support the maintenance of goal model artifacts. In particular, existing maintenance approaches ignore or do not sufficiently take into account the structure of a goal model. This can result in erroneous modifications. On the other hand, various issues exist when structured goals models, which are typically formalized as AND/OR graphs, need to be maintained. A maintenance approach that supports AND/OR graphs, but does not experience their deficiencies, is therefore desirable.

Where goal models are manually maintained, erroneous and sub-optimal modifications are often unavoidable, simply because of human cognitive limits on how much of the space of alternative modifications we can explore. Algorithmic maintenance functionality has the capability to explore this large space of alternatives for optimal modifications. Algorithmic maintenance functionality is therefore desirable. In particular, gaps have remained in leveraging the structure of a goal model to provide efficient procedures to compute optimal change options.

As a goal model justifies system requirements, it must ultimately be ensured that goals in the goal model remain fulfilled after a modification. This requires the ability to algorithmically reason about the fulfilment of goals with both functional and non-functional characteristics.

This thesis presents a framework that supports the algorithmic maintenance of formal and hierarchically structured goal models. In particular, a novel goal model formalization is introduced. It is based on AND/OR graphs, but overcomes their deficiencies in a maintenance setting. This thesis further demonstrates how a set of maintenance operators can be used to address a multitude of change requests and how the impact to the system requirements can be assessed. We then show how the key maintenance operator can be cast as a state space search problem and efficiently solved using A* search. Experiments with a prototype implementation demonstrate the feasibility of our approach.

KEYWORDS: Goal-oriented Requirements Engineering (GORE), Maintenance, Evolution, Inconsistency Management, Goal Model, AND/OR graph, A* search.

List of Publications

Over the course of my PhD studies, I (co-)authored the following publications (see list below). The material, presented in publications for which I am the first-author is completely my own work, guided by my supervisor Prof. Aditya Ghose and my co-supervisor Dr. Hoa Khanh Dam. Some of this material also appears in this thesis.

1. **Konstantin Hoesch-Klohe** and Aditya Ghose. Business Process Improvement in Abnoba. In *ICSOC 2010 International Workshops PAASC, WESOA, SEE, and SC-LOG.* Springer LNCS, 2010.

2. **Konstantin Hoesch-Klohe** and Aditya Ghose. Carbon-Aware Business Process Design in Abnoba. In *Proceedings of the 8th International Conference on Service Oriented Computing*, 2010.

3. **Konstantin Hoesch-Klohe** and Aditya Ghose. Towards Green Business Process Management. In *Proceedings of the 7th International Conference on Services Computing (Industry and Application Track)*, 2010.

4. **Konstantin Hoesch-Klohe** and Aditya Ghose. Making use of scenarios for environmentally aware system design. In *REFSQ 2012 Workshops*, 2012.

5. **Konstantin Hoesch-Klohe**, Aditya Ghose, and Hoa Khanh Dam. Maintaining motivation models (in bmm) in the context of a (wsdl-s) service landscape. In *Proceedings of the 10th International Conference on Service Oriented Computing*, 2012.

6. **Konstantin Hoesch-Klohe** and Aditya Ghose. Environmentally aware business process improvement in the enterprise context. In *Harnessing Green IT: Principles and Practices*, pages 265–281. Wiley, 2012.

7. Aditya Ghose, **Konstantin Hoesch-Klohe**, Lothar Hinsche, and Lam-Son Lê. Green business process management: A research agenda. *Australian Journal of Information Systems*, 16(2), 2009.

8. Aditya Ghose, Lam-Son Lê, **Konstantin Hoesch-Klohe**, and Evan Morrison. The business service representation language: a preliminary report. In *Service-Wave 2010 Workshops*, pages 145–152. Springer, 2011.

9. Lam-Son Lê, Aditya Ghose, Muralee Krishnan, Krishnajith Krishnankunju, and **Konstantin Hoesch-Klohe**. Correlating business objectives with services: An ontology-driven approach. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 306–313, 2011.

10. Constantin Houy, Markus Reiter, Peter Fettke, Peter Loos, **Konstantin Hoesch-Klohe**, and Aditya Ghose. Advancing business process technology for humanity: Opportunities and challenges of green bpm for sustainable business activities. In *Green Business Process Management*, pages 75–92. Springer, 2012.

11. Evan Morrison, Aditya Ghose, Hoa Khanh Dam, Kerry Hinge, and **Konstantin Hoesch-Klohe**. Strategic alignment of business processes. In *Service-Oriented Computing-ICSOC 2011 Workshops*, pages 9–21. Springer, 2012.

Chapter 1

Introduction

This chapter outlines the motivation for the problem that is addressed in this thesis, identifies a list of research questions, and outlines how these are addressed in the following chapters. An overview of the structure of this thesis is given at the end of the chapter.

1.1 Motivation

In today's fast changing world, the frequency with which an organisation must take into account changes - like new regulative frameworks such as the "Sarbanes Oxley Act" [162], or the development of new markets and technologies like "Cloud Computing" [160] - has increased rapidly. This forces organisations to constantly change, and in extreme cases to reinvent themselves. In a world where software systems underpin major parts of business operations (e.g. in human resources, controlling, accounting, production and distribution - to name some), this also increases the frequency with which new software functionality (or performance) is required to accommodate these changes.[1] As a result, the requirements that an organisation imposes upon its software systems are constantly changing and therefore have to be maintained.

The search for a requirements specification that satisfies all organisational needs/goals is known as the "Requirements Problem" [168]. It is a central concern in the requirements engineering process [168, 89], which encompasses the finding, recording and maintaining of system requirements as part of the software development process. [146] The instability of requirements, particularly at an *early stage* of the software devel-

[1] A manifestation of this trend is the ongoing transition to more flexible architectures like Service Oriented Architectures (SOA) [163].

opment process, is due to their "tentative" [82] nature, often caused by an increased level of incomplete and inconsistent knowledge. [82, 171] On the one hand, this has given rise to a large body of requirements engineering research, concerned with the detection, toleration and resolution of inconsistencies in requirements specifications (e.g. see [73, 137, 99, 13, 48, 125, 81, 82]). On the other hand, it highlighted that it is not sufficient to just focus on the requirements specification, but it is also necessary to pay closer attention to the "early-phase of requirements engineering" [165] (also referred to as Goal-Oriented Requirements Engineering (GORE)), which puts organisational intent/goals into the centre of attention. In other words, instead of focusing on what a software system should do, the emphasis is on organisational intent and the question "Why are we doing this?". Organisational intent is encoded in what is referred to as a goal model. The idea is that ultimately all artifacts generated in the course of a systems development project should be traceable back to (and justified by) goals in the goal model. [152] The goal model thereby acts as an additional layer of abstraction that is more stable and reluctant to change. [3, 152] As pointed out by Lamsweerde [152], a requirement, which denotes a particular way of achieving a goal, is more likely to change and realize that same goal, than the goal itself. However, often the goal model that supports and justifies a particular system is ignored after the development phase, or simply assumed to be invulnerable to change (as also pointed out in Ernst et al. [46]). This can have serious consequences. An outdated (or ignored) representation of stakeholder intention can easily lead to systems that do not fulfil their purpose. As we are moving towards an ever faster changing world, depending upon software systems, this problem is deemed to become more and more severe.

There is a growing body of literature on self adaptive systems (see [142, 156, 159, 14, 132, 147] for a representable subset), which allow systems to quickly adapt themselves to changes in the environment in a manner that ensures that the system maintains its ability to fulfil organisational goals. These techniques explicitly take into account the goal model, post the development phase. However, they all share the commonality that the way in which a system reacts to a given change is *explicitly* encoded in the goal model. Fundamentally, this requires the ability to *anticipate* changes. [44] However, not all changes can be anticipated. Inverardi and Mori [84] accordingly distinguish between "foreseen" and "unforeseen" changes. A shift in stakeholder intention, like adopting new goals or ceasing to support goals, may not be foreseen and it is this type of change that requires the maintenance of a goal model.

A goal model needs to be maintained when stakeholders change their mind, or goals

have become undesirable or infeasible to realize, e.g. goals might become too costly to realize or non-compliant with new regulations. There is little work concerned with the evolution/maintenance of goal models. Although, there is a large body of literature on software maintenance, which focuses on adapting, correcting and perfecting a software product [149], these activities are typically centred around the maintenance of code and architectural artifacts. [46] While there is some work that addresses the evolution of goal models (most notable is the work by Ernst and his co-authors [42, 44, 45], Ghose [64, 63], Zwhogi et al. [170], MacNish and Williams [113], Nuseibeh and his co-authors [126, 82] and Garces and his co-authors [52, 31]), gaps have remained in leveraging the structure of a goal model and providing efficient procedures to compute optimal change options.

1.2 Research Questions

We now summarize the most important gaps in the literature and formulate a set of research questions that will be addressed in this thesis. A more detailed and inclusive coverage of related work is given in Chapter 3 and at the end of subsequent chapters.

Existing maintenance frameworks (like Ghose [64, 63], Zwhogi et al. [170], MacNish and Williams [113], Nuseibeh and his co-authors [126, 82] and Garces and his co-authors [52, 31]) do not pay enough (or no) attention to the hierarchical *structure* of a goal model (a notable exception being the work by Ernst and his co-authors [42, 44, 45]). Some goals are adopted and pursued since their realization serves a greater goal. For example, a goal to "Have ice cream." may justify the adoption of a goal "Ice cream paid" to the goal model. In this example, the goal "Ice cream paid." does not have a standing for itself, but merely is accepted as a possible way of *refining* and eventually fulfilling the goal "Have ice cream.". Making this justification/refinement relationship between goals precise allows the hierarchical structuring of a goal model. Such information is particularly important when the goal model needs to be maintained. For example, in a situation where the goal to "Have ice cream" is given up, the goal "Ice cream paid" loses its justification and should be given up as well (unless it is justified by yet another goal). Failing to take into account this structure may result in an inaccurate modified goal model with superfluous goals.

Typically, goals in the goal model are refined using AND/OR graphs (e.g. see pop-

ular goal model notations and frameworks like i* [167], NFR [121] and KAOS [28]).[2] This has advantages beyond the maintenance process, e.g. it eases readability and allows exploration of alternative ways of fulfilling a goal. [152] A maintenance framework that is based on AND/OR graphs is therefore desirable. However, a goal model, formalized as an AND/OR graph, has deficiencies (particularly) during the maintenance process. For example, AND/OR graphs require that all root goals[3] are consistent. However, this is an unrealistic assumption, particularly in a multi-stakeholder environment, where stakeholder goals (or "views", as argued in the context of the ViewPoints framework [48]) may initially be conflicting/inconsistent[4]. We hence require the ability to tolerate inconsistencies. Additionally, AND/OR graphs do not naturally support the *reuse of goals*. In other words, goals, which are currently not feasible or desirable to realize, would have to be removed and not be retained in anticipation of future reuse. As the formal specification of goals requires expert knowledge and is known to be time consuming, it is crucial to avoid the loss of such an asset. [170] Furthermore, goal models, formalized as AND/OR graphs, cannot avoid situations in which a desired change to the goal model (e.g. adding a new goal) permits the *derivation* of goals that are undesired. In other words, while each individual goal may be desirable, conjointly the goals may permit the derivation of goals that are undesired or even non-compliant. The above elaboration can be summarized in the following questions.

Q1: *How can a maintenance framework that supports AND/OR graphs, but does not face their deficiencies during the maintenance process, be devised? In particular: (a) How can we represent and reason with inconsistent goals? (b) How can the justification relation between goals be taken into account when the model is maintained? (c) How can the reuse of goals be supported? (d) How can the reasoning with derived goals be supported?*

In formal goal models, goals are, in addition to their natural language description, specified in *formal logic*. This allows us to capture the goal's semantics and, besides others, identify inconsistencies in the goal model. A multitude of logics have been

[2] In many cases, the AND/OR graph itself is referred to as a goal model. [152]

[3] A root goal, in the context of an AND/OR graph, is a goal that is not a refinement of any other goal.

[4] We only consider goals which are specified in a formal logic and therefore "conflict" and "inconsistency" ultimately refer to logical inconsistency.

considered to formally specify goals, including real time temporal logic (e.g. in [30]), computational tree logic (e.g. in [61]), linear time temporal logic (e.g. in [49]), many sorted logic (e.g. in [64]) and propositional horn logic (e.g. in [87]). While some logics have computational advantages (e.g. a propositional logic offers efficient procedures for satisfiability checking [139]), others are more expressive at the expense of efficiency and decidability (e.g. real time temporal logic). Dependent on the particular application scenario, all these logics have their merits. However, existing work, notably the work by Ernst and his co-authors [42, 44, 45]), requires goals to be specified in a particular logic (e.g. as propositions as part of a propositional horn logic [42, 44, 45]). This restricts the application of these maintenance frameworks.

Often there are many options to modify the goal model to incorporate a given change request and the intent is to commit to the "optimal" one. We might interpret *optimality* in many different ways. One approach, motivated by the need to protect existing investments in a project, is to try and minimize deviation from the prior goal model (the approach we adopt). Minimal deviation or minimal change can follow various intuitions and ultimately a framework should be non-prescriptive about how minimal change is defined.

The manual maintenance of a goal model can be a costly and error-prone exercise. For example, we might overlook inconsistencies or commit to sub-optimal modifications (often simply because of human cognitive limits on how much of the space of alternative modifications we can explore). A goal maintenance framework must therefore also provide algorithmic support in exploring this large space of alternatives to identify optimal modifications that maintain the goal model's integrity. While some of the existing work is agnostic to the way minimal change is defined (e.g. Ghose [64, 63], Zwhogi et al. [170] and Ernst and his co-authors [42, 44, 45]), gaps have remained in leveraging the structure of a goal model to efficiently compute optimal change options (an exception being the work by Ernst and his co-authors [42, 44, 45]). This poses the following questions.

Q2: *How can we algorithmically support the maintenance of formal goal models? In particular: (a) What maintenance operators are required and how can they be defined in a modular manner that is agnostic to the language in which goals are formalized or in which minimal change is defined? (b) How can we use the structure of a goal model to efficiently compute optimal change options?*

We started this chapter by highlighting that one of the challenges organisations face in today's fast changing world is to ensure that the supporting software systems remain to satisfy organisational goals. In other words, given a set of modifications of the goal model, the challenge is to maintain answers to the requirements problem. This includes the identification of goals which are not realized by a requirement (or other description of functionality like a business process or service), or requirements which do not contribute to the realization of a goal. This requires the continuous re-assessment and -establishment of realization links between goals in the goal model and requirements part of the specification (or larger repository). This exercise is, if manually performed, in the best case cumbersome, and in the worst case erroneous. Automation, or at least partial automation of this process is therefore desirable. Existing maintenance frameworks only take into account the impact of changes at the goal level to the requirements specification to some extent (e.g. Ghose [64, 63] verifies whether changes are consistent with a set of system trajectories), or assume that the realization links are manually established (e.g. Ernst and his co-authors [42, 44, 45]).

Jureta and his co-authors [89] revisited the original definition of the requirements problem [168], arguing that, amongst others, an answer to the requirements problem must also take into account the partial fulfilment of goals which do not have a "clear-cut"[111] satisfaction criteria. Goals which do not have a clear-cut satisfaction criteria are referred to as "soft-goals"[121] and typically relate to non-functional (i.e. performance) aspects of the system. [121] For example, in the context of a normal software system the soft-goal "Minimize carbon dioxide emissions." relates to system performance and does not have a clear-cut satisfaction criteria, as it may always be possible to further decrease carbon dioxide emissions. We are hence looking for a description of system functionality that satisfies all stakeholder goals and can be argued to be as close as feasible to fulfilling all soft-goals. On the one hand, this requires the ability to estimate, or approximate, during design-time, the performance of a description of system functionality. On the other hand, this requires the ability to identify the "best" description of system functionality, i.e. the one that is closest to fulfilling all soft-goals, from some finite space of alternatives. This space of alternatives can be manually defined, or (more interestingly) algorithmically generated. This and the above challenge can be summarized via the following question.

Q3: *How can the exploration of answers to the functional as well as non-functional aspects of the requirements problem be supported to ensure that the system requirements*

accurately reflect changes in the goal model? In particular: (a) How can we assess the impact of changes to the goal model on a pool of service descriptions? (b) How can the non-functional performance of a design artifact (like a business process design) be estimated? (c) How can we algorithmically optimize the estimated non-functional performance of a business process design?

1.3 Research Outcomes

The primary outcome of this thesis is a framework to support the maintenance of formal goal models. In particular, we addressed the identified research questions, as follows.

Research Question Q1: Research question **Q1** was addressed by introducing a novel goal model formalization, which is based on AND/OR graphs, but overcomes their deficiencies in a maintenance setting. In this formalization, stakeholder intention is represented via a set of "solution graphs"[5]. This allows the representation of one or more AND/OR graphs (with distinct root goals) via a single set of solution graphs and hence overcomes the representational issues when stakeholders have distinct and inconsistent (root) goals. Furthermore, all goals in a solution graph correspond to a consistent subset of a (potentially inconsistent) goal repository (referred to as "goal library") and hence allow us to reason with inconsistent goals (addressing **Q1** (a)). We further distinguished between goals in the goal library, which are directly justified by stakeholder desire (we refer to these goals as "desired goals" - typically these goals are root goals) and goals that are dependent on other goals (we refer to these as dependent goals). This allows us to ensure that a solution graph does not contain any superfluous goals after a modification, i.e. either a goal is desired or it is dependent in which case it must contribute to the fulfilment of a desired goal (by being part of one of its refinements) (addressing **Q1** (b)). The goal library also ensures that a goal which is removed from a solution graph is not completely lost, but retained in the goal library (addressing **Q1** (c)). The formalization permits reasoning with "derived goals" by associating an entailment relation with each "level"[6] of a solution graph. This not only allows us to make statements about derived goals at each level in a solution graph, but also about derived goals from a set of solution graphs (addressing

[5] Roughly, a solution graph is an AND/OR graph without alternative refinements, i.e. an AND graph.

[6] For example, all root goals of a solution graph are on level 0.

Q1 (d)).

Research Question Q2: Research question **Q2** (a) was addressed via a family of maintenance operators, which were demonstrated to address a multitude of change requests. The maintenance operators were defined as abstract operator classes, which permits their instantiation for different intuitions of minimal change and languages in which goals are specified. A highlight of these operators is the "not entail assertion" operator. This operator leverages the entailment relation which is associated with each solution graph to identify a revised set of solution graphs for which a particular "state of affairs"[166] (i.e. goal) is not derivable. Such an operator is not only interesting in compliance scenarios (e.g. in situations where we want to ensure that non-compliant states of affairs are not actively pursued by being derivable from the goal model), but was also shown to implement other maintenance operators. Research question **Q2** (b) was addressed by demonstrating how the "not entail assertion" operator can be cast as a state space search problem and efficiently solved using "A* search" [72] - which makes particular use of the structure of solution graphs. An experimental evaluation of a prototype implementation suggested that the "not entail assertion" operator can be applied on industry size goal models. This confirms the feasibility of our algorithmic maintenance approach.

Research Question Q3: Research question **Q3** (a) was addressed in our earlier work [80]. In this work, we showed how services, described in a semantic service description standard like WSDL-S [22] can be represented via their formal post-conditions as part of a goal model. In particular, we demonstrated how the "not entail assertion" operator can be used to address changes at the goal level and how the impact on the underlying service landscape can be assessed - this involves answering which goals in the goal model are unsatisfied and which services in the service catalogue are not contributing to the satisfaction of a desired goal. To address research question **Q3** (b), we then showed how our earlier work on (what we refer to as) "Green Business Process Management" [79, 78, 77] can be used to provide an answer to the non-functional part of the requirements problem. In particular, in [79] we show that the non-functional performance (we focused on carbon dioxide emissions) of a design artifact like a business process can be estimated by correlating activities of the process design with resources of a resource model. In [78], we continued to show how the performance values of multiple heterogeneous measures can be accumu-

lated through a business process design to obtain cumulative performance values of the design. The ability to estimate the non-functional performance of a design artifact enables the application of algorithmic improvement machinery. Research question **Q3** (c) is addressed in [77], where we proposed a process improvement procedure that leverages a library of process fragments to find business process re-designs (by swapping fragments), which remain to fulfil correlated stakeholder goals but have a "better" non-functional performance. This allows us to make statements about the fulfilment of (non-functional) soft-goals. A soft-goal (e.g. "minimize carbon dioxide emission") is satisficed by a business process design, if the process improvement procedure cannot find another process design with a better carbon dioxide emission performance.

1.4 Thesis Structure

The remainder of thesis is structured as follows.

- **Chapter 2** provides the reader with necessary preliminaries and is split into two parts. The first part covers formal preliminaries and clarifies terminology in the areas of graph theory, logic, non-monotonic reasoning and belief revision. The second part summarizes relevant requirements engineering concepts and terminology.

- **Chapter 3** summarizes related work and highlights gaps left by these approaches. In particular, the first part discusses existing approaches for representing and reasoning with inconsistent goals/requirements. The second part focuses on the maintenance aspects of these approaches. The chapter is concluded with a summary of the identified gaps.

- **Chapter 4** starts by highlighting issues in AND/OR graph based goal model formalizations when they need to evolve. This is followed by an alternative goal model formalization, which overcomes these deficiencies. The chapter ends with a comparison of our proposed formalization with existing work.

- **Chapter 5** first motivates various sensible intuitions of minimal change before introducing a family of goal model maintenance operator classes. Each maintenance operator is motivated via a set of normative properties and its application is illustrated. A comparison with existing work is given at the end of the chapter.

- **Chapter 6** is split into two main parts. The first part presents an application of our maintenance framework in the context of a service landscape [80] and discusses its usefulness in exploring answers to the functional part of the requirements problem. The second part summarizes our earlier work on "Green Business Process Management" [79, 78, 77, 80] and discusses how it can be used in the context of the proposed maintenance framework to support the exploration of answers to the non-functional part of the requirements problem.
- **Chapter 7** shows how the "not entail assertion" operator can be implemented via A* search.
- **Chapter 8** presents and discusses the results of an experimental evaluation of a prototype implementation.
- **Chapter 9** concludes this thesis and outlines future directions of research.

Chapter 2

Preliminaries & Background

This chapter is separated into two parts. The first part is intended as an introduction/refreshment of computer science techniques and terminology that will be heavily used throughout this thesis. In particular, the basics of graph theory, logic and non-monotonic formalism are recapitulated. Extra attention is given to the latter, as these formalisms are the basis of many approaches that support inconsistency management and evolution. The second part of this chapter, is intended as an introduction to Goal-Oriented Requirements Engineering (GORE). In particular, we summarize the representation of goal models as AND/OR graphs (at the same time introducing the running example), recapitulate the requirements problem as well as popular GORE frameworks.

2.1 Formal Preliminaries

2.1.1 Graphs

In this thesis, we will be using graphs to represent a goal model and hence it is useful to clarify some of the concepts and terminologies. The following paragraph summarizes the introduction to graphs given in [161].

A "*graph*" is a set of objects and relations. The objects are commonly referred to as "nodes" or "vertices", while the relations are referred to as "edges". A graph is said to be "labelled" if vertices and/or edges are associated with a label. A graph in which all edges relate to exactly two vertices is referred to as a "binary graph". The majority of graphs that are encountered in practice are binary graphs. The more general case of a graph with n-arry relations is referred to as a hyper-graph. The edges

of a graph may be directed or undirected, and the corresponding graph is referred to as a directed- or an undirected graph respectively. For an edge that points from vertex v to vertex v', we say that v is the source and v' is the target of the edge. A graph is said to be "cycle free" if there does not exists a sequence of edges that connect a sequence of vertices, such that the "start"- and "end" vertex are the same. A graph that is cycle free and where all vertices are connected is called a "tree". A disjoint set of trees is referred to as a "forest". A tree where the edges are directed is referred to as a "directed tree". In a directed tree, a vertex is referred to as a "root" if there is no directed edge that points toward it. Correspondingly, a vertex is referred to as a "leaf" if there is no directed edge that points from it. In a directed binary graph, a set of vertices is referred to as the "children" or "sub-vertices" of another distinct vertex (which is referred to as a "parent" vertex) if, for each child vertex, there exists an edge that points from the parent to the child.

2.1.2 Logic

In this thesis, we use Logic to represent domain knowledge and goals in the goal model. Consequently, we will make use of logic related concepts and terminology quite frequently. The following section is intended as a brief introduction to Logic and provides a summary of the more detailed introduction given in Russell and Norvig [139].

A knowledge representation language is defined by its syntax and semantics. While the syntax describes what constitutes a sentence in the formal language, the semantics describe the facts in the world to which the sentence relates. This mapping between sentences in the language and facts in the world is also referred to as interpretation of the sentence. Knowledge representation languages (or formal languages) allow the representation of facts in the world via sentences in the language.

As facts do follow from other facts in the world, in a formal language one sentence may follow from another sentence. This relation is referred to as "entailment" (denoted by $\vDash$), i.e. given the sentences A and B, then $A \vDash B$, if B follows from A. Inference procedures implement the entailment relation by generating new sentences from existing sentences following a set of rules. They are used to either establish that the relation $A \vDash B$ does hold, or to identify sentences B from A for which the entailment relation holds. Given an inference procedure i, we use $A \vdash_i B$ to denote that the procedure i can generate the sentences B from A (the procedure i is called sound if

whenever $A \vdash_i B$ then $A \models B$ also holds[1]). A formal language associated with a proof procedure is referred to as a logic. Given a set of sentences T, we use the operation $Cn(T)$ (referred to as the closure under logical consequence of a set of sentences T) to denote every possible sentence that can be generated from T. Observe, if $B \in Cn(A)$, then $A \vdash B$ and consequently $A \models B$. In the following, for ease of elaboration, we will use these notations interchangeably.

A set of sentences is said to be consistent or satisfiable if and only if there exists an interpretation for which it is true. A sentence is inconsistent or unsatisfiable if it is not satisfiable. The symbol $\perp$ is a constant and is interpreted as *False*. Any set of sentences containing the sentence $\perp$ is necessarily unsatisfiable. Classical logics like Propositional- and First-Order logic (a good introduction to these logics can be found in Russell and Norvig [139]) obey "ex falso quodlibet", according to which any sentence (including $\perp$) can be derived from a contradiction. In other words, if $\perp$ can be proven from A then A must be inconsistent. In the following, we use $\perp \notin Cn(A)$ (or $A \not\models \perp$) and $\perp \in Cn(A)$ (or $A \models \perp$) to denote that the sentences A are satisfiable or unsatisfiable, respectively. The principle of explosion trivializes a logic theory such that no sensible reasoning can be performed. There are various ways in which inconsistent theories can be managed, including paraconsistent logics which do not obey the principle of explosion or non-monotonic reasoning with consistent subsets. We will have a closer look at non-monotonic reasoning in the next section, and will see applications of it and paraconsistent logic in managing and evolving inconsistent requirements specifications in Chapter 3.

2.1.3 Non-Monotonic Logic & Formalisms

Non-Monotonic Logics are formal systems that allow conclusions to be inferred defeasibly. In other words, defeasibly inferred conclusions (via *defeasible inference*) can be withdrawn as additional information becomes available. This is in contrast to classical logics like Propositional or First-Order Logic, where inferred conclusions from a set of premises are deductively valid and therefore not withdrawn, i.e. the set of conclusions is monotonically increasing (i.e. never decreasing), as opposed to the non-monotonic character of defeasibly inferred conclusions in a Non-Monotonic Logic. [4]

Defeasible inference is part of daily human reasoning, where knowledge about the world is inherently uncertain and incomplete (the latter follows from the former), but

[1] In the following, we only consider the case of sound inference procedures.

we nevertheless draw useful conclusions. For example, consider a rule of defeasible character, which states that "if X is a bird, then in most cases X can fly", given the knowledge that Tweety is a bird, we can defeasibly infer that Tweety can fly.[2] Although we cannot with certainty conclude that Tweety can fly, such a conclusion is nevertheless useful in daily reasoning. However, defeasible conclusions may have to be retracted in the presence of new information, e.g. when we learn that Tweety has a broken wing.

2.1.3.1 Default Logic

One of the pioneers of non-monotonic formalisms is R. Reiter with his seminal work on *Default Logic*[134]. Default logic distinguishes between facts (i.e. knowledge that we can be certain of) and default rules (i.e. rules of thumb that allow the inference of plausible but not necessarily true conclusions [7]). A default rule takes the following form:

$$\frac{\alpha : \beta}{\gamma}$$

In this default rule, α (called prerequisite), β (called justification) and γ (called conclusion) denote well formed formulas of a formal language (e.g. a first-order language, as used by Reiter in his original paper [134]). The default rule itself can be understood as a meta-rule over this language and interpreted as follows: "if α and it is consistent to assume β then it can be concluded that γ". Following the above example, the sentence "If X is a bird and it is consistent to assume that X can fly, then conclude that X can fly" could be captured by the following default rule (schema[3]):

$$\frac{Bird(X) : Flies(X)}{Flies(X)}$$

A default theory is a pair $\langle W, D \rangle$, where W is a set of closed well formed formulas and D a set of default rules. Given the default $\delta_1 = \frac{Bird(Tweety):Flies(Tweety)}{Flies(Tweety)}$, the following default theory captures the above example:

$$W = \{Bird(Tweety)\},\ D = \{\delta_1\}$$

[2] The original example of Tweety the bird is due to Reiter [134] and has been widely used in the literature on non-monotonic reasoning.

[3] Formulas in a default rule have to be closed.

The semantics of Default Logic are given by extensions which represent consistent possible world views (theories) of the underlying default theory. [9] Extensions are obtained by applying default rules. A default rule can be applied if its prerequisite is derivable from the theory and its justification is consistent with the theory. The application of a default rule results in the addition of its conclusion to the theory. An extension is a maximal set of defaults that may be applied. For example, for the given default theory above, all defaults can be applied and the default theory has one extension (i.e. a singleton set of extensions E):

$$W = \{Bird(Tweety)\},\ D = \{\delta_1\}$$
$$E = \{\{Bird(Tweety), Flies(Tweety)\}\}$$

Now, let us assume we know that Tweety is a baby bird ($Baby(Tweety)$). The default rule is that if Tweety is a baby bird and it is consistent to assume that he cannot fly, then we can conclude that he cannot fly $\delta_2 = \frac{Baby(Tweety):\neg Flies(Tweety)}{\neg Flies(Tweety)}$. This gives us the following default theory:

$$W = \{Bird(Tweety), Baby(Tweety)\},\ D = \{\delta_1, \delta_2\}$$
$$E = \{\{Bird(Tweety), Flies(Tweety)\}, \{Bird(Tweety), \neg Flies(Tweety)\}\}$$

The above default theory has two extensions, since default δ_1 and δ_2 cannot both be applied. This is due to the fact that applying the default δ_1 (alternatively δ_2) implies that $Flies(Tweety)$ (alternatively $\neg Flies(Tweety)$) is part of the theory, which in turn does not permit the application of δ_2 (alternatively δ_1), since its justification cannot be consistently assumed. Defaults can disable, as well as enable, each other, which results in a complex interplay and default extension being non-constructive. Reiter [134], handles this by defining extensions of a default theory as fixpoints of an operator Γ.

Default Extension (due to Reiter [134], taken from [19])
Let (W, D) be a default theory and S a set of formulas. $\Gamma(S)$ is the smallest set such that (1) W is a subset of $\Gamma(S)$, (2) $\Gamma(S)$ is deductively closed, (3) if $\frac{\alpha:\beta}{\gamma} \in D$, $\alpha \in \Gamma(S)$, and $\neg B \notin S$, then $C \in \Gamma(S)$. E is an extension of (D, W) if and only if $\Gamma(E) = E$.

Skeptical and Credulous Reasoning: Default Logic (and Non-Monotonic Logics in general) may have more than one extension (i.e. permit distinct defeasible conclusions). This begs the question: which extension (defeasible conclusion) is used

for reasoning? There are two basic ways to handle this. We can reason in a "skeptical" or "credulous" manner. In skeptical reasoning, a sentence is derivable if it is derivable from all extensions (i.e. from all defeasible conclusions). Note, this implies that the sentence is consistent with any other sentence. In credulous reasoning, a sentence is derivable if there exists at least one default extension from which it is derivable (i.e. there exists at least one defeasible conclusion). [4]

Restricted Default theories: Default theories with (unrestricted) default rules lack desired properties like a constructive definition of an extension, semi-monotonicity (adding a new default to the default theory should extend existing extensions or result in more extensions), or the guaranteed existence of an extension (to name some). [34] As has been shown (e.g. in Delgrande and Jackson [34]), these properties can be obtained by restricting default rules. The literature considers various restrictions: default rules without pre-requisites (called a *pre-requisite free defaults*); default rules with a single justification and equivalent conclusion (called a *normal defaults*); default rules where the justification entails the conclusions (called *semi-normal defaults*). Normal defaults that are prerequisite free are referred to as *supernormal default*. A more detailed discussion of restricted default rules is given in Delgrande and Jackson [34].

THEORIST: A practical default reasoning system which is restricted to prerequisite free semi-normal defaults is THEORIST [131]. A THEORIST specification consists of facts (F), hypotheses (H) and constraints (C). While facts are (open) formulas which are necessarily true, hypotheses are (open) formulas which are tentatively true (more precisely hypotheses resemble pre-requisite free semi-normal defaults). This allows default reasoning by identifying extensions (called "maximal scenarios"). A maximal scenario contains all facts, as well as hypotheses that are consistent with the facts. Constraints are (closed) formulas and restrict the scenarios of a THEORIST specification to the ones that are consistent with the constraints. To sum up, a maximal scenario of a THEORIST specification $\langle F, H, C\rangle$ is a set $H' \subseteq H$, such that $F \cup H' \cup C$ is satisfiable. [131] Maximal scenarios are the basis for (skeptical or credulous) non-monotonic reasoning.

2.1.3.2 Truth Maintenance Systems

Truth Maintenance Systems (TMS) were pioneered by Doyle [35] to support more efficient reasoning in the context of incomplete knowledge by explicitly recording and maintaining the reasons/justifications for a belief. A Truth Maintenance System (TMS) records and maintains a dependency graph of sentences which is linked to

their justifications. TMS are agnostic to the actual inference engine used to draw inferences and establish the network (e.g. in Ernst's thesis [44] the requirements engineer plays the role of the inference engine).

In a Justification-Based Truth Maintenance System (JTMS) [35], each justification has an "IN" and "OUT" list of sentences. A sentence is supported by a justification (i.e. the justification is pointing to the sentence in the dependency network) if all sentences in the "IN" list are held and no sentence in the "OUT" list. *Sentences* are either facts or assumptions and can be true or false. True facts are known as premises, false facts as contradictions[35]. Each sentence is labelled "IN" or "OUT". A sentence is labelled "IN" if it is a premise, or an assumption assumed to be true, or there exists a justification that supports it. Given a current set of assumptions (and facts), the TMS allows exploration of the consequences, i.e. an "IN" or "OUT" labelling of nodes. A JTMS permits non-monotonic reasoning since a currently justified sentence may lose its justification either directly (i.e. when we stop to assume the truth of the sentence), or indirectly (e.g. when the label of sentences in the "IN" and "OUT" list of the respective justification of the sentence changes). A disadvantage of JTMS is the need to relabel nodes in the case that an assumption changes, which makes it computationally expensive to reason over alternative sets of assumptions.

Assumption Based Truth Maintenance Systems (ATMS), proposed by de Kleer [32], are an extension of JTMS and allow reasoning over alternative sets of assumptions (referred to as environments). In an ATMS, each sentence in the dependency graph is labelled with a set of environments in which it holds (i.e. is justified by). A key characteristic of an ATMS is that it associates every sentence with the minimal set of environments from which it is derivable. This is captured by the label of the sentence. Depending on the label, a sentence is either a premise and universally true (i.e. part of all contexts), a contradiction and universally false (i.e. not part of any context), or an assumption (part of at least one context) [33]. We can encode the premise that Tweety is a Bird and a Baby by creating the following two sentences, which have as a context an empty set (i.e. they are part of any context):

$$\langle Bird(Tweety), \{\{\}\}\rangle$$
$$\langle Baby(Tweety), \{\{\}\}\rangle$$

In an ATMS, justifications are horn rules that relate a set of sentences to a single sentence denoting that the latter is derivable from the former. The assumption that birds usually fly and that baby birds usually do not fly can be encoded via the following two justifications:

$$Bird(Tweety) \Rightarrow Flies(Tweety).$$
$$Baby(Tweety) \Rightarrow NotFlies(Tweety).$$

Note, ATMS do not have an explicit notion of negation and therefore treat a proposition $Flies(Tweety)$ and its negation $\neg Flies(Tweety)$ as two distinct sentences (to avoid any confusion, we have used $NotFlies(Tweety)$). Given the above, the sentences $Flies(Tweety)$ and $NotFlies(Tweety)$ would be encoded as assumptions with the following minimal environments associated to them:

$$\langle Flies(Tweety), \{\{Bird(Tweety)\}\}\rangle$$
$$\langle NotFlies(Tweety), \{\{Baby(Tweety)\}\}\rangle$$

Given a consistent environment (i.e. a set of consistent assumptions), the *context* refers to the set of all sentences derivable from the justifications and the given environment. In the above example, there exists one single context (which corresponds to an extension in Reiter's default logic [134]):

$$\{Bird(Tweety), Baby(Tweety), Flies(Tweety), NotFlies(Tweety)\}$$

Now let us encode that $Flies(Tweety)$ and $NotFlies(Tweety)$ are two contradictory sentences. Such information is provided by the inference engine and captured by a justification, which links the two sentences to the contradiction:

$$Flies(Tweety), NotFlies(Tweety) \Rightarrow \bot$$

This results in the two contexts below:

$$\{Bird(Tweety), Baby(Tweety), Flies(Tweety)\}$$
$$\{Bird(Tweety), Baby(Tweety), NotFlies(Tweety)\}$$

It is easy to see that the ATMS indirectly computes the context of each sentence by computing the sentence's environments.[33] In other words, a sentence is part of a context if the assumptions of any of its environments are a superset of the assumptions of the context.

2.1.3.3 Belief Revision

Belief revision describes the process by which a rational agent changes his beliefs when confronted with new information. The most dominant framework to describe this process is the AGM paradigm [1] (named after its founding fathers Alchourrón, Gärdenfors, and Makinson). The AGM paradigm views the change of belief as a function between two sets of beliefs and a list of postulates (known as the AGM postulates) that characterize this function. [129]

In the AGM paradigm, the beliefs of an agent are represented as a theory, i.e. a set of sentences in a formal language closed under logical deduction[4]. This set of sentences is called a *belief set*. The AGM paradigm distinguishes between three types of change: belief expansion (the addition of new beliefs to the belief set), belief contraction (the removal of beliefs from the belief set) and belief revision (the consistent addition of a belief to the belief set). In the AGM paradigm, each change is a function between belief sets, where all three types of change are characterized by a set of postulates. In the following, we summarize the AGM postulates for the expansion, contraction and revision functions.

Belief Expansion: In belief expansion, a new belief (and its consequences) is added to the belief set without removing any beliefs from the set (this permits the belief set to become inconsistent). Given an initial belief set K and a sentence A, the expansion of K by A is denoted by K_A^+ and must satisfy the following properties (taken from [53]).

- (K^+1) K_A^+ is a belief set.
- (K^+2) $A \in K_A^+$.
- (K^+3) $K \subseteq K_A^+$.
- (K^+4) if $A \in K$, then $K_A^+ = K$.
- (K^+5) if $K \subseteq H$, then $K_A^+ \subseteq H_A^+$.
- (K^+6) For all belief sets K and all sentences A, K_A^+ is the smallest belief set that satisfies (K^+1)-(K^+5).

The first two postulates state that the expansion of a belief set K by a sentence A must result in a belief set that contains A, i.e. the expansion must be successful. The

[4] A set of sentences is closed under logical deduction if any sentence that is derivable from the set is an element of the set.

third postulate captures the idea that we want to keep as much as possible of our beliefs after the addition of new beliefs. In other words, we want to avoid any unnecessary loss of information. This criteria is also referred to as *informational economy* and is central to the AGM paradigm. This is aligned with the idea behind the fourth postulate, which states that adding A to K where $A \in K$ should have no effect on K, i.e. the change operation should be vacuous. The fifth postulate states that expansion should be monotonic. In other words, if a belief set H is derivable from a belief set K and both sets are expanded by the sentence A, then K_A^+ should remain derivable from H_A^+. The last postulate captures the idea that the expansion of K should not result in new beliefs that have no connection with A. Note , if $\neg A \in K$, then K_A^+ is inconsistent (i.e. $\perp \in K_A^+$). In other words, expansion permits the belief set to become inconsistent[5].

Contraction: In belief contraction, a belief is given up by removing it (and beliefs that have it as a consequence) from the belief set. Given an initial belief set K and a sentence A, the contraction of A from K is denoted by K_A^- and must satisfy the following basic[6] properties (taken from [knowledge in flux]).

(K^-1) K_A^- is a belief set.

(K^-2) $K_A^- \subseteq K$.

(K^-3) if $A \notin K$, then $K_A^- = K$.

(K^-4) if not $\nvdash A$, then $A \notin K_A^-$.

(K^-5) if $A \in K$, then $(K_A^-)K_A^+ \subseteq K$.

(K^-6) if $\nvdash A \leftrightarrow B$, then $K_A^- = K_B^-$.

As with expansion, the contraction of a sentence from the belief set should result in a belief set. Furthermore, giving up a belief, i.e. contracting a sentence from the belief set, should not result in new beliefs to be derivable. Note that the principle of this second postulate does not hold in the context of a default logic (as well as other Non-Monotonic Logics), where the contraction of a belief may permit the application of a default rule (that otherwise would not have been applicable), which in turn may allow additional conclusions to be derived. Following informational economy, the third postulate suggests that a belief set that already does not include a sentence A should

[5] Recall that everything is derivable from an inconsistent belief set and therefore such a set must be a superset of any consistent set of beliefs.

[6] Two additional postulates are defined that deal with the connection between K_A^- and $K_{A \wedge B}^-$. For our purposes, the basic postulates are sufficient.

not be affected by K^-_A. The contraction of a sentence is successful if it is not part of the resulting belief set (unless A is a tautology, i.e. always true). The fifth postulate also follows the principle of information economy and states that contracting a belief A from a belief set and then expanding the resulting set with A must correspond to the original belief set (before A was contracted), i.e. all sentences should be recovered. This postulate generated a lot of discussion and criticism. For example, Ghose [65] argues that for a default logic, contracting a belief may permit the application of previously non-applicable defaults where the conclusion of the default rule may exclude the addition of the previously contracted belief - i.e. the original belief of the agent cannot completely be recovered.

Revision: In belief revision, a (consistent) belief is added to a belief set such that the resulting belief set is consistent. We use $K_\perp$ to explicitly refer to an inconsistent set of beliefs. Given an initial belief set K and a sentence A, the revision of K by a sentence A is denoted by K^*_A and must satisfy the following basic set of postulates.

(K^*1) K^*_A is a belief set.

(K^*2) $A \in K^*_A$.

(K^*3) $K^*_A \subseteq K^+_A$.

(K^*4) if $\neg A \notin K$, then $K^+_A \subseteq K^*_A$.

(K^*5) $K^*_A = K_\perp$ if and only if $\nvdash \neg A$.

(K^*6) if $\nvdash A \leftrightarrow B$, then $K^*_A = K^*_B$.

The first two postulates state that the revision of a belief set by a sentence A should result in a belief set that includes the sentence (i.e. the revision should be successful). The third postulate suggests that the revision of a belief set by a sentence A should not include beliefs that would not also be included when expanding the belief set by A. Postulate four takes into account the case where a sentence A can be consistently added, in which case revision becomes a special case of expansions. The fifth postulate states that the resulting belief set after revising by A is assumed to be consistent, unless $\neg A$ is a tautology. The last postulate requires that the revision of a belief set by logically equivalent sentences A and B has the same effect.

Relevant Observations and Results In the following, we briefly summarize some relevant observations that have been made about the expansion, contraction and revision functions.

Revision is non-monotonic: The revision of a belief set K is non-monotonic, since the addition of a new belief may require the removal of existing beliefs to guarantee that the resulting belief set is consistent. This is different to the expansion of a belief set, which monotonically increases.

Levi identity: The revision function is related to contraction and expansion. Formally $K^*_A = (K^-_{\neg A})^+_A$, i.e. revising a knowledge base K with A is given by the contraction of $\neg A$ from K followed by the expansion of K by A. This result was established by Levi and Harper [104] and is referred to as the "Levi Identify" (it also shown that the contraction function relates to a revision function).

Minimal change: Given the contraction of a sentence $\neg A$ from a belief set, it may not be sufficient to just remove $\neg A$, since it may be concluded from other sentences and therefore these would have to be removed as well. Since there may be more than one distinct set of sentences from which A can be concluded, there are multiple options to contract $\neg A$ from the belief set. Information economy dictates that the agent should retain as much as possible of the original beliefs, i.e. be interested in maximal consistent belief subsets that do not contain $\neg A$. However, there may be more than one such maximal consistent belief subset. Alchourrón et al. [1] show various ways in which a contraction function can be constructed to handle this. One way is to merge all maximal consistent subsets via set intersection (this is called "full meet contraction"). Another way is to deploy a selection function which would select the "best" belief subset (this is called "maxi-choice contraction"). While the former generally returns belief sets that are too "small" the latter returns belief sets that are too "big". In partial meet contraction, a selection function is used to select the "most" preferred belief subsets and to merge those selected sets via set intersection. Alternatively, beliefs may be ranked according to their usefulness (i.e. entrenchment) to the agent, with the idea being that the agent wishes to retain more entrenched beliefs over less entrenched ones. Alchourrón et al. [1] show that for both intuitions a contraction (and correspondingly a revision) function can be constructed, with the same results being obtained.

Belief sets vs. Belief bases: Belief sets are logically closed and therefore no distinction can be made between sentences that have a standing of their own [71] and sentences which are merely derivable (the distinction between assumptions and justified beliefs in TMS takes this explicitly into account). It has been argued that, in many cases, this representation is too idealized. [122] Representing an agent's beliefs via a belief base [122], which is not required to be logically closed, addresses this. It

has been shown that two logically equivalent representations of an agent's beliefs, as a belief base and belief set, may behave different in the context of change (e.g. the recovery postulate does not hold for partial meet contraction [105]).

Belief revision vs. belief update Katsuno and Mendelzon [92] take into account the distinction between revising (as considered in the AGM framework) and updating an agent's beliefs. While belief revision assumes a static world (where an agent's beliefs have to be revised due to incomplete or faulty information about that static world), belief update takes into account a dynamic world (where an agent's beliefs have to be updated due to changes in the world). Katsuno and Mendelzon [92] motivated a modified version of the AGM postulates for updating a belief base, which generated intensive debate about the formal difference between the two. For our purposes, it is sufficient to know that there may be distinct reasons why an agent's beliefs change and that it has been argued that belief update and revision (under certain circumstances) are "much the same" [129].

Revision & Default Theories An agent has to handle incomplete knowledge about the world.[7] In belief revision, an agent handles incomplete knowledge by revising its current knowledge to incorporate new information. Non-monotonic formalisms (e.g. default logic) handle incomplete knowledge by allowing an agent to tentatively reason with its beliefs and omit them in the face of new knowledge (i.e. by enabling and disabling default rules). There are two main differences: (1) The AGM framework for belief revision forces us to commit to a single revised theory (i.e. a single view of the world), while non-monotonic formalisms like default logic permit reasoning with multiple extensions (i.e. multiple distinct views of the world). (2) In the AGM framework, the revision of a theory may require the permanent removal of beliefs from an agent's belief set to consistently incorporate new beliefs, while non-monotonic formalisms like default logic allow an agent to retain the belief in anticipation of future situations when the inconsistency goes away. As argued in [65], the latter is following the principle of information economy more consequently.

There has been some work on belief revision in the context of default theory (e.g. [19], [65]). In [65], Ghose (and others) shows how to revise PJ-default theories (default theories restricted to pre-requisite-free and semi-normal defaults). More precisely, the beliefs of an agent are not represented by a single theory but by a set of theories. Correspondingly, a belief revision function is defined as a mapping between two *sets*

[7] Note that an agent's knowledge may be incomplete with reference to a static world, as well as incomplete with reference to the changes in a dynamic world.

of theories, where each theory of the set returned by the function must incorporate the new knowledge (following the principles of the AGM success postulate).

2.2 Goal-Oriented Requirements Engineering

Requirements Engineering is the process of finding and maintaining an implementable description of a system (referred to as requirements *specification*) that satisfies the stakeholders. [146] Traditional requirements engineering efforts were primarily concerned with answering what functionality a system should provide, captured via a set of *requirements*. [165] *Goal-Oriented Requirements Engineering* (GORE) takes one step back and focuses on why the system is needed and how the stakeholders interests and concerns might be addressed. It is therefore commonly referred to as "early-phase requirements engineering" [165]. GORE centres around stakeholder goals and the research emphasis is on how goals can be modelled and reasoned with to support requirements elaboration, verification and validation as well as conflict management. A good overview of research efforts in these other areas can be found in Lamsweerde [152]. In this thesis, the emphasis is on how representations of stakeholder goals are maintained, which requires us to summarize some of the key concepts and terminology used in goal modelling and reasoning.

2.2.1 Goal Models

The heart of GORE is the *goal model* in which stakeholder goals are expressed, refined and eventually related to and realized by a system design artifact. We now give an example of a goal model, expressed in the popular i* notation [165], which will serve as a running example throughout this thesis.

Example. *We use a running example based on the well known Teleservices and Remote Medical Care System (TRMCS) case study*[8]*. Following the current trend to transition patients from hospital- to home care, the TRMCS system is used to provide and guarantee medical assistance to at-home or mobile users. This requires the coordination of health care professionals, amongst others, to assist patients in need of medical help, but also to protect patients medical health data. Figure 2.1 shows a goal model expressed in the popular i* notation that contains some of the goals that may arise in such a setting. In Figure 2.1, goals are denoted by green coloured circles.*

[8] We use the description given in: `www.ics.uci.edu/~irus/iwssd/case-study.pdf`

Note, the white coloured circles attached to each goal are not part of the i notation, but are associated to each goal as an identifier and used for convenience throughout the rest of the paper.*

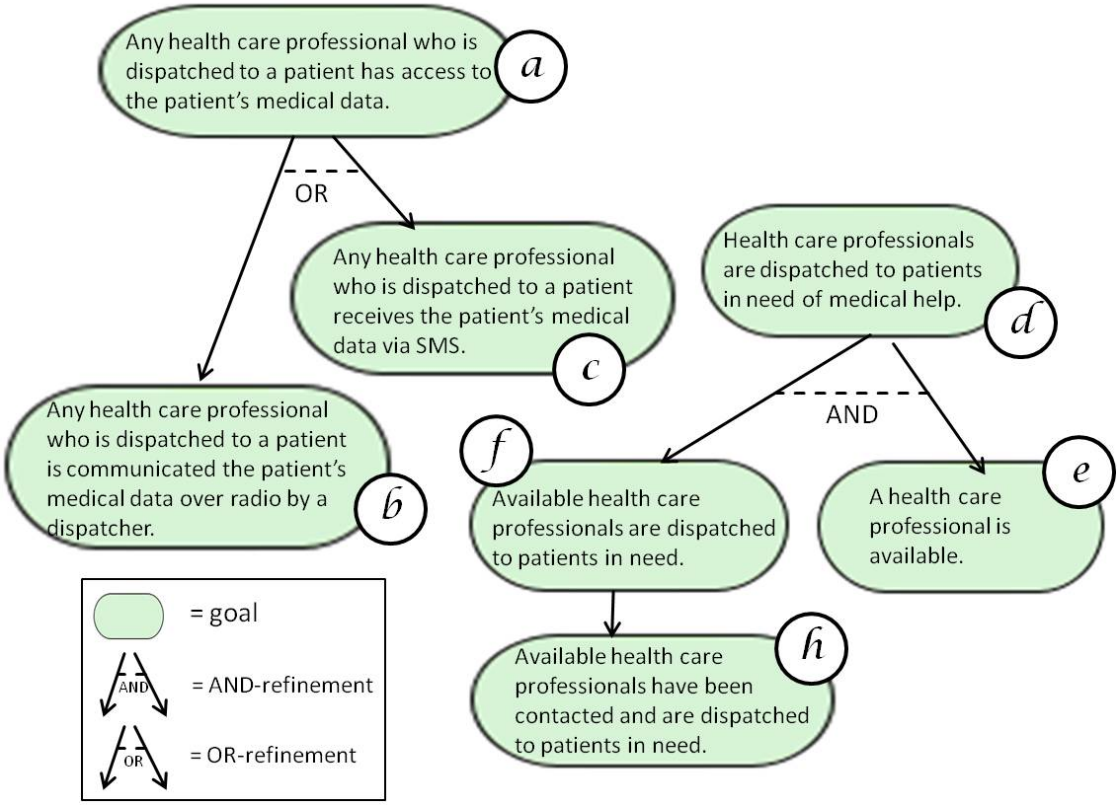

Figure 2.1: Example of a goal model in the i* notation.

The central element of any goal model is a goal. A *goal* is an "optative statement" [168] and describes desired properties of the environment, i.e. how the stakeholders would like the environment to be. [168] In other words, a goal denotes a desired "*state of affairs*"[166]. Typically, one distinguishes between *functional goals* and *non-functional goals*, where the former refers to functional aspects of a system and the latter to quality aspects of it, i.e. how well a system does something.[121] Goals which do not have a clear-cut satisfaction criterion are often of non-functional character and referred to as *soft-goals*.[121].[9] A detailed overview of the role of non-functional requirements in software engineering can be found in Chung et al. [23]. In Dardenne et al. [28], goals are also classified into "*achievement goals*"[28] and "*maintenance*

[9] The goal model shown in Figure 2.1 does not contain any soft-goals.

goals"[28], where the former requires that a property is eventually satisfied, while the latter requires that a property remains satisfied.

Goals in a goal model may range from high-level strategic concerns to lower level technical concerns, where the former is refined via "AND-refinement links"[152] and "OR-refinement links"[152] to the latter.[152] A refinement of a goal via an AND-refinement link denotes that all sub-goals (the goals the link is pointing to) must be satisfied in order for the parent goal (the link is pointing from) to be satisfied. A goal refinement of type "OR" denotes that at least one of the sub-goals must be satisfied for the parent goal to be satisfied. [152]

Example. *In Figure 2.1, the goals* ***a****,* ***d*** *are root goals. Goal* ***a*** *is linked via a refinement link of type "OR" to the goals* ***b*** *and* ***c****, denoting that either goal* ***b*** *or* ***c*** *must be satisfied for goal* ***a*** *to be considered as satisfied. Conversely, in Figure 2.1, the goal* ***d*** *is linked via a refinement link of type "AND" to the goals* ***f*** *and* ***e****, denoting that both goal* ***b*** *and* ***c*** *must be satisfied for goal* ***d*** *to be considered as satisfied.*

Goal models share many principles with AND/OR graphs, which are commonly used to formalize goal models (in many cases the AND/OR graph itself is referred to as goal model). [152] AND/OR graphs were first introduced by Nilsson [124] as a problem reduction technique in the area of Artificial Intelligence.[152] In Nilsson's AND/OR graph, formalization nodes are labelled as "AND-node" or "OR-node". A goal for which all sub-nodes are labelled as AND-nodes is satisfied if all sub-goals are satisfied. A goal for which all sub-nodes are labelled as OR-nodes is satisfied if one of the sub-nodes can be satisfied. Observe, nodes are labelled according to their incoming edges (where edges point from the parent node to its sub-node).

Example. *Figure 2.2 shows an AND/OR graph representation of the goal model shown in Figure 2.1. Nodes in the graph denote goals, while edges denote refinement links between goals. Node* ***f*** *and* ***e*** *are "AND" labelled, denoting that both nodes must be satisfied for their parent nodes* ***d*** *to be satisfied. Nodes* ***b*** *and* ***c*** *are "OR" labelled, denoting that at least one of the nodes must be satisfied for their parent node* ***a*** *to be satisfied. The goal* ***h*** *is the only sub-goal of* ***f*** *and therefore could be labelled "AND" or "OR".*

Goals are refined up to a point where they can be correlated to an implementable system requirement, i.e. description of system behaviour denoted by an artifact like a business process. A goal that is not further refined will be referred to as a leaf

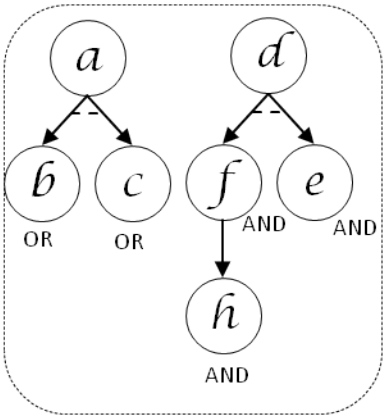

Figure 2.2: Visualization of the goal model show in Figure 2.1 as AND/OR graph.

goal. We say that a (leaf) goal that can be correlated to an artifact is realized by the artifact. A goal which is realized is also considered as *satisfied*. We are deliberately vague about what it means to correlate an artifact to a goal (i.e. to realize a goal), as this will be explored in more detail in Chapter 6. Typically, a goal model permits more than one option to refine and eventually satisfy stakeholder root goals.

Example. *For example, the goal model in Figure 2.1 has the root goals* $\boldsymbol{a}$ *and* $\boldsymbol{d}$*, which are satisfied if the (leaf) goals* $\{\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{e}\}$ *or* $\{\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{e}\}$ *are satisfied. For ease of elaboration, we assume that all (leaf) goals in the goal model are satisfied.*

As pointed out by Lamsweerde [152] a goal model is an important artifact in the system development process. On the one hand, goal models allow the identification of irrelevant system behaviours, i.e. descriptions of system behaviour that cannot be correlated to a (leaf) goal and hence do not contribute to the satisfaction of stakeholder root goals. On the other hand, goal models allow highlighting of missing system behaviour, i.e. (leaf) goals that are not satisfied. This is not only important during system development, but also during system maintenance, as it must be continuously ensured that the system provides the required functionality (and performance) to the stakeholders.

2.2.2 The Requirements Problem

The requirements problem was first formalized by Zave and Jackson [168] to determine when requirements engineering is successfully completed. In a nutshell, the requirements problem is the search for a requirements specification (i.e. a description of

system behaviour) that satisfies all properties the stakeholders desire of the environment. Formally, given a set of high-level requirements **R** (i.e. goals), a description of domain assumptions and constraints **K** and a requirements specification **S**, Zave and Jackson [168] state the requirements problem as follows:

$$\mathbf{K}, \mathbf{S} \vDash \mathbf{R}.$$

The requirements problem is successfully answered after a specification **S** has been found that (together with the domain knowledge **K**[10]) satisfies the entailment relation ($\vDash$).

More then 10 years later, Jureta and his co-authors [89] took into account the efforts that had been made in GORE over the past years and revisited Zave and Jackson's formalization [168] of the requirements problem. The authors point out that the original definition of the requirements problem by Zave and Jackson [168]: (1) does not consider partial fulfilment of goals (e.g. soft-goals); (2) does not allow one specification to be "better" than others; and (3) does not discriminate between "must have" (i.e. compulsory-) and "nice to have" (i.e. optional-) goals. In response to this criticism, Jureta et al. [89] re-formulated the requirements problem as the search for the most preferred specification that satisfies (as close as is feasible) all compulsory- and as many optional goals as possible. The need to distinguish between mandatory and optional goals as well as stakeholder preference has also previously been considered in Zowhi et. al [170] and Ghose [64] (in their work, mandatory goals are referred to as necessary goals and optional goals as tentative goals). The essence of the requirements problem, as described in Jureta et al. [89], can be formulated as follows

Given domain knowledge and assumption $\mathcal{KB}$, compulsory goals $\boldsymbol{G}_C$ and optional goals $\boldsymbol{G}_O$ (of functional and non-functional character), as well as compulsory and optional soft-goals $\boldsymbol{Q}_C$ and $\boldsymbol{Q}_O$, then the problem is to find a specification $\boldsymbol{S}$, such that:

(1) $\boldsymbol{KB}, \boldsymbol{S} \vDash \boldsymbol{G}_C, \boldsymbol{G}'_O$, *where* $\boldsymbol{G}'_O$ *is a maximal subset of* $\boldsymbol{G}_O$ *such that* $\boldsymbol{KB}, \boldsymbol{S}, \boldsymbol{G}_C, \boldsymbol{G}'_O$ *is consistent.*

(2) $\boldsymbol{S}$ *satisfices* $\boldsymbol{Q}_C, \boldsymbol{Q}'_O$, *where* $\boldsymbol{Q}'_O$ *is a maximal subset of* $\boldsymbol{Q}_O$ *that is conjointly satisficable with* $\boldsymbol{Q}_C$ *and*

[10] The role of domain knowledge is described in more detail in Chapter 4. For now, it is sufficient to know that **K** is used to encode the knowledge (or assumption) that goal **a** is in fact satisfied, if the goal **b** or **c** are satisfied.

(3) $\boldsymbol{G}'_O$ *and* $\boldsymbol{Q}'_O$ *are the most preferred set of optional goals and soft-goals.*

For ease of exposition, we have modified the original definition provided by Jureta et al. [89] and only consider a special case where domain knowledge and the specification are of non-tentative nature (and therefore did not use a non-monotonic inference relation to relate the domain knowledge and specification to goals). Furthermore, we do not explicitly distinguish between goals of functional and non-functional character.

It is important to note that finding an answer to the requirement problem is not only a task pre system deployment, but also a task during system maintenance, since stakeholder goals and domain assumptions and constraints may change over time, invalidating the existing answers. [42]

2.2.3 Representing and Reasoning with Goal Models

A goal model and its goals can be specified at different levels of formal precision. In semi-formal approaches like the "Non-Functional Requirements" (NFR) framework [121], goals and relationships (e.g. refinement, or operationalization/realization relationships) are stated in terms of their types and attributes, but are described (informally) in natural language. On the other hand, in formal approaches like "Knowledge Acquisition in autOmated Specification" (KAOS) [30], goals are, in addition to their natural language description, expressed in a *formal language* - in KAOS goals are expressed in real time temporal logic. In formal approaches, the satisfaction of a goal by a set of sub-goals and eventually the specification can be formally verified, often in an automated manner by deploying model checking and theorem proofing technology. However, not all goals have a clear-cut satisfaction criterion. This is a situation where semi-formal approaches like NFR [121] (and its extensions [66, 143, 43]), which permit qualitative reasoning over goal satisfaction, have merits.

2.2.3.1 Semi-Formal Approaches

Non-functional requirements often do not have a clear-cut satisfaction criteria. [121] For example, the requirement "The system must be secure." does not have a clear-cut satisfaction criteria, since it may always be possible to make a system more secure. Recall that requirements that do not permit clear establishment of when they are satisfied are referred to as soft-goals. This poses a challenge, since the aim of requirements engineering is to find and maintain a specification that keeps stakeholder goals satisfied. This problem was addressed in Mylopoulos et al. [121] as part of the

(Non-Functional Requirements) NFR framework. The authors extend the AND/OR graph formalization of a goal model with additional "*contribution links*"[121] that state how positively- or negatively a goal impacts another. A soft-goal is said to be "satisficed"[121] (as opposed to satisfied) by a selected set of requirements, if satisfying all requirements of the selected set positively impacts the respective soft-goal(s).

Giorgini et al. [66] extended the NFR framework by providing a qualitative as well as quantitative framework for axiomatizing goal model elements. The heart of this framework is a label propagation algorithm (for the qualitative as well as quantitative axiomatization). The algorithm, given an initial assignment to (leaf) goals (e.g. a goal label may state that a goal is "Fully Satisfied", "Partially Satisfied", "Partially Denied", or "Fully Denied"), propagates these labels (according to a set of propagation rules) through the goal model to identify how positively- or negatively all other goals (including soft-goals) are satisfied. In other words, given an initial label assignment to leaf goals, the algorithm finds all consequent label assignments of goals in the goal model.

Sebastiani et al. [143] extended this work with a technique for determining a required label assignment to leaf goals as well as the cost-optimal label assignment to leaf goals, such that the desired status of root goals is satisfied.

Ernst et al. [43] further extends the work of Sebastiani et al. [143] by allowing the ability to distinguish between mandatory goals (i.e. goals that must be satisfied) and optional goals (i.e. goals that are nice to satisfy) as well as noting a preference amongst them. The authors show how local search can be used to find a label assignment to leaf goals that satisfy all mandatory goals and as many optional goals as possible, and is the most preferred with respect to the user defined preference relation. In other words, the technique proposed in Ernst. et al. [43] can be used to find answers to the requirements problem as defined in Jureta et al. [89].

A limitation of the above line of work is that relationships between goals cannot be verified, which permits situations where a relation may have been incorrectly established. This is a particular problem in a dynamic setting where domain knowledge may evolve and render a previously correctly established relationship as incorrect. Frameworks in which goals are formally specified (in addition to their natural language description) allow verification of the relation after a change.

2.2.3.2 Formal Approaches

The formal representation of requirements (goals) and domain knowledge has various advantages, including clear and unambiguous semantics, enforcement of formal rigour, and the use of formal reasoning techniques like consistency checking or question-answering. [68]

Early Work: The advantages of a formal representation have long been acknowledged by the requirements engineering community. Early proposals for formal requirements frameworks include RML (Requirements Modelling Language) [69], its successor TELOS [120], ERAE (Entity, Relation, Attribute, Event) [36, 37] and Cake [135]. A good overview and discussion on this early work can be found in Greenspan et a. [68].

KAOS: A formal approach that has gained popularity both in industry and academia is KAOS, proposed in Dardenne et al. [29].

KAOS, like RML [69] and TELOS [120], is a two layered modelling language. The "outer" level allows the declaration of entities and relationships of domain level concepts like goals, constraints, agents, and events, which are linked via various types of relationships, e.g. And-/Or-refinements and operationalization/realization links. The "inner" level consists of formal assertions, expressed in typed temporal First-Order logic, that can be attached to domain specific concepts, i.e. the formal assertions are used to formulate the desired state of affairs denoted by a particular goal, or the properties that do hold after performing an action.

A key contribution of the KAOS framework is its emphasis on stakeholder goals and their refinement up to an operational level to eventually acquire requirements that allows their realization. On the one hand, this refinement process is supported by an acquisition methodology. [29] On the other hand, the KAOS framework formally defines what constitutes a "correct" refinement (relation) of a goal to its sub-goals. This, amongst others, permits the verification of goal refinements [30]. The definition of a correct goal refinement, which is given below, is adopted from Darimont and Lamsweerde [30].

Definition 1. *(Correct Goal Refinement) Given a domain knowledge base $\mathcal{KB}$, a refinement of a goal g by a set of sub-goals T is correct if and only if (1) $\bot \notin Cn(\mathcal{KB} \cup T)$ and (2) $g \in Cn(\mathcal{KB} \cup T)$ and there does not exists a set $T' \subset T$ such that $g \in Cn(\mathcal{KB} \cup T')$.*

A refinement of a goal by a set of sub-goals is correct if the sub-goals are consistent and together minimally entail the parent goal. A set of sub-goals *minimally* entails a goal if taking out any goal of the set of sub-goals results in a remainder that does not

entail the parent goal.

To sum up, the NFR frameworks (and subsequent work) use qualitative reasoning techniques (i.e. label propagation) to establish goal satisfaction of goals expressed in natural language. The KAOS framework uses formal reasoning techniques like model checking and theorem proving to establish goal satisfaction of goals that are declaratively expressed in a formal language. A more detailed review of some of the key results around the KAOS and NFR framework is given by Lamsweerde [98].

2.3 Summary

The intention of this chapter was to ease the exposition of our work as well as the discussion of related work, by providing the reader with an understanding of relevant techniques and terminology.

- The first part of this chapter (Chapter 2.1) recapitulated formal preliminaries. A brief introduction to *graph theory* and *formal logic* was given, followed by a more detailed summary of *non-monotonic reasoning* approaches. In particular, a brief introduction to *default logic*, *truth maintenance systems* and (AGM) *belief revision*, was given.

- The second part of this chapter (Chapter 2.2) gave an introduction to (Goal-Oriented) Requirements Engineering. In particular, the representation of a *goal model* as an AND/OR graph was described and its advantages in addressing the *requirements problem* summarized. We further recapitulated key aspects of popular approaches for *semi- and formal reasoning with goal models*.

Chapter 3

Related Work

The main purpose of this chapter is to provide an overview of existing work and highlight gaps that are addressed in this thesis. Our primary focus is on frameworks that are concerned with the maintenance of formally specified goal/requirement[1] models. The ability to manage inconsistencies is an essential concern when goals are formally specified. Inconsistency management is also important when the goal model is to be modified, as the addition of a new goal may be inconsistent with existing goals/requirements. There is a large body of requirements engineering literature that is concerned with managing inconsistencies (e.g. [73, 137, 99, 13, 48, 125, 81, 82]). The first part of this chapter provides an overview of this body of literature. At the same time, the goal/requirements representation that is used in various maintenance frameworks is summarized and gaps are highlighted. Chapter 3.2 then focuses on the maintenance of these representations by summarizing and highlighting gaps in existing maintenance frameworks. Chapter 3.3 concludes this chapter by summarizing the key gaps and deficiencies. Note, a comparison with our framework is given at the end of subsequent chapters.

3.1 Inconsistency Management

Managing inconsistencies is an essential part of requirements engineering. [82] Inconsistencies between goals/requirements may be the result of "tentative requirements" [82], incomplete domain knowledge [171], conflicting stakeholder views (in a multi-

[1] In this chapter, we often do not distinguish between the maintenance of goals and the maintenance of requirements, since most existing maintenance approaches are agnostic to the ontological distinction between the two, i.e. they are both treated as formal assertions.

stakeholder setting) [48], or a shift in stakeholder motivation [170], which may require the addition of (potentially inconsistent) goals/requirements.

In the body of literature on managing inconsistent goals/requirements, one can separate between techniques to *identify and resolve inconsistencies* as well as techniques to *tolerate inconsistencies*.

3.1.1 Inconsistency Identification and Resolution

Early work on managing inconsistent goals/requirements is primarily concerned with the identification, classification and resolution of inconsistencies.

Heitmeyer et al. [73] showed how consistency checking techniques can be used to detect errors in formal requirements specifications. Robinson [136] proposed the use of formal negotiation techniques and corresponding tool support to resolve conflicts between different stakeholder. In Robinson and Volkov [137], object restructuring and object conditionalizing are discussed as conflict resolution techniques. A detailed study and classification of various types of inconsistencies that can arise during requirements engineering, as well as their resolution, is given by Lamsweerde et al. [99]. The authors particularly focus on the detection and resolution of a weaker notion of conflict, referred to as "goal divergence" [99], which corresponds to a situation where a particular combination of circumstances (called "boundary condition" [99]) can be found that make (otherwise consistent) goals inconsistent. This work was further extended in Lamsweerde and Letier [154], where the authors present formal techniques for systematically generating obstacles from goal formulations and domain properties, and techniques to resolve these.

3.1.2 Tolerating Inconsistencies

Inconsistencies need to be detected and eventually resolved. However, the immediate resolution of inconsistencies may, amongst others, result in premature and sub-optimal commitments [48] and therefore various techniques for tolerating inconsistent requirements/goals have been studied and applied. Among the first to address the need of flexible inconsistency handling was Balzer [13] by introducing the concept of a "pollution marker". A pollution marker is a relation defined over the data that is found to be violating a consistency criteria and added in the case where inconsistencies have been detected and retrieved when the identified inconsistency could be resolved. A body of literature followed, using consistency rules to relate potentially inconsistent

viewpoints (as in the Viewpoints framework [48, 125, 81]), leveraging paraconsistent representation and reasoning techniques (like Quasi-boolean multi-valued logics [39] and Quasi-classical logic [82]), as well as non-monotonic formalism (as in Techne by Jureta et al. [87], the REFORM framework by Ghose [64, 63], and the work by Zwhogi et al. [170])

3.1.2.1 Managing inconsistencies in the Viewpoints framework

The need to tolerate inconsistencies (in addition to the identification and eventual resolution) was popularized by work on the Viewpoints framework (see [48, 125, 81] for a representative subset of the early contributions to the framework). The Viewpoints framework acknowledges that the development of any larger system involves multiple stakeholders with different and often conflicting views (i.e. "viewpoints"), different methodologies as well as specification languages. The consolidation of multiple conflicting viewpoints in (potentially different specification language) forces stakeholders (besides others) to prematurely commit to certain views and thereby avoid the exploration of potentially useful alternatives. The Viewpoints framework avoids this by allowing us to maintain separate viewpoints, which are loosely related via consistency rules. Two viewpoints are inconsistent if a consistency rule between them is violated. Consistency rules are associated with action rules, which are activated when a consistency rule is violated, and they specify how to act according to the respective inconsistency. Inconsistencies can tolerated and addressed at a desired point in time by delaying the application of the activated actions. Formally, the specification information of each viewpoint is translated into formal logic via a set of specification language specific rewrite rules. Consistency rules are captured in the same logic language, such that the ViewPoints, which are required to be internally consistent, can be checked against the latter for logical consistency (e.g. by calling a theorem prover). The associated actions are captured in an action-based meta language, which is based on linear temporal logic and allows us to take into account how the inconsistencies have been handled in the past (e.g. it avoids that the rule is fired when the user wants to ignore it).

An application of the Viewpoints framework in the context of an evolving specification has been considered in Easterbrook and Nuseibeh [40] where the authors seek to avoid the loss of inconsistency resolutions due to evolutionary changes by keeping track of unresolved as well as resolved inconsistencies. The merging of inconsistent and incomplete views using typed graphs and category theory has been considered in

[140, 141]. More recently, Eramo et al. [41] show how multi-view specifications can be evolved by propagating changes in one viewpoint to other related viewpoints. This is done by explicitly connecting viewpoint elements via correspondence relationships to elements of other views and to compute change options using Answer Set Programming [54].

3.1.2.2 Managing inconsistencies with Quasi-Classical (QC) logic

A inconsistent specification (formalized in a traditional logic) cannot be used for reasoning, since everything is derivable from it. Non-classical paraconsistent logics permit reasoning with inconsistent specifications and hence have been considered as a means to tolerate inconsistencies.

Hunter and Nuseibeh [82] present a set of techniques for tolerating, diagnosing and tracking inconsistencies. More precisely, a labelled version of Quasi-classical logic (QC) [16] is proposed as a requirements specification language. QC logic permits reasoning without trivialization and therefore allows us to tolerate and reason with inconsistent requirements specifications. Inconsistencies are diagnosed and tracked by uniquely labelling each sentence and combining labels whenever a rule of inference is applied. This allows us to identify all sentences that are involved in a particular inference, i.e. in inferring an inconsistency $\perp$. The authors proceed to show how the degree of confidence into an inference can be identified by distinguishing between an *existential inference* (the inference can be made from at least one maximal consistent subset) and *universal/free inference* (the inference can be made from all maximal consistent subsets), with the idea being that there is more confidence into part of the specification that is universally derivable. Observe, existential- and universal inference have a direct correspondence to credulous and skeptical inference in non-monotonic reasoning.

While the work by Hunter and Nuseibeh [82] permits reasoning with inconsistent requirements specifications, it requires goals to be formalized in QC logic and treats the requirements specification as an unstructured set of requirements. The former limitation is (to some extent) addressed in the work by Easterbrook and Chechik [39], who further abstract from the underlying logic by specifying truth values as "horizontally-symmetric lattices" and defining the corresponding logical operations (i.e. disjunction and negation) on these.

3.1.3 Managing inconsistencies using Non-monotonic Formalisms

Incomplete knowledge is inherent to requirements engineering and requires tentatively reasoning with requirements. It is therefore not surprising that non-monotonic formalisms have found an application in the area of requirements engineering.

3.1.3.1 Managing inconsistencies in THEORIST

The work by Zowghi et al. [170] pioneered the use of a non-monotonic formalism to represent the requirements specifications. More precisely, Zowghi et al. [170] show how the THEORIST system $\langle F, H, C \rangle$ (see Chapter 2.1.3.1 for a brief summary) can be used to record and non-monotonically reason with inconsistent requirements. In essence, the domain knowledge, as well as requirements that are necessarily true (i.e. mandatory), are captured in the set of facts F, while tentative requirements are recorded as (pre-requisite free semi-normal) default rules in the set of hypotheses H. The THEORIST system allows us to non-monotonically reason with inconsistent requirements by constructing maximal scenarios (i.e. default extension). As pointed out by Zowghi et al. [170], this allows us to reason over maximal consistent sets of requirements and avoids an early commitment to one particular set of consistent requirements. Our approach (and others [64, 63]) follows this principle.

The primary gap in the work by Zowghi et al. [170] is the lack of structural support. On the one hand, the lack of structure makes a THEORIST requirements specification hard to access by human engineers. On the other hand, it does not allow us to answer why a requirement is part of the specification. One may think that the justification for a requirement could be encoded via a pre-requisiste of the corresponding hypothesis, but this is not possible, since THEORIST requires all hypotheses to be pre-requisite free. [170]

3.1.3.2 Managing inconsistencies REFORM

In Ghose [64, 63], the REFORM framework is motivated by a set of features that any framework for handling requirements inconsistency and evolution should support. We now briefly summarize the features that relate to managing inconsistencies. According to Ghose [64, 63], a requirements engineering framework for managing inconsistencies should support

- *Multiple Stakeholder Perspectives*: ...reasoning with multiple potentially inconsistent stakeholder perspectives (i.e. viewpoints).

- *Generation of Maximal Consistent Sets of Requirements:* ...the generation of maximal consistent sub-sets.
- *Requirements Reuse:* ...requirements reuse, i.e. requirements should never completely be discarded from the goal model in anticipation of future reuse.
- *Prioritization of Requirements:* ...the partitioning of requirements according to their level of priority.
- *Justification of Requirements:* ... the representation of a requirements justification.

Following the joint work with Zowghi and Peppas [170], Ghose makes use of a non-monotonic formalism to represent and reason with inconsistent requirements. In particular, the system allows us to prioritize requirements by distinguishing between functional and non-functional requirements of essential or tentative nature. While essential requirements are treated inviolate, tentative requirements may be violated. Requirements justification is taken into account by representing requirements as a pair $\alpha : \beta$ where α represents the requirement and β its justification (or rational). A set of requirements is defined to be r-consistent if the requirements are conjointly satisfiable with their justifications and the system trajectories. A set of requirements is maximally r-consistent if it is r-consistent, contains all essential (functional and non-functional) requirements and as many tentative requirements as possible. As shown in the work by Zowghi et al. [170], reasoning with maximal consistent sets provides an elegant way to handle inconsistency and nurtures a late requirements commitment.

While the framework to be proposed in this thesis sets out to support the same list of features as listed by Ghose, gaps have remained in how these features are addressed. For example, while the REFORM framework allows us to explicitly capture the justification of a requirement, Ghose's primary concern is to ensure consistency between requirements and their justifications and does not consider the removal of requirements that have become "unjustified" due to the removal of their justification. More precisely, given a requirement $\alpha : \beta$, the "removal" of the requirement's justification (i.e. β) does not result in the consequent removal of α, since α remains part of at least one r-consistent set. This may result in superfluous requirements being part of the specification. Another drawback is that no additional constraints are enforced on the justification of a requirement. In principle, this permits the existence of justification cycles, i.e. where a requirement is its own justification.

3.1.3.3 Managing inconsistencies in Techne

A recently proposed, abstract modelling language that comes with built-in inconsistency management (via non-monotonic reasoning) and allows to explicitly represent the requirements problem, as defined in Jureta et al. [89], is Techne [87].

More precisely, the requirements problem is expressed by stating requirements as mandatory- and optional goals, quality constraints, soft-goals and tasks, as well as capturing the domain assumptions that constrain these requirements. A candidate solution to the requirements problem is a consistent set of task and domain assumptions, which satisfy all mandatory goals, quality constraints and some of the preferred goals and quality constraints.

Techne leverages a graph (called r-net) to record atomic statements as nodes, and various types of relationships between them. The atomic statements (e.g. ϕ) are typed depending whether they denote a goal ($\mathbf{g}(\phi)$), quality constraint ($\mathbf{q}(\phi)$), soft-goal ($\mathbf{s}(\phi)$), task ($\mathbf{t}(\phi)$) or domain assumption ($\mathbf{k}(\phi)$). In other words, each requirement or domain assumption corresponds to an atomic statement in the r-net. An attitude free r-net (is a subgraph of the respective r-net and) relates atomic statements via relationships of type *inference* and *conflict*. For example, relating the atomic statements $\mathbf{g}(g_1)$, $\mathbf{g}(g_2)$ and $\mathbf{k}(k_1)$ to the statement $\mathbf{g}(g_0)$ via an inference relation denotes that the two goals g_1 and g_2 together with the domain assumption k_1 entail the goal g_0. In other words, the statement g_0 is justified by the statements g_1, g_2 andk_1. A special type of the inference relation is the conflict relation. Relating a goal g_3 and g_1 via an inference relation to a special atom $\perp$ denotes that g_3 and g_1 are inconsistent, i.e. there is a conflict relation between g_1 and g_3. A r-net (with attitude) allows us to distinguish between mandatory and optional requirements. While mandatory requirements must be part of any candidate solution, optional requirements are merely desired to be part of a candidate solution. In addition, a r-net allows us to state that some requirements are strictly preferred over others, which allows us to place a requirement on the preference spectrum between mandatory- and optional requirements (the preference relation is only vaguely discussed in [87]).

The mindful reader may have noticed that an r-net has some resemblances to the dependency network of an Assumption-based Truth Maintenance Systems (ATMS). In fact, a r-net (without preference relations) *exactly* resembles an ATMS dependency network. Requirements and domain assumptions which are mandatory correspond to premises, while tasks correspond to assumptions in the dependency network. The inference- and contradiction relation correspond to ATMS justifications. Recall that

the ATMS leverages a (separated) inference engine to derive justifications between nodes. In Techne, the requirements engineer takes the role of the inference engine. In other words, the requirements engineer would state that a particular set of requirements is minimally inconsistent. The close connection between an ATMS and an r-net is no coincidence, as a solution to the requirements problem corresponds to finding a minimal and consistent set of assumptions, such that the context of these assumptions includes all mandatory statements as well as optional statements. More precisely, the ATMS ensures that each node maintains minimal and consistent sets of assumptions (i.e. environments) that justify the node. As pointed out in Ernst's PhD thesis [44], finding a solution to the requirements problem for a single goal g_0 corresponds to selecting one of the (potentially many) environments of the node.

Given the space of all maximal conflict free r-nets, for which all source nodes are tasks or domain assumptions, the additional relationships are then used to identify candidate solutions to the requirement problem and to discriminate amongst these (in Ernst et al. [43] the authors show how this can be cased as a local search problem).

Techne [87] extends existing approaches on inconsistency management in requirements engineering (e.g. Ghose [64, 63], Zowghi et al. [170] and Hunter and Nuseibeh [82]) by taking into account the hierarchical structure of goals/requirements.

However, an important limitation of Techne [87] is the prerequisite to represent goals (and domain assumptions) as atomic statements of a propositional horn logic. While this has computational advantages, it does not allow us to consider goal semantics. On the one hand, this requires the human engineer to *manually* establish relationships (like conflict relationships) between goals (and domain assumptions), potentially resulting in overlooked or erroneous conflicts. On the other hand, it does not (or only to some extent) allow reasoning with derived goals, as may be desirable in a compliance setting.

3.2 Requirements/Goal Maintenance

Tolerating inconsistencies is important and many frameworks that have been discussed in the previous section support this feature. However, to address a given change request, the potentially inconsistent, multi-perspective goal/requirements model needs to be maintained. We now summarize how this is done in existing frameworks.

3.2.1 Maintenance in QC Logic

The work by Hunter and Nuseibeh on paraconsistent Quasi-classical logic (QC) [82] is extended in Nuseibeh and Russo [126] where the authors show how consistencies of an evolving specification can be resolved via abductive reasoning. Following the principles of the Viewpoints framework [125], a requirements specification is treated as multiple partial specifications related to each other via consistency rules - both are expressed in QC logic. Such a requirements specification is inconsistent if one of the rules is violated. The key idea is that abductive reasoning can be used to identify explanations that violate a rule, i.e. that make the negation of the rule derivable. Removing these explanations makes the negation of the consistency rule non-derivable and therefore resolves the conflict.

The definition of abduction for QC logic (as given by Nuseibeh and Russo [126]) suggests that explanations are not required to be minimal. Consequently, resolving a conflict may remove more from the specification as is actually required. Like retaining superfluous goals/requirements, such a situation should be avoided. In implementation of their approach using abductive logic, programming is discussed, but no experimental evaluation is given.

3.2.2 Maintenance in ARC

Following the work by Nuseibeh and Russo [126], Garcez and his co-authors [52, 31] propose a cyclic analysis and revision approach (we refer to their work as "ARC") to evolve a requirements specification. During the specification analyses, abductive reasoning is used to identify hypotheses (counterexamples), which allows the derivation of the negation of a system property. If no such counterexample is found, then all system properties are considered as satisfied. In the case that a counterexample is found, then inductive learning techniques are used to revise the specification, such that the counterexample is avoided. This is done by translating the specification in a neural network and making use of back-propagation to find a trained neural network, which, if translated back into the requirements specification, avoids the counterexamples.

An important deficiency of the work by Garcez and his co-authors [52, 31] is that the revised requirements specification may not be a minimal modification of the original specification. Like in the work by Nuseibeh and Russo [126], this may result in the unnecessary loss of (potentially) important goals/requirements.

3.2.3 Maintenance in THEORIST

Given a requirements specification in Theorist, Zowghi et al. [170] show how such a specification can be evolved via belief revision inspired contraction and revision operators. Following the work by Ghose on practical belief revision [65], the operators map one THEORIST system to another and are shaped by a set of AGM postulate inspired properties. The properties are as follows: (1) The outcome of a revision/contraction is a consistent THEORIST specification, i.e. $F \cup C$ is satisfiable; (2) The outcome of a revision/contraction must be successful, i.e. in case of revision all maximal scenarios include a particular requirement, in case of contraction all maximal scenarios do not include a particular requirement; (3) The revision/contraction of equivalent formulas has the same outcome, and (4) change should be minimal. Following these properties, the revision of a requirements specification $\langle F, H, C\rangle$ by a formula ϕ is defined by a revision of the set of facts F with ϕ using (classical) AGM revision. Formulas, which have to be contracted to incorporate ϕ in F, are added to the set of hypotheses H. Furthermore, all constraints that are conflicting with the new set of facts are contracted from the set of defaults. The contraction of a formula ϕ from a requirements specification $\langle F, H, C\rangle$ is given by revising the set of constraints C with $\neg\phi$ and ensuring that the facts are not inconsistent with this revised set, i.e. by contracting the negation of the revised set from the facts. The facts that have to be contracted are added to the set of hypotheses. A key feature of both the contraction and revision operator is that permanent loss of requirements is avoided by always retaining requirements either in the set F or H.

The framework proposed in this thesis builds in many ways on the results presented in Zowghi et al. [170]. We also make use of non-monotonic reasoning to manage inconsistencies and draw on belief revision principles (like minimal change) when revising a goal model. While the maintenance operators, proposed in Zowghi et al. [170] support a modular definition of minimal change, no concrete instantiation of minimal change is discussed. We believe that examples of concrete instantiations provide additional guidance to the requirements engineering when applying the framework. This is particularly true when the goal model has a structure and goal with a distinct ontological status (as is the case in our work). We shall return to the work by Zowghi et al. [170] in Chapter 5.3 for a detailed comparison.

Following Zowghi et al. [170], Antoniou and Williams [8] show how a default theory representation of a requirement specification (e.g. in THEORIST) is revised without the immediate need to recompute extensions (i.e. maximal scenarios). In other words,

extension computation is deferred until it becomes necessary and is referred to as "lazy evaluation". This has computational advantages, as computing extensions is known to be a hard problem. The authors consider the default theory revision in a credulous and sceptical case. In the credulous case, the default theory is revised such that the change is reflected in at least *one extension*, while leaving the other extensions unchanged. If the default theory is restricted to pre-requisite free semi-normal defaults, the revision of the default theory by a requirement ϕ is captured by removing $\neg\phi$ from the facts and adding $\frac{:\phi}{\phi}$ to the set of defaults (the authors also show how this can be achieved for non-restricted default theories). In the sceptical case, the revision must be reflected in *all* extensions. This is simply achieved by revising the set of facts with ϕ (recall that the set of facts must be consistent and is part of all extensions).

Also following the principles of lazy evaluation, Antoniou and Ghose [5] show how direct conflicts among requirements can be identified and represented in a "blocking graph" [109], without the need to compute default extensions. In essence, a set of defaults does block a single default δ (i.e. is a blocking set of δ) if it is a minimal set of defaults such that all defaults in the blocking set can be applied and the default δ cannot be applied. The vertices of a blocking graph are defaults (i.e. requirements), and edges between them denote conflicting relationships. Besides avoiding the need to compute extensions, a blocking graph provides a compact overview of the conflicts between requirements, which eases their resolution during consequent stakeholder interviews.

3.2.4 Maintenance in REFORM

In the REFORM framework by Ghose [64, 63], the evolution of a requirements specification is supported by a set of change operators. Inspired by the belief revision literature, Ghose motivates a set of features these operators should support. The operators should support

- *Minimal Change:* ... minimal change, i.e. changes should be incorporated in a manner that minimizes the extent of change to the model.
- *Trade-off analysis:* ... a trade-off analysis, to ensure that less important requirements are discarded in favour of relatively more important requirements.
- *Delayed Commitment:* ... to delay the choice amongst multiple (equally preferred) change outcomes as long as possible.

The REFORM framework [64, 63] includes two maintenance operators, one operator for adding essential requirements and another operator for removing (tentative or essential) requirements. Roughly, an essential requirement is added by adding it to the set of essential requirements and computing maximal consistent subsets of essential requirements. A selection function is deployed to choose amongst the multiple options. Requirements which are not part of the selected option are demoted from essential to tentative requirements. The removal of a requirement $\alpha : \beta$ is done by ensuring that α is non-derivable from any maximal r-consistent set. This is simply done by adding a new essential requirement $\varnothing : \alpha$.

An important characteristic of the work by Ghose [64, 63] is that the success criteria of the operators is inspired by non-prioritized belief revision, i.e. requirements should only be discarded after a detailed trade-off analysis, to further ensure that important requirements are not discarded in favour of less important requirements. This, in principle, permits the rejection of a change request. We agree with Ghose that less important requirements should be discarded in favour of more important requirements.

A detailed description of how our framework addresses the above features in comparison to the work by Ghose [64, 63] is given at the end of Chapter 5.

3.2.5 Maintenance in REKB

In Ernst et al. [42] (a more detailed account is given in Ernst's PhD thesis [44]), the authors build on the work on Techne and show how a solution to the requirements problem can be found in a dynamic context, i.e. when requirements and domain assumptions change. More precisely, the authors propose a *requirements engineering knowledge base* REKB (which treats a Techne model as an ATMS dependency graph) and show how it is maintained and queried in the context of change.

To take into account various kinds of evolutionary changes, e.g. adding and removing goals, tasks and domain assumption, the REKB needs to be maintained. The REKB is maintained by manipulating the underlying ATMS dependency graph. This is done via a set of basic tell and untell operations, which allow us to add, remove, or modify sentences and justifications in the dependency graph. More interesting are the three query operators, which are used to retrieve information from the REKB. The first operator "ARE_GOALS_ ACHIEVED_FROM" takes as input a set of tasks and goals and returns whether the latter satisfies the former (i.e. the latter entails the former). The second operator "MINIMAL_ GOALACHIEVEMENT" takes as input a set of goals and returns all sets of tasks that entail the goals that were given as

an input. As pointed out by the authors, the query operators require that REKB is conflict free, i.e. reasoning is stopped when an inconsistency is encountered. In successive work Ernst and his co-authors [45] overcome this by allowing credulous non-monotonic reasoning with the REKB. In other words, reasoning is restricted to consistent subsets of an REKB. It is worth pointing out that this kind of reasoning is naturally supported by ATMS. Recall, the ATMS ensures that each node in the dependency network maintains minimal and consistent sets of environments (i.e. tasks) that justify it. Consequently, finding all tasks that make a single goal derivable is straight forward, i.e. it corresponds to selecting one of the environments of the node. In the case of multiple goals, they are linked via a justification to a single dummy node for which the respective label is determined by the ATMS, as described above.

The "MINIMAL_ GOALACHIEVEMENT" operator can be used to compute a solution to the requirements problem by selecting the mandatory goals as operator input. It is worth pointing out that this approach requires additional consistency checking on top of the computations performed by the ATMS, i.e. it must be ensured that the selected mandatory (and some subset of the optional goals) are in fact consistent. The computation of potentially sub-optimal solutions using local search is discussed in Ernst et al. [43].

The third operator "GET_MIN_CHANGE_TASK_ENTAILING" addresses the requirements problem in a dynamic (i.e. changing) environment. It is a modification of the second operator. It takes a set of goals as input, and returns sets of tasks which minimize a distance function, i.e. the operator returns task sets which are minimal modifications to the current set of tasks. The operator can be implemented via a call to the second operator and then by filtering the returned set of tasks according to the distance function. [42] The distance function can be instantiated in various ways and the authors propose different intuitions of minimal change.

While the work by Ernst and his co-authors [42, 44, 45] is, in many ways, a big step forward (e.g. by emphasizing the structure of the goal model and providing concrete algorithms and tool support), gaps have remained. The work by Ernst and his co-authors is based on Techne [87], which requires goals (and domain knowledge) to be atomic statements; hence goal semantics cannot be considered during the maintenance. Furthermore, the solutions returned by the query operators do not indicate how they have been derived. For example, the "MINIMAL_ GOALACHIEVEMENT" takes a set of goals as input, and returns all sets of tasks that minimally entail the given goals. However, it is not clear what domain knowledge and sub-goals were used to make

this inference. Additionally, the application of the query operator presumes that there exists a solution. In situations where only partial solutions are available, the operators do not return a solution. This is undesirable in situations where stakeholders want to work and reason with partial solutions.

3.2.6 Maintenance in GSA

The goal-structured analysis (GSA) framework proposed by Duffy, MacNish, McDermid and Morris [38] provides a structured representation of goal model elements. In this representation, elements are represented using "frames"[115]. The justification of each element is captured via references to other elements in the frames justification slot. To avoid justification cycles, a partial ordering over all elements is enforced. The idea is that an element's justification must have a strictly higher order than the element itself. In addition, the formal assertion associated with elements of strictly lower order must together entail the assertions of elements of strictly higher order for the latter to be supported by the former.

While the GSA framework supports a structured representation, it faces two crucial limitations. First, elements of lower order are not required to *minimally* entail elements of higher order (as, for example, is required in the definition of a correct refinement given in Darimont and Lamsweerde [30] (summarized in Chapter 2.2)). This permits the existence of elements at a lower order which may be superfluous, i.e. they could be removed, and elements at a higher order that would still be supported (entailed by) elements of lower order. Second, their approach does not support the toleration of inconsistencies. In other words, all elements in the model must be consistent - at all times. As mentioned above, this may require a premature, and potentially suboptimal, commitment to a consistent set of elements. This is a crucial distinction between our approach and the ones discussed in the previous section.

MacNish and Williams [113] extend the GSA framework to take into account situations in which the model needs to be revised.[2] In particular, the partial ordering enforced over elements is used to apply a belief revision technique called "maxi-adjustment transmutation" [164], which modifies a partial ordering over sentences as little as possible whilst incorporating new information.

On the positive, this approach supports reuse of goals by retaining goals at the lowest order when they are inconsistent with other goals that have been assigned a

[2]In the following, we will refer to the work by MacNish and Williams [113] as a part of the GSA framework.

higher order. However, the work by MacNish and Williams [113] fails to address important aspects (as acknowledged by the authors). Given a goal, and its refinement into sub-goals and the removal of the parent goal, the maxi-adjustment transmutation only ensures that one of the sub-goals is removed, resulting in (potentially) superfluous goals remaining in the goal model. Furthermore, their approach does not support multi rooted structures and elements that are shared by more than one refinement.

3.3 Summary

This section provides a summary of gaps in existing maintenance approaches. In particular, Table 3.1 compares existing approaches in terms of supported maintenance features.

In Table 3.1, the top row lists all related maintenance approaches, while the first column lists all considered features. We use "x", or "∘", or "-" to denote that a particular feature is supported, or partially supported, or not supported, respectively. The feature "just. struc." (justification structure) is supported if the justification relation between elements can be captured and is taken into account in the maintenance process. The feature "hierarch. entail." (hierarchical entailment) is supported if the goal/requirements model has an associated entailment relation that permits reasoning at different levels of abstraction. The feature "goal reuse" is supported if goals are not permanently lost due to a change. The feature "lang. angostic" (language agnostic) is supported if the formalization of goals/requirements is not restricted to a particular language. The feature "minimal change" is supported if it is ensured that the model is minimally modified to incorporate a given change request. The feature "tol. incon." (tolerate inconsistency) is supported if inconsistencies can be tolerated. The feature "impl.+eval." (prototype implementation and evaluation) is supported if the performance of a prototype was evaluated.

As can be observed in Table 3.1, there are important gaps in *all* existing maintenance frameworks. Most dominant is the lacking support of a hierarchical entailment relation which avoids the need to reason with all elements in the goal/requirements model. Furthermore, not much attention was given to the rationale of each goal/requirement. In particular, the justification links between goals/requirements are not explicitly captured and rarely taken into account during maintenance. Also, very little has been

[3] Allows us to capture a requirement's justification, but does not avoid situations in which unjustified requirements remain after a change.

[4] An entailment relation is discussed to relate elements at different levels of refinement.

	GSA [113, 38]	**QC** [126, 82]	**ARC** [52, 31]	**THEORIST** [170]	**REFORM** [64, 63]	**REKB** [42, 44]
just. struc.	∘[3]	-	-	-	∘[3]	x
hierarch. entail.	-[4]	-	-	-	-	-
goal reuse	x	x	-	x	x	x
lang. agnostic	x	-	-	x	x	-
minimal change	x	-	-	x	x	x
tol. incon.	-	x	-	x	x	x
impl.+eval.	-	-	-	-	-	x

Table 3.1: Comparison of maintenance frameworks and their supported features ("x" = support, "∘" = partial support, "-" = no support).

done in terms of prototype implementation and evaluation.

Over the course of the next chapters, we propose a maintenance framework that supports all of the above listed features.

Chapter 4

Goal Model Representation

In this chapter, we introduce a novel goal model formalization that supports the list of maintenance features motivated in the previous Chapter (see Chapter 3.3). The formalization supports a hierarchical entailment relation (see Chapter 4.2.5), goal reuse (see Chapter 4.2.2) and leverages a particular type of AND/OR graph (a "solution graph") to capture the justification relation between goals (see Chapter 4.2.3) and to manage inconsistencies via possible world reasoning (see Chapter 4.2.4). AND/OR graph formalizations are commonly used to represent goal models and also play an important role in our approach. We therefore start this Chapter by highlighting issues that existing AND/OR graph formalizations of goal models face in a dynamic context, i.e. when these structures need to be maintained.

4.1 Issues in AND/OR graph based goal formalizations

In popular GORE frameworks like i* [167], NFR [121] and KAOS [28] goal model formalizations are based on AND/OR graphs. Often the AND/OR graph itself is referred to as the goal model. [152] The need for a maintenance approach that takes into account structure was motivated in the previous chapter. This section highlights issues of AND/OR graph based goal models in an evolutionary setting, i.e. when these structures need to be maintained.

4.1.1 Adding refinements requires addition of "dummy" nodes

Typically, AND/OR graph based goal models are formalized as *binary* graphs, e.g. following the definition by Nilsson [124], as pointed out by Lamsweerde [152]. In a binary graph (i.e. a graph with binary edges) extra information is required to distinguish whether sub-goals are part of an AND- or OR-refinement. In Nilsson's work [124] this is done by associating "AND" or "OR" labels with nodes/goals to denote that either all, or respectively one, of the sub-goals must be satisfied for the respective parent goal to be satisfied. This formalization may require additional nodes/goals to be included in the graph.

Example. *Figure 4.1 (a) shows the AND/OR graph representation of the goal model shown in Figure 2.1. Now consider an additional goal* ***g****, which states that exactly one available health care professional is sent to patients in need. Suppose that goal* ***d*** *can also conjointly be refined to goals* ***e*** *and* ***g*** *(besides* ***f*** *and* ***e****). Observe, linking the goal*

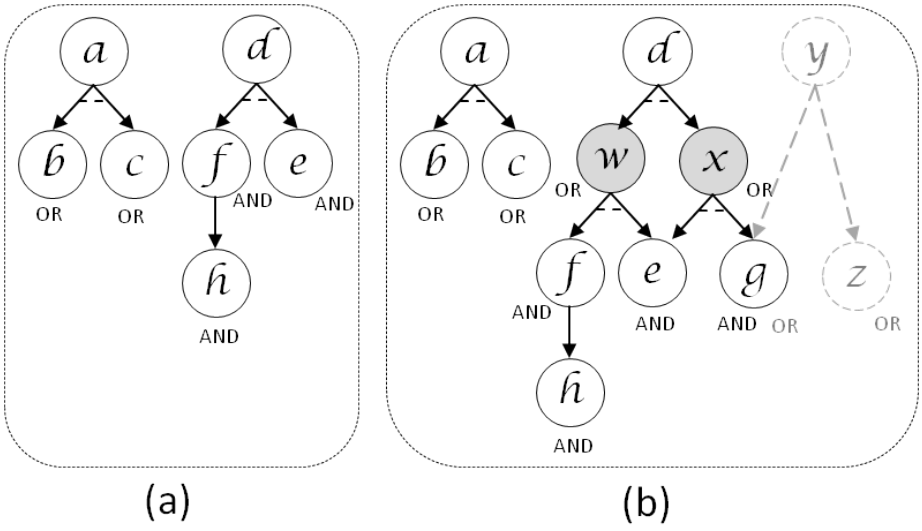

Figure 4.1: Example of an AND/OR graph.

g *as a sub-goal of* ***d*** *and labelling it as an AND-node would denote that* ***d*** *is satisfied, if* ***f****,* ***e*** *and* ***g*** *are satisfied - which is not what we want to state. This can be resolved by including additional "dummy" nodes* ***w*** *and* ***x*** *(grey shading) as exclusive parents of the two alternative refinements* $\{\boldsymbol{e}, \boldsymbol{f}\}$ *as well as* $\{\boldsymbol{e}, \boldsymbol{g}\}$ *and labelling them as "OR" nodes. The respective AND/OR graph is shown in Figure 4.1 (b).*

In a GORE setting, the addition of "dummy" nodes may cause confusion. In this setting, nodes in the graph directly correspond to goals that have been elicited and

stakeholders may hence be wondering where these additional ("dummy") goals are coming from. On the other hand, one can argue that the "dummy" nodes do add an additional (intermediate) level of refinement, which may nurture the exploration of further refinements. Nevertheless, the decision to add or omit nodes like **w** and **x** should be made by the user and not forced upon him by the AND/OR formalization.

It is also worth highlighting, as already pointed out by Martelli and Montanari [114], that the formalization given by Nilsson [124] also suffers the problem that a node can be required to be labelled as both AND and OR. This is the case when a node has two different parent vertices, where one requires it to be an AND and the other to be an OR.

Example. *In Figure 4.1 (b), consider the (dashed) nodes* ***y*** *and* ***z****, where* ***g*** *and* ***z*** *are alternative refinements of* ***y****. Observe that* ***g*** *is required to be an AND-node (as a refinement of* ***x****) and an OR-node (as a refinement of* ***y****).*

One way to overcome this problem is to add (yet another) "dummy" node. For example, adding a node **n**, labelled "OR" as the only refinement of **y** and adding the nodes **g** and **z** as a refinement of **n**, allows the node **g** (and **z**) to be an "AND"-node.

Another way to overcome this problem has been proposed by Martelli and Montanari [114]. As opposed to labelling nodes according to their ingoing edges (as done by Nilsson), Martelli and Montanari overcome this problem by labelling nodes as "AND" and "OR" according to their outgoing edges.

Example. *In Figure 4.1 (b), labelling the node* ***y*** *as "OR" denotes that either* ***g*** *or* ***z*** *must be satisfied for* ***y*** *to be satisfied - this achieves the intended meaning.*

Nevertheless, the approaches by Nilsson as well as Martelli and Montanari both make use of labelled nodes to denote AND/OR-refinements. In situations, in which we wish to say that a node has both an AND-refinement as well as an OR-refinement (e.g. if the AND/OR graph must be build from a fixed given set of nodes - such a situation will be discussed in Chapter 4.2.2) this makes the addition of one or more "dummy" nodes unavoidable.

4.1.2 Permanent loss of removed goals

A goal is a state of affairs stakeholder(s) would like to bring about [152]. In formal frameworks, goals are expressed in a *formal language* $\mathcal{L}$ in addition to their natural language description.

Example. *The goals of the running example are formulated in a many-sorted first-order language (we omit temporal operators for simplicity) as follows:*[1]

***a**: Any health care professional who is dispatched to a patient has access to the patients medical data. Formally:*
$\forall\ p \in HCProfessionals,\ \forall\ d \in Dispatchers,\ \forall\ pa \in Patients:$
$Dispatch(d, p, pa) \rightarrow AccessHealthData(p, pa).$

***b**: Any health care professional who is dispatched to a patient is communicated the patients medical data over radio by a dispatcher. Formally:*
$\forall\ p \in HCProfessionals,\ \forall\ d \in Dispatchers,\ \forall\ pa \in Patients:$
$Dispatch(d, p, pa) \rightarrow RadioTransHealthData(d, p, pa).$

***c**: Any health care professional who is dispatched to a patient receives the patients medical data via SMS. Formally:*
$\forall\ p \in HCProfessionals,\ \forall\ d \in Dispatchers,\ \forall\ pa \in Patients:$
$Dispatch(d, p, pa) \rightarrow ReceiveHealthDataSMS(p, pa).$

***d:** Health care professionals are dispatched to patients in need of medical help. Formally:*
$\forall\ pa \in Patients,\ \exists\ p \in HCProfessionals,\ \exists\ d \in Dispatchers:$
$RequiresHelp(pa) \rightarrow Dispatch(d, p, pa)$

***e:** A health care professional is available. Formally:*
$\exists\ p \in HCProfessionals:\ \ Available(p).$

***f:** Available health care professionals are dispatched to patients in need. Formally:*
$\forall\ pa \in Patients,\ \forall\ p \in HCProfessionals,\ \exists\ d \in Dispatchers:$
$(RequiresHelp(pa) \wedge Available(p)) \rightarrow Dispatch(d, p, pa).$

***g:** Exactly one available health care professional is dispatched to each patient in need. Formally:*
$\forall\ pa \in Patients,\ \forall\ p \in HCProfessionals,\ \exists\ d, d' \in Dispatchers,$
$\neg\exists p' \in HCProfessionals:\ \ ((RequiresHelp(pa) \wedge Available(p)) \rightarrow$
$Dispatch(d, p, pa)) \wedge Dispatch(d', p', pa) \wedge p \neq p'.$

***h:** Available health care professionals have been contacted and are dispatched to patients in need. Formally:*

[1] The sort "HCProfessionals" denotes the set of health care professionals.

$$\forall\ pa \in Patients,\ \exists\ p \in HCProfessionals,\ \exists\ d \in Dispatchers:$$
$$(RequiresHelp(pa) \wedge Available(p)) \rightarrow$$
$$(Contact(p) \wedge Dispatch(d, a, p)).$$

While the formulation of goals in a formal language permits the application of automated reasoning techniques [152], it is often costly and time consuming to formulate goals [170]. Therefore, in situations where stakeholders cease to desire a goal's realization, or may find that the goal is infeasible to achieve, the goal should not completely be removed from the goal model, but retained in anticipation of further reuse, such as when the achievement of the goal becomes desirable and feasible [170].

4.1.3 Goal justification is not clear

The removal of a goal from an AND/OR graph may require additional sub-goals to be removed. However, in a GORE setting it is often not clear which (sub-)goals should remain and which should be removed, as they may have multiple sources of justification.

Example. *Figure 4.2 (a) extends the AND/OR graph shown in Figure 4.1 (a) with the goal* ***o*** *and its refinement to the sub-goals* ***p***, ***q***, *and* ***h***. *Suppose, the goal* ***o*** *has become undesirable and is removed from the AND/OR graph (see Figure 4.2 (b)). This begs the question whether goal* ***q***, ***p***, *and* ***h***, *should also be removed? One may argue that* ***p***, ***q*** *and* ***h*** *were only adopted to refine and eventually contribute to the fulfilment of goal* ***o*** *and hence should be removed. However, goal* ***h*** *also contributes to the refinement of another goal (i.e. goal* ***f****) and the stakeholders may, regardless of goal* ***o***, *desire the fulfilment of goal* ***p***. *On the other hand, retaining all goals* ***q***, ***p***, ***h*** *after the removal of goal* ***o*** *may result in a situation where the organisation actively tries to bring about a goal (e.g. goal* ***q****) that is not desired and was merely accepted as part of fulfilling goal* ***o***.

In a GORE setting, as illustrated in the example, a goal may have more than one source of justification. It may be part of more than one refinement of a parent goal, or directly justified by stakeholder desire (i.e. the goal can stand by itself although it participates in the refinement of another goal). In particular, the latter information is not captured in an AND/OR graph. As a consequence it is not clear whether sub-goals should remain or be removed as a consequence of removing their parent goal.

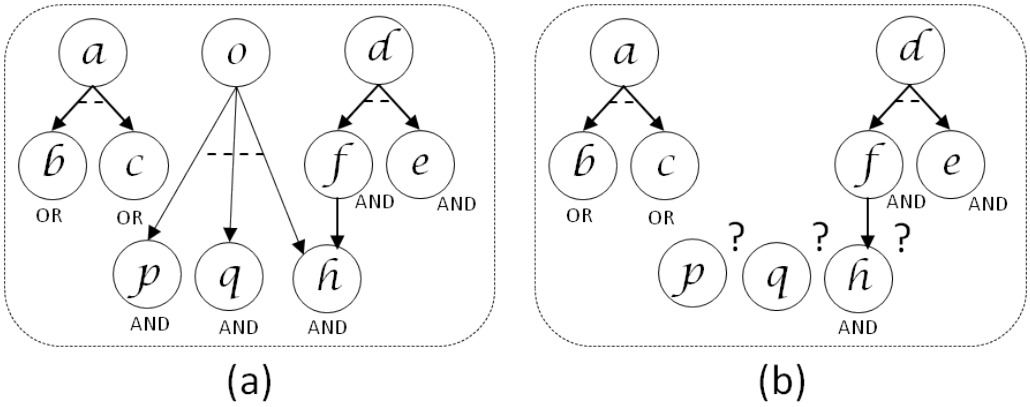

Figure 4.2: Example of sub-goals that have more than one source of justification.

4.1.4 No support of "derived goals"

Goal models formalized as AND/OR graphs, allow us to adopt different sets of goals, such that their realization satisfies all stakeholder root goals. We refer to these sets of goals as solution candidates, since these goals represent different solution alternatives to the "problem" denoted by the AND/OR graph.

Example. *The AND/OR graph in Figure 4.1 (b) permits four different ways to satisfy the (root) goals* $\boldsymbol{a}$, $\boldsymbol{d}$, *i.e. by realizing the goals* $\{\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{e}\}$ *or* $\{\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{e}\}$ *or* $\{\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{g}\}$ *or* $\{\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{g}\}$.

An AND/OR graph may permit us to adopt a solution candidate, whose realization may not be feasible, since it permits the derivation of states of affairs (i.e. "derived goals") that are non-compliant or undesired.

Example. *Given is the assumption that "at least one patient is in need of medical help and more than one ambulance is available at some point" (i.e.* $dk_3 = \exists\ p,p' \in HCProfessionals, \exists\ pa \in Patients:\ RequiresHelp(pa) \wedge Available(p) \wedge Available(p') \wedge p \neq p'$*). This permits solution candidates where stakeholders would seek to dispatch more than one health care professional to a patient in need of medical help. More precisely, the assertion* $\alpha = \exists\ p, p' \in HCProfessionals, \exists\ pa \in Patients, d \in Dispatchers: Dispatch(d, p, pa) \wedge Dispatch(d, p', pa) \wedge p \neq p' \wedge pa = pa.$ *is derivable from the solution candidates* $\{\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{e}\}$ *and* $\{\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{e}\}$. *Such a derived state of affairs may be undesirable, although each individual goal may be desirable.*

To avoid such situations, the ability to reason with "derived goals" is needed.

4.1.5 Maintenance of irrelevant solutions is nurtured

Goal models admit a *very large number* of solution candidates. This is problematic in settings where we need to maintain these graphs, such as ensuring that a goal model does not lead us to adopt a solution candidate that permits the derivation of states of affairs that are non-compliant or undesirable. In the worst case, this requires as many checks (e.g. calls to a theorem prover) as there are solution candidates. However, in many cases, we are only interested in solution candidates that satisfy certain properties, i.e. not all solution candidates are of interest.

Example. *Figure 4.3 (a) shows an AND/OR graph*[2] *with four solution candidates, i.e.* $\{\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{e}, \boldsymbol{p}\}$, $\{\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{e}, \boldsymbol{p}\}$, $\{\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{g}, \boldsymbol{p}\}$, $\{\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{g}, \boldsymbol{p}\}$. *Following the previous example, the solution candidates* $\{\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{e}, \boldsymbol{p}\}$ *and* $\{\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{e}, \boldsymbol{p}\}$ *both permit situations in which stakeholders seek to dispatch more than one health care professional to a patient in need of medical help (i.e. the assertion* α *is derivable). In other words, the solution candidates may be considered as irrelevant.*

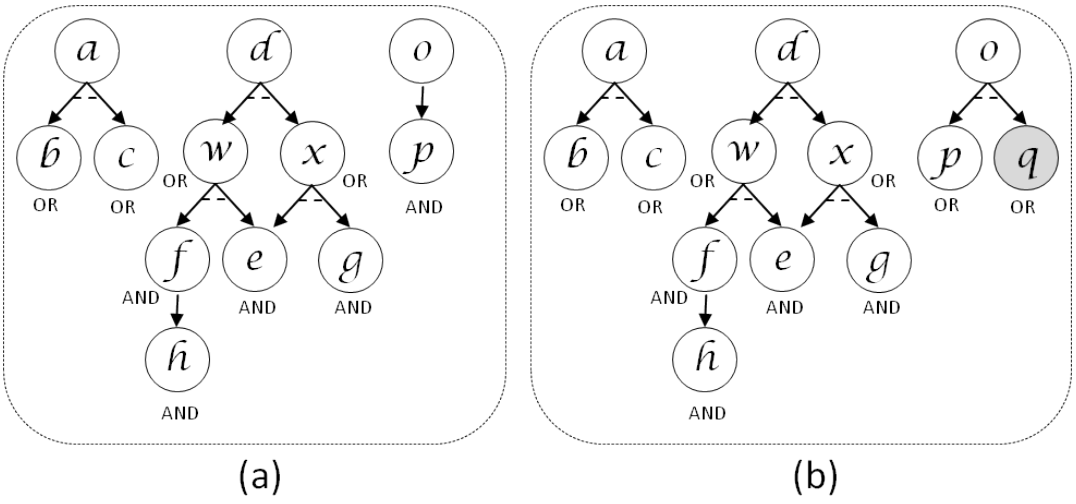

Figure 4.3: Increase of solution candidates.

Suppose a new goal $\boldsymbol{q}$ *is added as an alternative refinement of the goal* $\boldsymbol{o}$ *(see Figure 4.3). This additional refinement, doubles the total number of solution candidates. The additional solution candidates are* $\{\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{e}, \boldsymbol{q}\}$, $\{\boldsymbol{c}, \boldsymbol{h}, \boldsymbol{e}, \boldsymbol{q}\}$, $\{\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{g}, \boldsymbol{q}\}$, $\{\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{g}, \boldsymbol{q}\}$. *The four additional solution candidates require four additional calls to the theorem prover to ensure that no solution is adopted from which* α *is derivable.*

[2]The goals **o**, **p**, **q** only serve this example. We therefore omit a detailed description.

*If we can restrict the maintenance exercise to parts of the AND/OR graph that satisfy certain properties, e.g. solution candidates that do not entail α (i.e. {**c**, **e**, **g**, **p**}, {**c**, **h**, **e**, **p**}), then (in the given example) an addition of **q** only results in two additional solution candidates (i.e. {**c**, **e**, **g**, **q**}, {**b**, **e**, **g**, **q**}) and only two additional calls to the theorem prover.*

Other desired properties have been considered. For example, in Jureta et al. [89] (and earlier in Zowghi et al. [170] and Ghose [64]), the authors distinguish between compulsory goals (that *must* be satisfied) and optional goals (that are "nice" to satisfy) and require that any solution candidate to be considered should satisfy all compulsory goals and as many optional goals as possible. Typically, only a small number of solution candidates have this property.

To sum up, there is a need to be able to restrict the maintenance exercise to the "interesting" parts of the goal model.

4.1.6 No toleration of inconsistencies at the "root level"

In GORE it is often infeasible to trace all stakeholder goals back to a single goal so that a single rooted AND/OR graph is not an adequate representation. AND/OR graphs which are defined as multi-rooted trees (forests) require that all root goals are conflict free, as otherwise any solution candidate would be necessarily conflicting. However, this is an unrealistic assumption, particularly in a multi-stakeholder environment where stakeholders may have conflicting goals. This forces a potentially premature commitment to a particular set of conflict free root goals. In other words, AND/OR graphs do not tolerate inconsistencies at the root level.

Example. *Given the addition of a goal **i** "Patients have granted access to their medical data before it is accessed" (i.e. $\forall$ pa $\in$ Patients, $\forall$ p $\in$ HCProfessionals: AccessHealthData(p, pa) $\rightarrow$ GrantedHealthDataAccess(pa).) and the additional domain knowledge (or assumption) that it is possible to have patients in need of medical help who have not granted access to their medical health data (i.e. dk_1: $\exists$ pa $\in$ Patients: RequiresHelp(pa) $\wedge$ $\neg$GrantedHealthDataAccess(pa)), the goals **a**, **d**, **i** are inconsistent. We would be forced to commit to one of the consistent subsets, e.g. **a**, **d**.*

4.2 Alternative goal model formalization

In the following sections, we build up to an alternative formalization of a goal model, which is also based on AND/OR graphs, but addresses the deficiencies listed above .

4.2.1 A Simplified Formalization of AND/OR Graphs

In [50], an alternative F-hypergraph formalization of AND/OR graphs is introduced which overcomes the need to define labelling functions and additional vertices, and hence provides a simpler and more intuitive formalization. *F-hypergraphs* are generalizations of simple directed graphs that allow edges with a single source vertex, but (potentially) more than one target vertex. The following definitions are adapted from Gallo et al. [50].

Definition 2. *(F-hypergraph)*
An F-hypergraph is a pair $\langle V, E \rangle$*, where* $V = \{v_1, v_2, ..., v_n\}$ *is a set of vertices and* $E = \{e_1, e_2, ..., e_n\}$ *is a set of directed F-hyperedges, where each edge* $e \in E$ *is an ordered pair* $e = (x, Y)$*, such that* $x \notin Y$ *and* $\{x\} \cup Y \subseteq V$*.*

A cycle free path through a F-hypergraph is a sequence of distinct vertices and F-hyperedges.

Definition 3. *(Cycle Free Path)*
Given an F-hypergraph $\langle V, E \rangle$*, let* $S(e)$ *denote the (single) source vertex, and* $T(e)$ *the set of target vertices of an F-hyperedge* $e \in E$*. A path (of length q) through an F-hypergraph is a sequence of vertices and F-hyperedges* $(v_1, e_1, v_2, e_2, ..., e_q, v_{q+1})$*, where* $v_1 = S(e_1)$*,* $v_{q+1} \in T(e_q)$ *and* $v_j \in T(e_{j-1})$ *and* $v_j \in S(e_j)$ *for* $j = 2, ..., q$*. A path is cycle free if and only if all F-hyperedges and vertices in the path are distinct.*

Definition 4. *(Acyclic F-hypergraph)*
An F-hypergraph is acyclic if and only if all paths are cycle free.

An edge $e = (x, Y)$ is used to denote AND-refinements of a vertex (goal) x to a set of sub-vertices (sub-goals) Y, such that all elements of Y conjunctively refine a vertex (goal) x. In an *OR*-refinement, an edge $e = (x, Y)$ and $e' = (x, Y')$ is used to denote that the sets of vertices (goals) $Y \subset V$ and $Y' \subset V$ are distinct refinements of x. The following example illustrates this.

Example. *Figure 4.4 shows an acyclic F-hypergraph representation of the AND/OR graph shown in Figure 4.1 (b). Goal* $\boldsymbol{d}$ *is refined by* $\boldsymbol{f}$ *AND* $\boldsymbol{e}$ *via the hyperedge e1* $= (\boldsymbol{d}, \{\boldsymbol{f}, \boldsymbol{e}\})$*; d is also refined by* $\boldsymbol{e}$ *AND* $\boldsymbol{g}$ *via the hyperedge e2* $= (\boldsymbol{d}, \{\boldsymbol{g}, \boldsymbol{e}\})$*; The alternative refinement of goal* $\boldsymbol{a}$ *by goal* $\boldsymbol{b}$ *OR* $\boldsymbol{c}$ *is denoted by the two hyperedges e4* $= (\boldsymbol{a}, \{\boldsymbol{b}\})$ *and e5* $= (\boldsymbol{a}, \{\boldsymbol{c}\})$*. Observe that the alternative refinement of* $\boldsymbol{y}$ *by* $\boldsymbol{g}$

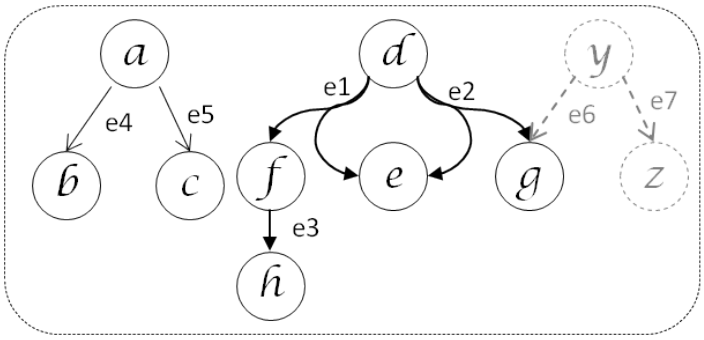

Figure 4.4: Visualization of a goal model as F-hypergraph.

OR $\boldsymbol{z}$ *(which caused problems in the formalization given by [124]) is represented via hyperedges* $(y, \{g\})$ *and* $(y, \{z\})$*.*

As can be observed, an F-hypergraph formalization of an AND/OR graph addresses the aforementioned deficiencies of Nilsson's [124] AND/OR graph formalization (e.g. the need for "dummy" nodes). It is worth pointing out that although the formalization of an AND/OR graph as a F-hypergraph is different to the formalization given by Nilsson [124] (and the improvements suggested by Martelli and Montanari [114]), their semantics is the same, as both describe the exact same set of solutions (i.e. solution candidates).

4.2.2 Goal Library

The idea of a goal library is to support reuse and avoid unnecessary loss of information by retaining goals that are currently infeasible, inconsistent, or not desired in the library, in anticipation of future situations where they may become feasible and desirable to realize.

We expect goals to be formulated in a formal language $\mathcal{L}$, i.e. each goal is denoted by a formal expression. Candidates for $\mathcal{L}$ which have been explored in the literature include real time temporal logic (e.g. in [30]), computational tree logic (e.g. in [61]), linear time temporal logic (e.g. in [49]), many sorted logic (e.g. in [64]) and proposi-

tional horn logic (e.g. in [87]). Our framework is not prescriptive about the language in which goals are formulated.

To further increase generality of the proposed framework, we do not explicitly distinguish between goals of functional or non-functional nature[3], or goals that are classified as achievement or maintenance goals [28], since the distinction is not required to perform logical inferences (e.g. to check consistency). Goals that do not have a clear-cut criteria are referred to as *soft-goals*. We omit soft-goals for now, but show how they can be addressed in Chapter 6.

Definition 5. *(Goal Library)*
Given a formal language $\mathcal{L}$, a goal library is a set of expressions $LIB \subseteq \mathcal{L}$.

Example. *For ease of readability, we summarize all goals of the running example, which are part of the goal library.*

***a**: Any health care professional who is dispatched to a patient has access to the patients medical data. Formally:* $\forall$ `p` $\in$ `HCProfessionals,` $\forall$ `d` $\in$ `Dispatchers,` $\forall$ `pa` $\in$ `Patients: Dispatch(d, p, pa)` $\rightarrow$ `AccessHealthData(p, pa).`

***b**: Any health care professional who is dispatched to a patient is communicated the patients medical data over radio by a dispatcher. Formally:* $\forall$ `p` $\in$ `HCProfessionals,` $\forall$ `d` $\in$ `Dispatchers,` $\forall$ `pa` $\in$ `Patients: Dispatch(d, p, pa)` $\rightarrow$ `RadioTransHealthData(d, p, pa).`

***c**: Any health care professional who is dispatched to a patient receives the patients medical data via SMS. Formally:* $\forall$ `p` $\in$ `HCProfessionals,` $\forall$ `d` $\in$ `Dispatchers,` $\forall$ `pa` $\in$ `Patients: Dispatch(d, p, pa)` $\rightarrow$ `ReceiveHealthDataSMS(p, pa).`

***d:** Health care professionals are dispatched to patients in need of medical help. Formally:* $\forall$ `pa` $\in$ `Patients,` $\exists$ `p` $\in$ `HCProfessionals,` $\exists$ `d` $\in$ `Dispatchers:` `RequiresHelp(pa)` $\rightarrow$ `Dispatch(d, p, pa)`

***e:** A health care professional is available. Formally:* $\exists$ `p` $\in$ `HCProfessionals: Available(p).`

***f:** Available health care professionals are dispatched to patients in need. Formally:* $\forall$ `pa` $\in$ `Patients,` $\forall$ `p` $\in$ `HCProfessionals,` $\exists$ `d` $\in$ `Dispatchers:` `(RequiresHelp(pa)` $\wedge$ `Available(p))` $\rightarrow$ `Dispatch(d, p, pa)`

[3] The distinction may be useful, though. For example, we may wish to satisfy functional goals over non-functional goals, or vice versa [64]

g: *Exactly one available health care professional is dispatched to each patient in need. Formally:* $\forall$ `pa` $\in$ `Patients,` $\forall$ `p` $\in$ `HCProfessionals,` $\exists$ `d, d'` $\in$ `Dispatchers,` $\neg\exists$`p'` $\in$ `HCProfessionals: ((RequiresHelp(pa)` $\wedge$ `Available(p))` $\rightarrow$ `Dispatch(d, p, pa))` $\wedge$ `Dispatch(d', p', pa)` $\wedge$ `p`$\neq$`p'`.

h: *Available health care professionals have been contacted and are dispatched to patients in need. Formally:* $\forall$ `pa` $\in$`Patients,` $\exists$ `p` $\in$ `HCProfessionals,` $\exists$ `d` $\in$ `Dispatchers: (RequiresHelp(pa)` $\wedge$ `Available(p))` $\rightarrow$`(Contact(p)` $\wedge$ `Dispatch(d`

i: *A patient must grant access to his/her medical data before it can be accessed. Formally:* $\forall$ `pa` $\in$ `Patients,` $\forall$ `p` $\in$ `HCProfessionals: AccessHealthData(p, pa)` $\rightarrow$ `GrantedHealthDataAccess(pa).`[4]

The term "goal library" might be misleading, since a goal is something that we seek to realize. A goal library, by contrast, is a collection of goals that we may *potentially seek to realize.*

In a goal library we refer to an element as a *desired goal* if the stakeholders would like to realize the goal. The set of desired goals is denoted by $LIB^{des} \subseteq LIB$. Related work distinguishes between (desired) goals of *necessary-* and *optional* character (these are also respectively referred to as *essential* and *tentative* goals in [63]). This distinction allows some goals to be given strict precedence over others in situations where not all (desired) goals can be consistently satisfied.[5] We do not make this fine grained distinction between desired goals. Note, this does not restrict the generality of the proposed framework, since any combination of necessary and optional goals can be reflected via a set of desired goals.

We refer to elements in the goal library that the stakeholders have not marked as desired goals as *dependent goals.* While the fulfilment of a *desired goal* is directly justified by stakeholder desire, the pursuit of a dependent goal must be justified by a desired goal. In other words, a dependent goal cannot stand by itself and is only pursued if doing so contributes to the fulfilment of a desired goal. Typically (but not necessarily), desired goals correspond to root goals, while dependent goals correspond to their refinements.

We treat each goal as atomic, i.e. we do not consider the modification of the individual goals themselves. We therefore require each *individual goal* in the goal

[4] Note, goal **i** could be said to positively contribute to a soft-goal like "maximize patient privacy".

[5] The idea is that all necessary/essential goals must be satisfied along with as many optional/tentative goals as consistently possible.

library to be consistent for the library to be well-formed. However, goals can be modified (e.g. to resolve inconsistencies) using goal-weakening techniques as described in Lamsweerde et al. [99].

Example. *The inconsistency amongst the desired goals* ***a****,* ***i****,* ***d*** *could be resolved (amongst others) by modifying goal* ***i****, or alternatively by modifying goal* ***a*** *as follows.*

- ***i'***:∀ *pa* ∈ *Patients*, ∀ *p* ∈ *HCProfessionals*: *AccessHealthData(p, pa)* → *(GrantedHealthDataAccess(pa)* ∨ *RequiresHelp(pa))*.
  In other words, a health care professional may access a patients information if the patient has previously granted permission or needs help.

- ***a'*** ∀ *p* ∈ *HCProfessionals*, ∀ *d* ∈ *Dispatchers*, ∀ *pa* ∈ *Patients*: *(Dispatch(d, p, pa)* ∧ *GrantedHealthDataAccess(pa))* → *AccessHealthData(p, pa)*.
  In other words, a health care professional dispatched to a patient may only access the patients information if the patient has granted permission.

A disadvantage of goal-weakening techniques is that they would require us to commit to one of the modifications which may be a premature commitment and avoid the exploration of alternatives.

4.2.3 Solution Graph

Given the definition of an acyclic F-hypergraph and a goal library, we now define a goal graph as an acyclic F-hypergraph, which relates elements of the goal library.

Definition 6. *(Goal Graph)*
Given a goal library LIB, a goal graph $G = \langle V, E \rangle$ is an acyclic F-hypergraph, where $V \subseteq LIB$.

In the following, we will make use of the vocabulary described below. Given a goal graph $G = \langle V, E \rangle$, a goal is a "root goal" if and only if there does not exist an edge for which the vertex is a target. We use the function $f_{root}(G)$ to return all root goals of G. Conversely, we refer to a goal as a "leaf goal" if and only if there does not exist an edge for which the goal is a source. We use the function $f_{leaf}(G)$ to return all leaf vertices of G. The functions $S(e)$ and $T(e)$ return the source and target vertices of an edge e, respectively. The functions $f_V(G)$ and $f_E(G)$ return all vertices and edges, respectively. The functions $f_{des}(G)$ and $f_{dep}(G)$ return all desired

and dependent goals of G, respectively. The function $depth(G)$ returns the length of the longest path between any root and leaf goal in G.

Definition 7. *(Goal graph vocabulary)*
Given an acyclic F-hypergraph $G = \langle V, E \rangle$:

- $f_{root}(G) = \{v \in V | \neg \exists e \notin E : v \in T(e)\}$.
- $f_{leaf}(G) = \{v \in V | \neg \exists e \in E : S(e) = v\}$.
- $f_{des}(G) = \{V \cap LIB^{des}\}$.
- $f_{dep}(G) = \{V \cap (LIB \setminus LIB^{des})\}$.
- $f_V(G) = V$.
- $f_E(G) = E$.
- $depth(G)$ *denotes the length of the longest path between any root and leaf goal.*

We refer to a goal graph which does not contain alternative refinements as a *solution graph*. Solution graphs are interesting, since satisfying their leaf goals guarantees that all other goals of the graph are satisfied. A solution graph (for which all leaf goals are considered satisfied/realized) can hence be viewed as a "solution" that satisfies high-level stakeholder goals.

Definition 8. *(Solution Graph)*
A solution graph $\mathscr{G} = \langle V, E \rangle$ is a goal graph that does not contain two distinct edges $e, e' \in E$, such that $S(e) = S(e')$.

A goal graph can have many sub-graphs which are solution graphs, e.g. (amongst many others) each individual goal of a goal graph satisfies this property. However, we are usually interested in the solution graphs which are *maximal* with respect to a goal graph. In other words, we are interested in solution graphs that include as many goals and refinement relations as possible of the original goal graph, but retain the property of a solution graph.

Definition 9. *(Maximal Solution Sub-Graph)*
Given a goal graph $G = \langle V, E \rangle$, a maximal solution sub-graph $\mathscr{G} = \langle V', E' \rangle$ is a sub-graph of G such that (1) $\mathscr{G}$ is a solution graph, (2) $f_{root}(\mathscr{G}) = f_{root}(G)$ and (3) there does not exist a solution graph $\mathscr{G}'' = \langle V'', E'' \rangle$ that satisfies (1) and (2) and $V' \subset V'' \subseteq V$ or $E' \subset E'' \subseteq E$.

Loosely speaking, a solution graph is maximal with reference to a goal graph G (i.e. a maximal solution sub-graph) if it contains as many goals and edges of G as possible without violating the AND-refinement property or creating new root goals.

Example. *In Figure 4.5, G_1 shows the goal graph of our running example. The graphs $\mathcal{G}_{1.1}$, $\mathcal{G}_{1.2}$, $\mathcal{G}_{1.3}$, $\mathcal{G}_{1.4}$, $\mathcal{G}_{1.5}$ are all solution graphs and are also sub-graphs of $\mathcal{G}_1$. While the solution graphs $\mathcal{G}_{1.1}$, $\mathcal{G}_{1.2}$, $\mathcal{G}_{1.3}$, $\mathcal{G}_{1.4}$ are maximal solution sub-graphs, the solution graph $\mathcal{G}_{1.5}$ is not a maximal sub-graph, since the goal **h** can be added as a refinement to **f** without violating the solution graph property. Also observe that without condition (2) in Definition 9, amongst others, the solution graphs $\mathcal{G}_{1.1}$, $\mathcal{G}_{1.2}$ would not be maximal sub-graphs, since the goal **b** or **c** could be added as root goals to $\mathcal{G}_{1.2}$ and $\mathcal{G}_{1.1}$, respectively without violating the solution graph property.*

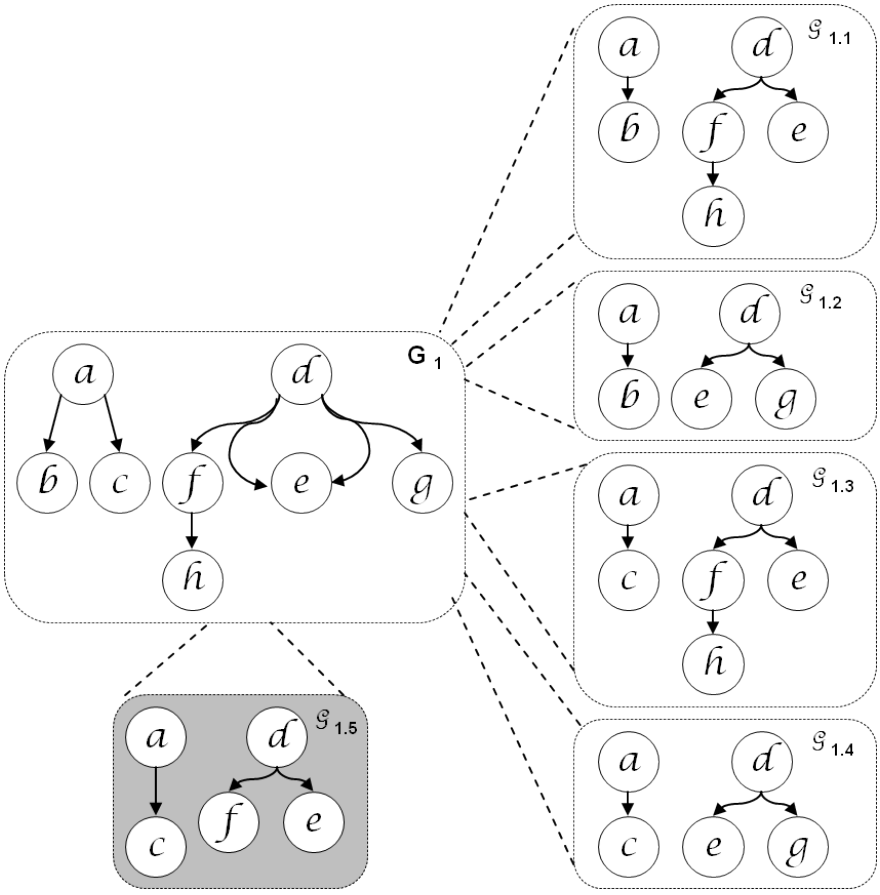

Figure 4.5: Example of a goal graph and its maximal solution sub-graphs.

We use the function $\Delta_{sol}(G)$ to denote all maximal solution sub-graphs of a goal graph G.

Definition 10. *Given a goal graph G, $\Delta_{sol}(G)$ is a set of maximal solution sub-graphs of G, such that there does not exist another set S of maximal solution sub-graphs of G where $\Delta_{sol}(G) \subset S$.*

Theorem 1. *A goal graph G can be, without loss of information, represented by a set of solution graphs* **G**.

Proof: Given a goal graph G, let $\Delta_{sol}(G) = \mathbf{G} = \{\mathscr{G}_1, \mathscr{G}_2, ..., \mathscr{G}_n\}$, then trivially $f_V(G) = f_V(\mathscr{G}_1) \cup f_V(\mathscr{G}_2) .... \cup f_V(\mathscr{G}_n)$ and $f_E(G) = f_E(\mathscr{G}_1) \cup f_E(\mathscr{G}_2) .... \cup f_E(\mathscr{G}_n)$. In other words, G is reconstructible from **G** *and hence no information is lost.*

The proof of the above theorem is completed with reference to the syntactical representation of a goal graph (i.e. AND/OR graph). However, there is also a semantic counterpart. Roughly, the semantic interpretation of an AND/OR graph is the set of solutions it represents, i.e. the AND/OR graph is uniquely identified by its solutions, like a statement in (classical) logic is uniquely identified by its interpretations. Therefore, by retaining all solutions of an AND/OR graph, we retain its semantics.

While a goal graph can be represented by all its maximal solution sub-graphs, not all of these sub-graphs are "valid", e.g. some of them may contain goals which are inconsistent and hence not conjointly satisfiable.

4.2.4 Valid Solution Graph

We now define and describe what constitutes a *valid solution graph*.

First, we require a definition of a *domain knowledge base*. A *domain knowledge base* $\mathcal{KB}$ is a consistent set of sentences expressed in the same language as the goals in the goal library. The $\mathcal{KB}$ is used to record domain specific knowledge and assumptions.

Example. *The $\mathcal{KB}$ may contain the aforementioned assumption that there exists a patient in need of medical help who has not granted access to his/her medical health data. The $\mathcal{KB}$ may also contain the domain knowledge that health care professionals are considered to have access to a patients medical health data if they have been communicated the patients health data by the dispatcher via radio or received the data via SMS. Formally, dk_2:* $\forall$ `p` $\in$ `HCProfessionals,` $\forall$ `pa` $\in$ `Patients,` $\forall$ `d` $\in$ `Dispatchers: (RadioTransHealthData(d, p, pa)` $\vee$ `ReceiveHealthDataSMS(p, pa))` $\rightarrow$ `AccessHealthData(p, pa)`

Solution graph validity is defined as follows.

Definition 11. *(Valid Solution Graph)*
A solution graph $\mathscr{G} = \langle V, E \rangle$ is valid with respect to a well-formed goal library LIB and a consistent domain knowledge base $\mathcal{KB}$, if and only if:

(1) $V \subseteq LIB$ *(library set membership) and*

(2) for all $(e = (g, S)) \in E$ *it holds that* $g \in Cn(S \cup KB)$ *and* $g \notin Cn(S' \cup KB)$ *for any* $S' \subset S$ *(correct refinements) and*

(3) $\bot \notin Cn(V \cup KB)$ *(consistency) and*

(4) there does not exist a set $DES \subseteq LIB^{des}$*, such that* $f_{des}(\mathscr{G}) \subset DES$ *and* $\bot \notin Cn(\mathcal{KB} \cup DES)$ *(desire maximality) and*

(5) $f_{root}(\mathscr{G}) \subseteq LIB^{des}$ *(desired root goals) and*

(6) for all $(e = (g, S)) \in E$*, there does not exist a* $g' \in LIB$*, where* $g' \neq g$ *and a set* $S' \subset S$*, such that* $g' \in Cn(S' \cup KB)$*. (immediate refinements).*

Library set membership: The *first condition* requires that all goals of a solution graph are also an element of the goal library. Amongst others, this ensures that the removal of any goal from a solution graph is retained in the goal library.

Correct refinements: The *second condition* requires that all refinements (i.e. hyper-edge relations between goals) are correct. Darimont and Lamsweerde [30] define a refinement of a goal by a set of sub-goals to be correct if the sub-goals are consistent and together minimally entail the parent goal (see Definition 1).

Consistency: A goal graph does not encode sequencing knowledge (exceptions include [100, 107]) that might suggest that some goals are to be achieved before others. In other words, all goals are assumed to be concurrently realizable and hence the third condition requires that all goals in a solution graph must be consistent, since it is not possible to realize two mutually exclusive states of affairs. Note, the definition of a correct goal refinement is not sufficient to ensure consistency, since it only requires the immediate sub-goals to be consistent, but permits inconsistencies via refinements of other goals.

Example. *Figure 4.6 shows a solution graph with inconsistent goals (recall that the goals* ***a****,* ***i****,* ***d*** *are not conjointly satisfiable with the $\mathcal{KB}$), but for which each refinement is correct.*

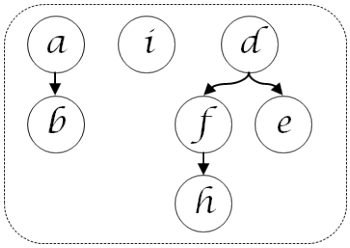

Figure 4.6: Solution graph with inconsistent goals, but correct refinements.

Desire Maximality: An organisation seeks to realize as many of the goals in LIB^{des} as possible. However, a desired goal m may not be realizable, since m may be *inconsistent* with other desired goals which may have been given higher precedence. We therefore argue that a solution graph should contain as many desired goals from the goal library as is consistently possible. This is captured in the fourth condition of Definition 11. Note, both cardinality-maximal and inclusion-maximal intuitions might be of interest. Sometimes, we aim to maximize the *number* of goals satisfied.

Example. *Given the desired goals* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{i}, \boldsymbol{d}\}$*, the following are maximal consistent subsets of desired goals:* $LIB^{des}_1 = \{\boldsymbol{a}, \boldsymbol{d}\}$, $LIB^{des}_2 = \{\boldsymbol{i}, \boldsymbol{d}\}$*. The solution graphs* $\mathscr{G}_{1.1}$, $\mathscr{G}_{1.2}$, $\mathscr{G}_{1.3}$, $\mathscr{G}_{1.4}$ *have the goals* $\boldsymbol{a}$, $\boldsymbol{d}$ *as desired goals and hence satisfy this property.*

Desired root goals: In a solution graph, parent goals provide the justification[6] for their sub-goals and consequently the root goals of a solution graph provide the justification for all other goals in the graph. We require that the root goals are desired goals, since we adopt the position that a goal designated as "desired" by a stakeholder needs no further justification. This is captured in the fifth condition of Definition 11.

We do not require all desired goals to be root goals, i.e. we permit the refinement of a desired goal into other desired goals, where possible. Note, although they are in a refinement relation, the justification of the latter remains independent of the former. In other words, the latter would still be justified, even if the former were to be removed. Also we do not require all goals of a solution graph to be elements of LIB^{des}. A *dependent goal* (i.e. a goal in LIB^{dep}) may be part of the solution graph if its realization contributes to bringing about a desired goal, i.e. if it is part of a refinement of a desired goal.

[6] The parent goal provides justification for the sub-goals, since it is the answer to "*Why?*" the sub-goals are to be brought about.

Example. *Figure 4.7 (a) shows a solution graph for which all refinements are correct and all goals are consistent. It contains a maximal consistent set of desired goals and all root goals are desired goals. Note, dependent goals* $\{\boldsymbol{b}, \boldsymbol{e}, \boldsymbol{f}, \boldsymbol{h}\}$ *can be part of the solution graph, since all of them contribute to the realization of a desired goal, i.e. they contribute to realizing the desired goals* $\boldsymbol{a}$ *and* $\boldsymbol{d}$*. On the other hand, the solution graph in Figure 4.7 (b) violates the "desired root goals" property, since* $\boldsymbol{g}$ *is a dependent root goal.*

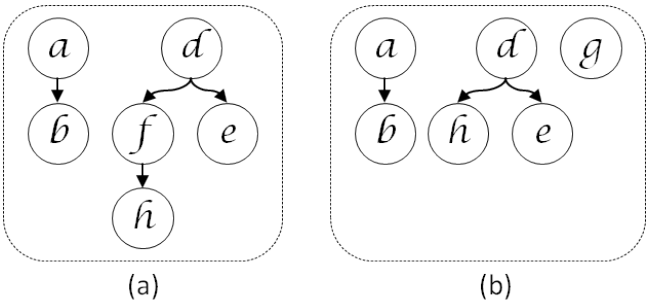

Figure 4.7: Example of two solution graphs.

Hierarchy enforcement: We say that a refinement of a goal by a set of sub-goals is *immediate* if it is correct and there does not exist another goal in the goal library that is entailed by any proper subset of the sub-goals. Intuitively, a non-immediate refinement suggests that we have skipped some levels (of sub-goals) that should have appeared in a correctly structured goal graph. Observe that otherwise (in the limit) the leaf goals of a solution graph could directly refine the graph's root goal(s), i.e. all refinements "in-between" would be omitted. Such a situation should be avoided, since each refinement provides useful insights, eases the exploration of alternative realizations and helps to resolve inconsistencies. In other words, by requiring each refinement to be immediate we *enforce the hierarchical structure* of a solution graph. This is captured in the sixth condition of Definition 11.

Example. *In Figure 4.7 (b) the refinement of goal* $\boldsymbol{d}$ *by the goals* $\boldsymbol{h}$ *and* $\boldsymbol{e}$ *is correct. However, the refinement is not immediate, since another goal in the library (i.e.* $\boldsymbol{f}$*) is entailed by a proper subset of* $\{\boldsymbol{h}, \boldsymbol{e}\}$*, i.e. the goal* $\boldsymbol{f}$ *is entailed by goal* $\boldsymbol{h}$*.*

Note, the validity of a solution graph is deliberately defined with respect to the goal library and domain knowledge base, and independently of a general goal graph from which it may be derivable (in the sense that it is an element of the goal graphs' set of maximal solution sub-graphs, $\Delta_{sol}(G)$). In other words, there can be one or more

solution graphs without the need to have identified a general goal graph from which it may be derivable. This (besides others) allows us to represent multiple solution graphs with distinct root goals (as may be the case when we have distinct maximal consistent sets of desired goals in the goal library) in a single set of solution graphs.

Given the definition of a goal library, domain knowledge base and solution graph, a goal model is defined as follows.

Definition 12. *(Goal Model)*
Let $\mathbf{G}$ *be a set of solution graphs. Given a domain knowledge base* $\mathcal{KB}$ *and a goal library* LIB*, a goal model is a tuple* $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$.

An important distinction of this goal model formalization is that goals are not represented in a single general goal graph, but as a set of solution graphs. It is worth pointing out that it follows from Theorem 1 that our goal model formalization is general enough to represent any AND/OR graph, i.e. an AND/OR graph can be represented via all its maximal solution sub-graphs.

Furthermore, the formalization does not force us to consider *all* maximal solution sub-graphs (in the case that a general goal graph has been identified beforehand). Instead the formalization allows us to restrict the maintenance exercise to solution graphs that are of interest, e.g. solution graphs that are valid[7]. It is worth highlighting that solution graphs which are not part of the set $\mathbf{G}$, are not "lost" as they are derivable from LIB and $\mathcal{KB}$ together (in Chapter 5 we describe an operator for computing solution graphs from the goal library using the domain knowledge base).

In the following, we use "valid*" as a "wildcard" for different intuitions of solution graph validity. For now, valid* can simply be substituted by solution graph validity as defined in Definition 11. In Chapter 5.2.4, this definition will be extended.

Definition 13. *(Well-formed Goal Model)*
A goal model $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$ *is well-formed if and only if (1) all solution graphs in* $\mathbf{G}$ *are valid* with reference to* LIB *and* $\mathcal{KB}$*, (2)* LIB *is well-formed and (3)* $\mathcal{KB}$ *is consistent.*

Additional properties for the set $\mathbf{G}$ could be considered. (1) A set $\mathbf{G}$ could be referred to as *complete* if every valid solution graph derivable from the goal library is an element of $\mathbf{G}$. Observe, this is not required by the definition of a well-formed

[7] Note, in some situations non-valid solution graphs may be useful to highlight inconsistencies and therefore could be the basis for the application of goal weakening techniques [99]

goal model, i.e. the set $\mathbf{G}$ is allowed to be empty although a solution graph my be constructible from the library. In other words, the stakeholders are in full control over the solution graphs they want to consider, i.e. to be part of $\mathbf{G}$.

(2) A set $\mathbf{G}$ could be said to be *desire complete* if for every maximal consistent set of desired goals $DES \subseteq LIB^{des}$[8], there exists a $\mathscr{G} \in \mathbf{G}$ such that $f_{des}(\mathscr{G}) = DES$. In other words, for every maximal consistent set of desired goals in the goal library, there exists at least one corresponding valid solution graph in $\mathbf{G}$ (there can be more than one valid solution graph for a set of desired goals, when these can be alternatively refined).

(3) A set $\mathbf{G}$ could be said to be *realizable* if the leaf goals of all valid solution graphs in $\mathbf{G}$ are satisfiable, i.e. can be correlated to a description of functionality that realizes it. For now, we assume (for ease of elaboration) that all goals in the library are satisfiable.

(4) A set $\mathbf{G}$ could be said to be *optimal* (or *k-optimal*) if $\mathbf{G}$ contains *the* most preferred (or k number of the most preferred) valid solution graphs constructible from the goal library. Such preference relation could be manually established, or with reference to satisficability of soft-goals (e.g. using techniques described in Ernst et al. [43]).

4.2.5 Goal Model Entailment Relation

We now associate a hierarchical entailment relation with a goal model.

We first require a notion of "level" to be associated with a solution graph. This allows us to reason with a solution graph at different levels of refinement. We will later show that this has both conceptual- and computational benefits.

Definition 14. *(k-level goals)*
Given a solution graph $\mathscr{G} = \langle V, E \rangle$, a goal $g \in V$ is on level 0, if and only if it is a root goal. A goal g is on level j, if (1) there exists a path of length j between a root goal and g, or (2) g is a leaf goal and the length of the shortest path between a root goal and g is i and $i < j$. We use $\mathscr{G}_k$ to denote the goals of the solution graph $\mathscr{G}$ at level k.

Definition 14 deliberately permits goals to be *associated with more than one level* (as opposed to the notion of level for a simple tree). A goal is associated with more than one level if it is a leaf goal and first appears on level i and the solution graph

[8] $DES \subseteq LIB^{des}$, where $\perp \notin Cn(DES \cup \mathcal{KB})$ and there does not exists a set DES', such that $DES \subset DES'$ and $\perp \notin Cn(DES' \cup \mathcal{KB})$

has more than i levels. This is important in the context of a solution graph, since otherwise any level j would be an incomplete description of the desired state of affairs. Consider the following example.

Example. *Figure 4.8 highlights the levels of the solution graph shown in Figure 4.7 (a). For ease of readability, goals that participate on more than one level are shown in the "first" (i.e. lowest) level in which they appear with a solid circle and also in all consequent levels with a dashed circle. The graph has 0-2 levels, i.e. goals on level 0 are:* $\boldsymbol{a}$, $\boldsymbol{d}$*; goals on level 1 are:* $\boldsymbol{b}$, $\boldsymbol{f}$, $\boldsymbol{e}$*; goals on level 2 are:* $\boldsymbol{b}$, $\boldsymbol{h}$, $\boldsymbol{e}$*. Note, if we did*

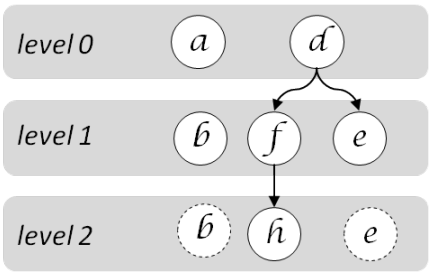

Figure 4.8: Example of k-level goals in a solution graph.

not require a leaf goal that first appears in some level i to be part of any level $j > i$, then level 2 would only consist of goal $\boldsymbol{h}$ *and be a highly incomplete representation of the stakeholders desired state of affairs at that level.*

Another situation where a goal can be part of more than one level is when it is part of more than one refinement (i.e. target goals).

Example. *Figure 4.9 (a) shows a slightly more complex solution graph (not part of the running example) and Figure 4.9 (b) shows the level(s) to which each goal in the solution graph is associated. Observe that goal* $\boldsymbol{q}$ *is part of the target goals of two distinct refinements. Again, if* $\boldsymbol{q}$ *were not part of level* 1 *or* 2*, then each level would be an incomplete representation of the desired state of affairs.*

Based on the previous definition, we will now define a hierarchical entailment relation for valid solution graphs, called k-level entailment. k-level entailment permits us to answer whether a particular assertion is derivable (or not derivable) from a certain level onwards in a valid solution graph.

An assertion is k-entailed in a *valid solution graph* $\mathscr{G}$ if and only if all k-level goals together with the $\mathcal{KB}$ entail the assertion and there does not exist a lower level in the goal graph for which this is the case, i.e. k is the "earliest" (i.e. lowest) level in which the assertion is entailed.

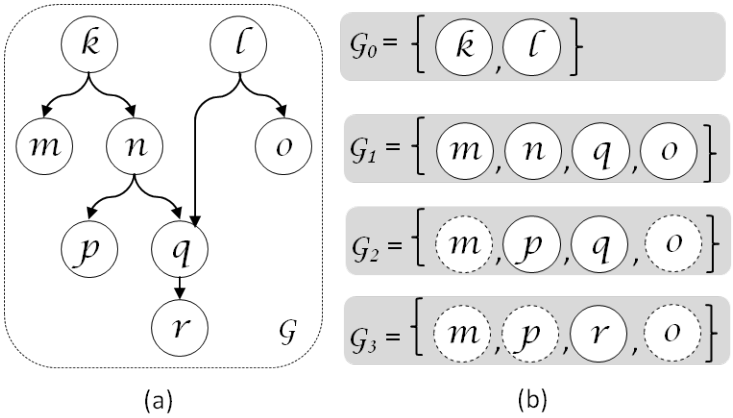

Figure 4.9: A more complex example of k-level goals in a solution graph.

Definition 15. *(k-level entailment)*
Given a valid solution graph $\mathcal{G}$ and a $\mathcal{KB}$, $\mathcal{G}$ entails an assertion α at level k (denoted by $\mathcal{G} \models_k \alpha$), if and only if (1) $\alpha \in Cn(\mathcal{KB} \cup \mathcal{G}_k)$ and (2) $\alpha \notin Cn(\mathcal{KB} \cup \mathcal{G}_{k'})$ for any $k' < k$.

From the definition of a valid solution graph, it follows that if an assertion α is k-level entailed, then α is entailed by all levels i that follow after k. We refer to this as α being a *k-consequence*.

Corollary 1. *(k-consequence)*
Let $depth(\mathcal{G})$ denote the length of the longest path between any root and leaf goal of a solution graph $\mathcal{G}$. Given a valid solution graph $\mathcal{G}$, if $\mathcal{G} \models_k \alpha$ then $\alpha \in Cn(\mathcal{KB} \cup \mathcal{G}_i)$ for any $k \leq i \leq depth(\mathcal{G})$.

Proof:
Observation 1: Given a valid solution graph $\mathcal{G}$, it follows from Definition 15 that if a goal g is on level k and $k < depth(\mathcal{G})$, then either g is on level $k+1$ or the sub-goals of g are on level $k+1$.
Observation 2: It follows from Definition 11 that the sub-goals of any goal g in a valid solution graph must (together with the $\mathcal{KB}$) entail g.

Corollary 1 is false if there exists an assertion α which is entailed by goals on level k, but not entailed by goals on level $k+1$. For this to be true, there must exist a goal g on level k which is not entailed by goals on level $k+1$. However, this cannot be the case since it follows from Observation 1 that either g is on level $k+1$ (and g trivially entails itself), or the sub-goals of g are on level $k+1$, which, following Observation 2,

must entail g.

Given the definition of k-level entailment for a valid solution graph, we now define the notion of strong entailment and strong non-entailment, which allows us to make statements about the derivability of an assertion from the set of solution graphs ($\mathbf{G}$).

Definition 16. *(strong entailment)*
Given a set of valid solution graphs $\mathbf{G}$, $\mathbf{G}$ *strongly entails an assertion* α *(denoted by* $\mathbf{G} \approx \alpha$*), if and only if for all* $\mathscr{G} \in \mathbf{G} : \mathscr{G} \models_k \alpha$ *for some* $k \leq depth(\mathscr{G})$.

Definition 17. *(strong non-entailment)*
Given a set of valid solution graphs $\mathbf{G}$, $\mathbf{G}$ *strongly does not entail an assertion* α *(denoted by* $\mathbf{G} \not\approx \alpha$*), if and only if there does not exist a* $\mathscr{G} \in \mathbf{G}$ *such that* $\mathscr{G} \in \mathbf{G} : \mathscr{G} \models_k \alpha$ *for some* $k \leq depth(\mathscr{G})$.

Note, the strong entailment and strong non-entailment relation resembles skeptical non-monotonic reasoning, but differs in such that the relations are defined over the set of solution graphs, which may not contain all solutions, but only selected ones. In other words, we reason sceptically over selected solutions. Of course, credulous versions could be defined as well.

Following the definition of k-level entailment, we say that two solution graphs agree with each other up to level k, if and only if the derivable at each respective level is identical.

Definition 18. *(k-level agreement)*
Two valid solution graphs $\mathscr{G}$ *and* $\mathscr{G}'$ *agree up to level* k *(denoted by* $agree_k(\mathscr{G}, \mathscr{G}')$*) if and only if* $Cn(\mathscr{G}_i \cup \mathcal{KB}) = Cn(\mathscr{G}'_i \cup \mathcal{KB})$ *for* $0 \leq i \leq k$ *and* $Cn(\mathscr{G}_{k+1} \cup \mathcal{KB}) \neq Cn(\mathscr{G}'_{k+1} \cup \mathcal{KB})$.

To sum up, k-level entailment permits the analysis (e.g. inconsistency checking) of a solution graph at different levels of abstraction. While root goals are the most abstract goals in the sense that they are not a refinement of any other goal, it is worth highlighting that not necessarily all individual goals on level 0 (or on another level) are of the same granularity. For example, in Figure 4.9 the goals **k** and **l** are both on level 0, but goal **k** could be a general statement and goal **l** a very concrete one. While this is not an impediment for analysing a solution graph via the k-level entailment relation, this should be kept in mind when levels are used to establish that some goals are more general than others (as is done in Chapter 5.1 to assess minimal change).[9]

[9] In Chapter 5.1 we argue that general goals should be more reluctant to change than concrete

4.3 Key features of our goal model formalization

In the following, we highlight key features of our goal model formalization. This is done by summarizing how our formalization addresses the deficiencies listed at the beginning of this chapter.

- **F-hypergraph based formalization**: We criticized that the AND/OR graph formalization by Nilsson [124] (which is often referred to) may requires additional "dummy" nodes, if new refinements are added to the AND/OR graph. This might cause confusion if the model is to be presented to stakeholders. The formalization of an AND/OR graph as an acyclic F-hypergraph with n-arry relations between nodes, does not face these deficiencies.

- **Goal library**: We criticized that goals which are removed from an AND/OR graph are permanently lost. Our formalization makes use of a goal library to avoid such permanent loss, i.e. the formalization allows to retain goals in anticipation of future situations where they might be feasible and desirable to achieve.

- **Distinction between desired- and dependent goals:** We criticized that in an AND/OR graph it is not clear whether stakeholders desire the fulfilment of (sub-)goals, regardless of their parent goals. In other words, it is not clear whether (sub)-goals should remain part of the graph after the removal of their parent. This is overcome by clearly distinguishing between goals in the goal library that are "desired" (that stand for themselves) or "dependent" (that cannot stand for themselves).

- **Focus on relevant solutions**: We highlighted that AND/OR graphs nurture the maintenance of irrelevant solutions, since (typically) not all solutions are valid or of interest. Our formalization avoids the maintenance of irrelevant solutions by allowing the maintenance exercise to be restricted to solution graphs that are valid and of interest (i.e. only solution graph in $\mathbf{G}$ are considered).

- **Entailment relation to reason with "derived goals":** We criticized that AND/OR graphs do not support reasoning with "derived goals". This allows

ones. The levels of a solution graph are used to establish how general a goal is - the idea being that a goal closer to level 0 is more general than a goal on a higher level. Goals on the same level are treated equally and hence it is implicitly assumed that they are of the same granularity. However, as pointed out above, this is not necessarily the case. Nevertheless, we believe the level of a goal is a very reasonable proxy of its granularity.

the adoption of solutions that may contain non-compliant or undesired (derived) goals. Our goal model formalization supports reasoning with derived goals via an entailment relation that is associated with each solution graph (i.e. "k-level entailment") and the set of solution graphs (i.e. "strong entailment" and "strong non-entailment"). A highlight of this entailment relation is that it explicitly takes into account the structure of a solution graph. This, amongst others, allows reasoning with derived goals without the need to consider all goals of a solution graph.

- **Inconsistency toleration:** We further criticized that goal models formalized as AND/OR graphs do not support the toleration of inconsistencies at the root level, i.e. root goals (desired goals) are required to be consistent. However, desired goals may not always be consistent, which is particularly true in a multi stakeholder setting. Our representation via a set of solution graphs does not face this problem, since each solution graph is permitted to have different root goals.

4.4 Related Work

This section, juxtaposes our formalization of a goal model with the formalization provided in existing frameworks. Our emphasis will be one the work by Ghose [64, 63], Zowghi et al. [170] and Jureta et al. [87], which, like our work, provide a goal model formalization that is of non-monotonic character and therefore is closest to our work. We will point out similarities and highlight where our formalization differs, while going through the features a requirements/goal maintenance framework should satisfy (extending the original list of features suggested bv Ghose [64]).

Prioritization of Requirements: Ghose [64, 63] and Zowghi et al. [170] support the partitioning of requirements according to their level of priority by distinguishing between essential- and tentative requirements. While essential requirements are unavoidable, i.e. must necessarily be satisfied, tentative requirements may be contracted in the event that they become inconsistent. The same distinction is used in the work by Jureta et al. [87], who refer to these as necessary and optional goals/requirements, respectively. In our framework, we do not explicitly have the notion of an essential requirement (i.e. essential goal) since we believe that ultimately any requirement is of tentative nature. This is particularly true in an evolutionary context, where a changing domain assumption may render a previously essential goal as tentative or even superfluous. Nevertheless, our framework is general enough to treat goals as essential.

In a static context (i.e. where there is no change), it is sufficient to restrict all solution graphs in $\mathbf{G}$ to the ones that contain a goal g, which results in g being satisfied by any considered solution. In a dynamic context, i.e. when a maintenance operator is applied, the selection function of the respective operator is to be encoded in such a way that, in the case that more than one modified set $\{\mathbf{G}'...\mathbf{G}''\}$ satisfies the respective operator properties, a revised set $\mathbf{G}'$ is selected, such that all its solution graphs contain g. This requires the existence of at least one set $\mathbf{G}'$ that satisfies this property. In a case where no such set exists then the respective change request, for example adding a new essential goal, is necessarily in conflict with g. Therefore, one of the goals must lose its essential status, and this can be reflected in the selection function,.

Generation of Maximal Consistent Sets of Requirements: Ghose [64, 63] and Zowghi et al. [170] support the generation of maximal consistent goal/requirements sets (referred to as scenarios in Zowghi et al. [170] and maximal r-consistent sets in Ghose [64, 63]). Jureta et al. [87] supports this feature in a similar manner via the computation of environments in the underlying ATMS system. Our framework does support this feature via the construction[10] of solution graphs. However, as opposed to a maximal scenarios [170] or r-consistent sets [64, 63], a solution graph is hierarchically structured.

Multiple Stakeholder Perspectives: Ghose [64, 63], Zowghi et al. [170] and Jureta et al. [87] support reasoning with multiple potentially inconsistent stakeholder perspectives via the identification of maximal consistent sets of goals/requirements (i.e. solution graphs in our work). Our formalization support this feature in a similar manner. Multiple and potentially inconsistent views are captured via distinct solution graphs. While Ghose [64, 63] and Zowghi et al. [170] construct and reason with *all* maximal consistent sets, this is not a requirement in our approach. For example, we may only generate the preferred solution graph(s), although others may be derivable. In Chapter 5.2, we show how such preference is encoded via a selection function as part of the operators for generating and modifying solution graphs. Roughly, for each operator a *selection function* is encoded to select an answer, i.e. a set of valid solution graphs, amongst the space of all valid answers.

Requirements Reuse: The formalizations given by Zowghi et al. [170], Ghose [64, 63] and Jureta et al. [87] support the reuse of requirements by defining the requirements specification, as a derivable of some larger model, e.g. scenarios are a

[10] In the next chapter, we will describe an abstract operator, i.e. the "completion" operator, that can be used to construct solution graphs from the goal library.

derivable of the THEORIST system. In our formalization of a goal model, this is done in a similar manner via the goal library, i.e. each solution graph in $\mathbf{G}$ is derivable from the goal library and domain knowledge base.

Justification & Structure of Requirements: While the justification and structure of goals/requirements is an essential part of the work by Jureta et al. [87], it is not taken into account in Zowghi et al. [170] and only to some extent in Ghose [64, 63]. While Ghose [64, 63] allows represent justification links between goals/requirement, the formalization permits justification cycles. In our goal model formalizations, justification links are explicitly captured via goal refinement, which do not permit justification cycles, since the refinement/justification links in a solution graph are by definition acyclic. Furthermore, the distinction between desired and dependent goals does not only identify which goals are the root of a justification chain, but is also used to ensure that dependent goals, which have become unjustified, do not remain part of any valid solution graph.

As pointed out earlier, the formalization provided in Zowghi et al. [170] and Ghose [64, 63] is of non-monotonic character and shares many similarities with the default theory formalism proposed by Reiter [134]. In Antoniou et al. [6], the authors study the refinement of default theories and define under which conditions a default theory is a refinement of another default theory. This is non-trivial, since a default theory may have multiple extensions. The authors propose to handle this via sceptical- and credulous refinements. Roughly, a default theory T' is a sceptical (credulous) refinement of another theory T if every sentence that can be sceptically (credulously) derived from T can also be sceptically (credulously) derived from T', where T' may contain additional sentences. In principle, the work by Antoniou et al. [6] may be used to introduce structure to the formalization provided bv Zowghi et al. [170] and Ghose [64, 63], by relating various default theories (or THEORIST systems) with each other. However, this would not allow us to consider refinement relations between individual goals/requirements.

In the work by MacNish and her co-authors [113, 38] goals are structured via a partial ordering, such that goals with a lower rank refine goals with a higher rank. As acknowledged by the authors, their approach cannot handle multi-rooted trees, nor goals that are shared by multiple refinements. Our formalization of a solution graph, based on F-hypergraphs [50], permits more than one root goal and goals to be part of more than one refinement.

Language Agnosticism The formalization of the goal model proposed in this thesis is

language agnostic, i.e. goals can be expressed in propositional logic or more expressive temporal or predicate logic. This is a key distinction from the Techne [87], where goals and domain assumptions are represented as atomic statements. While the former is sufficient to (abstractly) capture natural language sentences (as pointed out in [87]), it requires the requirements engineer to *manually* establish relationships between the sentences (i.e. the requirements engineer is assumed to act as the inference engine). Since there can be a very large number of these relationships (in particular conflict relationships), manually establishing these relationships is not only laborious, but may also result in overlooked or erroneous relations. This is particularly problematic when these relationships need to be maintained. In principle, this could be resolved by treating sentences of a more expressive language as atoms and using a language specific inference engine to infer the relationships between them. While this would address the above criticism, the computation of all justification relationships by the external inference engine would imply additional computational expenses and undo the computational advantages pointed out by the authors.

Entailment Relation Frameworks that permit the specification of goals/requirements in a formal language, in principle, permit the association of an entailment relation with the goal/requirements model. The work by Zowghi et al. [170] explicitly discusses the benefits of an associated entailment relation. Ghose [64, 63] and Nuseibeh and his co-authors [126, 82] also make references to an entailment relation. The entailment relation proposed in our approach (i.e. k-level entailment) differs from existing work by explicitly taking into account the structure of a solution graph to reason about derived goals at each level. This has various advantages. On the one hand, to check whether an assertion is derivable from a solution graph not all goals of the solution graph have to be considered, i.e. it is sufficient to only consider the goals on the highest level with respect to k (i.e. the leaf goals). This is important since satisfiability checking is expensive (e.g. it is NP-complete for a propositional logic [139]) and the time required to compute an answer can be significantly decreased by decreasing the input size (i.e. the number of goals). On the other hand, it allows us to identify the earliest level in which a particular assertion is entailed in a solution graph. This, again, has computational advantages, when finding a minimal revised solution graph that does not entail a particular assertion. We make use of these advantages in our prototype implementation, which is discussed in Chapter 7.

Finally, one may argue that our notion of k-level goals resembles the partial ordering defined over elements in the work by MacNish and her co-authors [38, 113].

While both approaches leverage the goal model structure to reason over elements at different levels of abstract, the partial ordering does not (per definition) permit goals to be part of more than one "level". As a consequence, the approach by MacNish and her co-authors [38, 113] does not permit reasoning with multi-rooted structures, as well as goals which are part of more than one refinement - both situations may allow a goal to be part of more than one level.

4.5 Summary

This chapter introduced a new goal model representation which was shown to have advantages when the model requires maintenance.

In the first part of this chapter (Chapter 4.1), we highlighted issues in goal models, formalized as AND/OR graphs, when they need to evolve. In particular, we identified the following issues:

- Adding new refinements to an AND/OR graph may require the addition of "dummy" nodes, which can cause confusion when the model is discussed with stakeholders.
- Goals which are removed from an AND/OR graph are permanently lost.
- AND/OR graphs do not allow the capture of all sources of justification for a goal. This permits situations in which it is not clear whether sub-goals should be removed or retained after the removal of their parent goal.
- AND/OR graphs nurture the maintenance of irrelevant solutions, which is computationally expensive.
- AND/OR graphs do not support reasoning with derived goals, which allows the adoption of "solutions" with non-compliant or undesired (derived) goals.
- AND/OR graphs do not tolerate inconsistencies at the root level, which nurtures premature commitments.

In the second part of this chapter (Chapter 4.2) we introduced a new goal model formalization which was shown (in Chapter 4.3) to address the identified issues. In particular, the presented formalization has the following advantages:

- An *F-hypergraph* based formalization was shown to avoid the need for "dummy" nodes.

- A *goal library* was deployed to avoid the permanent loss of goals.
- A distinction between *desired- and dependent goals* clearly identifies which sub-goals to retain after the removal of their parent goal.
- Stakeholder intention was captured via a set of *solution graphs*. While the set of solution graphs was shown to be general enough to represent AND/OR graphs, this also allows us to focus the maintenance exercise on relevant solutions and support the representation and reasoning with different (potentially inconsistent) root goals.

Chapter 5

Goal Model Maintenance

Today's dynamic business context requires organisations/stakeholders to constantly maintain their goal model. Similarly to classical software maintenance [149], the maintenance of goal models can generally be classified in terms of distinguishing the purposes of three activities: *perfective*, *adaptive* and *corrective* maintenance. Corrective maintenance introduces changes to remove errors in a goal model, e.g. one or more solution graphs in $\mathbf{G}$ may have become non well-formed due to, for example, a preceding revision of domain knowledge. Adaptive maintenance involves changing the goal model so that it can continue to conform to a changed environment, for instance modifying $\mathbf{G}$ to meet new compliance obligations. Finally, perfective maintenance aims to change a goal model resulting from changes in the stakeholders' motivation.

We now propose a set of operators that can be used to change an existing goal model to meet those maintenance purposes. The proposed operators (and their properties) are inspired by the belief revision literature. The operators are particularly influenced by the AGM[1] logic of theory change [1], which studies the rationale of knowledge revision. Previous work [113], [170] have applied these results to provide guidance in the revision/modification of other knowledge artifacts (e.g. requirement models). A key intuition provided by the AGM theory is that change should be incorporated in a manner that minimizes the impact to the original knowledge artifact. The proposed operators follow this intuition of minimal change. We therefore require means to assess the extent of change between two goal graphs.

[1] AGM stands for Alchourrón, Gärdenfors, and Makinson.

5.1 Solution Graph Proximity

We assess minimal change by the relative distance of two alternative modified solution graphs with respect to the original one. Given the solution graphs $\mathscr{G}$, $\mathscr{G}'$ and $\mathscr{G}''$ we say that $\mathscr{G}' <_{\mathscr{G}} \mathscr{G}''$ if $\mathscr{G}'$ is "closer" to $\mathscr{G}$ than $\mathscr{G}''$. The relation $<_{\mathscr{G}}$ may be defined in various ways and our proposed framework is not required to follow a particular one. In the following, we provide a step by step motivation of a definition of proximity that we believe is sensible.

From a pure *graph perspective*, it can be said that the more vertices and edges $\mathscr{G}'$ has in common with $\mathscr{G}$, the closer it is to $\mathscr{G}$. This places an equal weighting on all elements of the graph.

Example. *Figure 5.1 (a), (b) and (c) show the solution graphs $\mathscr{G}'$, $\mathscr{G}$ and $\mathscr{G}''$, respectively. For now, we omit the dashed refinement e5 and goal* **c** *in Figure 5.1 (c). From a pure graph perspective, the graphs $\mathscr{G}'$ (Figure 5.1 (a)) and $\mathscr{G}''$ (Figure 5.1 (c)) are equally close to graph $\mathscr{G}$ (Figure 5.1 (b)), since they share the same number of goals and refinement links (i.e. edges) with $\mathscr{G}$.*

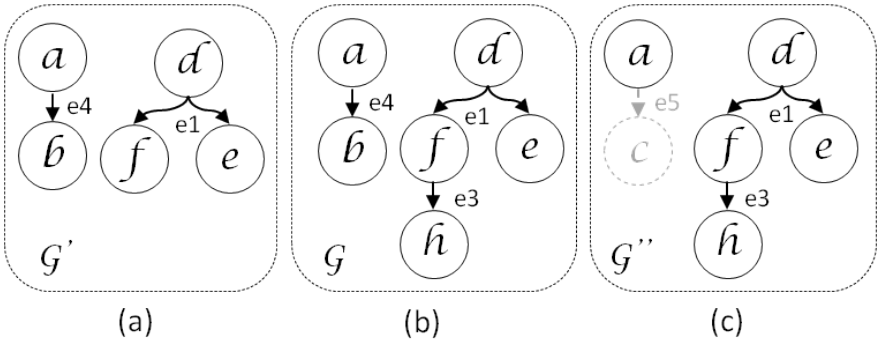

Figure 5.1: Example of solution graph proximity.

However, this is not intuitive in the context of a *solution graph*. A solution graph, above all, denotes a particular choice to a set of *desired goals* and therefore it is sensible to give precedence to them. More precisely, we may prefer a modified solution graph $\mathscr{G}'$ over $\mathscr{G}''$, if $\mathscr{G}'$ has more *desired goals in common* with the original graph $\mathscr{G}$ than $\mathscr{G}''$. We refer to this notion of minimal change as *"Desire Proximity"*.

Definition 19. *(Desire Proximity)*
Given graphs $\mathscr{G}$, $\mathscr{G}'$ and $\mathscr{G}''$, $\mathscr{G}' <_{\mathscr{G}} \mathscr{G}''$, if and only if $|(f_{des}(\mathscr{G}) \cap f_{des}(\mathscr{G}''))| < |(f_{des}(\mathscr{G}) \cap f_{des}(\mathscr{G}'))|$.

Example. *In Figure 5.1, the solution graphs $\mathcal{G}'$ and $\mathcal{G}''$ are equally preferred according to "Desire Proximity", since they share the same number of desired goals with $\mathcal{G}$.*

A solution graph is not only a choice of a particular set of desired goals, but also a choice of a particular refinement of these goals. The similarity between two graphs with respect to their refinements can be assessed at different levels of granularity, i.e. it may be assessed by the number of dependent goals the two graphs have in common, or alternatively, by the number of refinement relationships (i.e. hyper-edges between goals) they share. We believe the latter is more truthful to the 'stakeholders' original goal refinement choice, which is with respect to a particular set of (sub-)goals and not to individual dependent goals.

Example. *In Figure 5.1, the solution graphs $\mathcal{G}'$ and $\mathcal{G}''$ share the same number of desired goals with the original graph $\mathcal{G}$, and both $\mathcal{G}'$ and $\mathcal{G}''$ have the same number of refinements in common with $\mathcal{G}$ (the graph $\mathcal{G}'$ shares the hyper-edge e_1 and e_4 with $\mathcal{G}$, while $\mathcal{G}''$ shares the hyper-edges e_1 and e_3 with $\mathcal{G}$). According to the above argument they are again equally preferred.*

This places an equal weighting on all refinements. However, we believe that refinements on a lower level (with respect to k), i.e. which are closer to the stakeholders root goals, should be given precedence over refinements on a higher level (with respect to k). This view is also supported by MacNish and Williams [113] and Bell and Huang [15], who argue that goals closer to the essence of stakeholder ambition (i.e. root goals) should be given precedence over lower-level concerns by being considered as more reluctant to change.

Example. *Figure 5.1, $\mathcal{G}'$ and $\mathcal{G}''$ have the same number of refinements in common with $\mathcal{G}$, but $\mathcal{G}'$ shares more refinements on a lower level (with respect to k) with $\mathcal{G}$ than $\mathcal{G}''$. More precisely, $\mathcal{G}'$ shares two refinement of goals at level zero with $\mathcal{G}$, while $\mathcal{G}''$ shares one refinement with $\mathcal{G}$ of goals at level zero. Following the above argument, $\mathcal{G}'$ should be preferred.*

Furthermore, a minimally modified solution graph should not be an alteration of the stakeholders choice of refinement, i.e. it should stay true to the 'stakeholders' original choice of goal refinement and not contain any refinements which were not part of the original graph.

Example. *Consider the dashed goal* **c** *and refinement e_5 to be part of the solution graph $\mathcal{G}''$ shown Figure 5.1 (c). Observe, refinement e_5 is not part of the original solution*

graph $\mathcal{G}$ and therefore $\mathcal{G}''$ captures a different *choice of goal refinement. Consequently, $\mathcal{G}''$ should not be considered as a minimal modification of the original solution graph $\mathcal{G}$.*

The above is formally captured as follows. We say that a refinement is on level k if k is the earliest level at which the refinement's source goal appears. We use $f_{E_k}(\mathcal{G})$ to denote all refinements on level k in the solution graph $\mathcal{G}$. We further say, a solution graph $\mathcal{G}'$ is preferred over $\mathcal{G}''$ *at level* k, if $\mathcal{G}'$ includes more refinements of $\mathcal{G}$ at level k than $\mathcal{G}''$. A solution graph $\mathcal{G}'$ is preferred over $\mathcal{G}''$ (with respect to $\mathcal{G}$), if it is preferred at level k, k is the earliest level at which it is preferred, and $\mathcal{G}''$ is not preferred at any level before k.

Definition 20. *(Refinement Proximity)*
Given the solution graphs $\mathcal{G}$, $\mathcal{G}'$ and $\mathcal{G}''$, $\mathcal{G}' <_{\mathcal{G}} \mathcal{G}''$, if and only if

- *$\mathcal{G}''$ has distinct refinements from $\mathcal{G}$ and $\mathcal{G}'$ does not have distinct refinement from $\mathcal{G}$ (i.e.$(f_E(\mathcal{G}'') \nsubseteq f_E(\mathcal{G})) \wedge (f_E(\mathcal{G}') \subseteq f_E(\mathcal{G}))$) or*
- *$\mathcal{G}'$ has no distinct refinements from $\mathcal{G}$ and $\mathcal{G}'$ is preferred at level k, k is the earliest level at which it is preferred, and $\mathcal{G}''$ is not preferred at any level before k.*

We can combine different intuitions of minimal change via a lexicographic ordering. In other words, we may *strictly* prefer a graph $\mathcal{G}'$ over $\mathcal{G}''$ if $\mathcal{G}'$ is closer with respect to *Desire Proximity* (i.e. has more desired goals in common with $\mathcal{G}$ than $\mathcal{G}''$). In case $\mathcal{G}'$ and $\mathcal{G}''$ are equally preferred with respect to "Desire Proximity", then we may prefer $\mathcal{G}'$ over $\mathcal{G}''$ if $\mathcal{G}'$ is closer with respect to *Refinement Proximity*.

Definition 21. *(Goal Graph Proximity)*
Given the original solution graph $\mathcal{G}$, let $<_{\mathcal{G}}$ be a lexicographic ordering defined upon (1) "Desire Proximity" and (2) "Refinement Proximity", in decreasing order.

Observe, "Goal Graph Proximity" assesses the distance between solution graphs, based on their *structure*, but does not directly take into account the "*meaning*" of the goals given by the interpretation of their formulations in logic. This permits situations where two solution graphs are structurally different, but have the exact same derivable, i.e. their formulation is logically equivalent. We could also define the proximity relation $<_{\mathcal{G}}$ based on the "semantic proximity" between two solution graphs, leveraging k-level agreement. More precisely, we could define that a solution graph $\mathcal{G}'$

is preferred over a solution graph $\mathscr{G}''$, if $\mathscr{G}'$ agrees with the original solution graph $\mathscr{G}$ on more levels than $\mathscr{G}''$.

We believe proximity defined by *Goal Graph Proximity* intuitively captures the distance between solution graphs and we will use it throughout the remainder of this thesis. However, other intuitions of proximity can be defined and the proposed framework is not prescriptive about the one to be followed.

5.2 Maintenance Operators

This section introduces a family of goal model maintenance operators, i.e. a *"strong add desired goal"*-, *"weak add desired goal"*-, *"weak remove desired goal"*-, *"not entail assertion"*-, *"repair"*- and *"completion"* operator.

As new stakeholder desires arise, new (desired) goals have to be added and the goal model has to be maintained accordingly. Given the emergence of a *new* desired goal g, we consider a *strong* and *weak* case. Roughly, in the strong case g is added as a desired goal to *all* valid solution graphs in $\mathbf{G}$, while in the weak case it is ensured that all elements in $\mathbf{G}$ remain valid solution graphs when $g \in LIB^{des}$ (i.e. g is added to all valid solution graphs if consistently possible). Note, we do not consider an operator for adding a *dependent goal* to solution graphs in $\mathbf{G}$, since the choice to add a goal presumes a *desire* for the goal to be brought about.

5.2.1 "Strong add desired goal" Operator

The "*strong add desired goal*" operator is used to add a goal g as a *desired* goal to *all* valid solution graphs in $\mathbf{G}$ (denoted by $\mathbf{G} \otimes_s g$). Given a goal model $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$, the set of solution graphs returned by the operator satisfies the following properties.[2]

(1) *Validity:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \otimes_s g = \mathbf{G}'$ it must be the case that all $\mathscr{G}' \in \mathbf{G}'$ are valid solution graphs.

(2) *Success:* For a set of valid* solution graphs $\mathbf{G}$, if $\bot \notin Cn(\{g\} \cup \mathcal{KB})$ and $\mathbf{G} \otimes_s g = \mathbf{G}'$ it must be the case that all $\mathscr{G}' \in \mathbf{G}' : g \in f_{des}(\mathscr{G}')$.

(3 *Vacuity:* For a set of valid* solution graphs $\mathbf{G}$ and all $\mathscr{G}' \in \mathbf{G} : g \in f_{des}(\mathscr{G}')$ it must be the case that $\mathbf{G} \otimes_s g = \mathbf{G}$.

[2] Recall, that we use "valid*" as a "wildcard" for different intuitions of solution graph validity. For now, valid* can simply be substituted by solution graph validity as defined in Definition 11.

(4) *Minimal-change:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \otimes_s g = \mathbf{G}'$ it must be the case that for each $\mathscr{G} \in \mathbf{G}$ there exists a $\mathscr{G}' \in \mathbf{G}'$ where $g \in f_{des}(\mathscr{G}')$, and there does not exist a $\mathscr{G}''$ such that $g \in f_{des}(\mathscr{G}'')$ and $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$, and for each $\mathscr{G}' \in \mathbf{G}'$ there exists a $\mathscr{G} \in \mathbf{G}$ and there does not exists a $\mathscr{G}''$ where $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$.

The first property states that each solution graph in $\mathbf{G}$ should remain valid. The operator is successful if g is a desired goal of all solution graphs in $\mathbf{G}$. It should however be highlighted that the success property demonstrates the condition that g must be consistent with the $\mathcal{KB}$. Otherwise any solution graph which contains a goal that is inconsistent with $\mathcal{KB}$ is necessarily non-valid. In the case that g is already part of a solution graph, but is not desired, its addition as a desired goal is to be understood as "promoting" g to be a desired goal. The third property states that the operator should be vacuous in a case where the success property is already satisfied, i.e. no change should be made. The last property states that the change for each valid solution graph in $\mathbf{G}$ should be minimal. In other words, if there exists more than one way to bring about the change for a valid solution graph, then we commit to the one with the least impact. The extent of "impact" is assessed by the proximity relation $<_G$ and can follow different intuitions. Given these properties the *strong add desired goal* ($\otimes_s$) operator is formally defined as follows.

Definition 22. *Given a well-formed goal model* $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$ *and a goal* g*, let the operator* $\mathbf{G} \otimes_s^* g$ *return a set (of sets of valid solution graphs), such that each* $\mathbf{G}' \in \mathbf{G} \otimes_s^* g$ *satisfies the properties (1)-(4). We define* $\mathbf{G} \otimes_s g = s(\mathbf{G} \otimes_s^* g)$*, where* s *is a function that makes a selection from elements in* $\mathbf{G} \otimes_s^* g$*.*

There can exist more than one set $\mathbf{G}'$ that satisfies the above properties (1-4). This is the case when there exists more than one successful and minimal modification (with respect to $<_G$) of the valid solution graphs of the original set $\mathbf{G}$. In Definition 22, the $\otimes_s^*$ operator returns the set of *all* $\mathbf{G}'$ that satisfy properties (1)-(4) and $\otimes_s$ is a *selection function* that selects one of the elements. The selection function can be used to encode additional stakeholder preferences.

The $\otimes_s$ operator is an operator class, where the operator can be instantiated by instantiating the proximity relation $<_G$ (e.g. using "Goal Graph Proximity" as defined in Chapter 5.1), as well as instantiating the selection function s.

Example. *Given are an initial goal model with a single solution graph as shown on the left-hand side of Figure 5.2 (i.e.* $\mathbf{G} = \{\mathscr{G} = \langle \{a, i\}, \{\} \rangle\}$*), the desired goals* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{i}\}$ *and* $\mathcal{KB} = \{dk_1, dk_2\}$*.*

The stakeholders have decided that in all solutions to be considered, healthcare professionals are dispatched to patients in need of medical help, i.e. the goal $\boldsymbol{d}$ *should be part of each solution graph in* $\mathbf{G}$*. This change request can be implemented via an application of the "strong add desired goal operator". In the case that the operator class is instantiated by "Goal Graph Proximity" (see Definition 21), then* $\{\{\mathscr{G}_{1.1}\}, \{\mathscr{G}_{2.1}\}, \{\mathscr{G}_{3.1}, \mathscr{G}_{3.2}\}\}$ *is the outcome of* $\mathbf{G} \otimes_s^* \boldsymbol{b}$*. Each of these sets of solution graphs is a valid outcome of* $\mathbf{G} \otimes_s \boldsymbol{b}$*, since the solution graphs are valid, have* $\boldsymbol{d}$ *as a desired goal, and satisfy the minimal change criteria. Note, the selection function s can be used to discriminate amongst these otherwise indistinguishable solutions.*

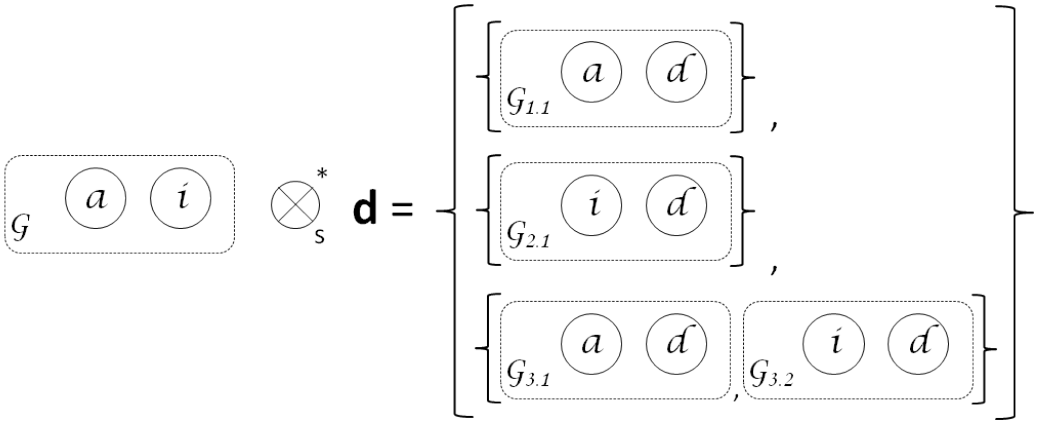

Figure 5.2: Example of the "strong add desired goal" operator.

The operator outcome can have fewer elements than the original set of solution graphs. This is the case when the modification of two distinct solution graphs results in the same solution graph (recall that a set does not contain duplicates).

Example. *Given are an initial goal model with two solution graphs as shown on the left-hand side of Figure 5.3 (i.e.* $\mathbf{G} = \{\mathscr{G}_{0.1}, \mathscr{G}_{0.2}\}$*), the desired goals* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{i}, \boldsymbol{d}\}$ *and* $\mathcal{KB} = \{dk_1, dk_2\}$*.*

Observe, that $\boldsymbol{d}$ *is already a desired goal of the solution graph* $\mathscr{G}_{0.2}$ *(but not part of all graphs in* $\mathbf{G}$*) and strongly adding* $\boldsymbol{d}$ *to all solution graphs can be understood as giving* $\boldsymbol{d}$ *precedence over all other desired goals, as other desired goals may have to be removed to consistently add* $\boldsymbol{d}$*. Let proximity be instantiated by "Goal Graph Proximity" then* $\{\{\mathscr{G}_{1.1}, \mathscr{G}_{1.2}\}, \{\mathscr{G}_{2.1}\}\}$ *is the outcome of* $\mathbf{G} \otimes_s^* \boldsymbol{d}$*. Observe, if we select* $\{\mathscr{G}_{2.1}\}$ *as an outcome of the "strong add desired goal" operator, then the set of solution graphs has fewer elements than before the change.*

Note, the set of solution graphs $\{\langle\{i, d\}, \{\}\rangle\}$ *contains a solution graph which is a minimal modification of* $\mathscr{G}_{0.1}$*, but not a minimal modification of* $\mathscr{G}_{0.2}$ *(as* $\mathscr{G}_{0.2}$ *already contains* $\boldsymbol{d}$ *as a desired goal, its minimal modification is* $\mathscr{G}_{0.2}$ *itself) and therefore not a correct outcome.*

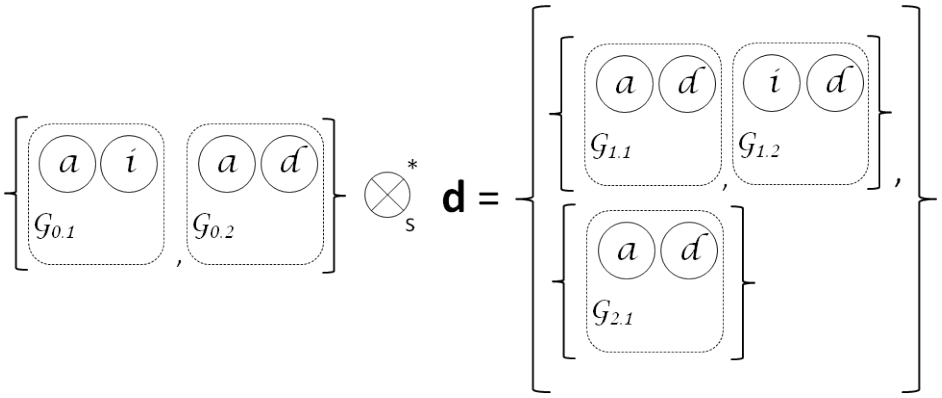

Figure 5.3: Another example of the "strong add desired goal" operator.

5.2.2 "Weak add desired goal" Operator

We use the "weak add desired goal" operator to add a *desired goal* to solution graphs in $\mathbf{G}$ if it can be added without causing inconsistencies with desired goals of the respective solution graph. Given a well-formed goal model $\mathbf{M} = \langle \mathbf{G}, LIB, \mathcal{KB} \rangle$, the set of solution graphs returned by the operator satisfies the following properties.

(1) *Validity:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \otimes_w g = \mathbf{G}'$ it must be the case that $\mathbf{G}'$ is valid*.

(2) *Success:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \otimes_w g = \mathbf{G}'$ it must be the case that for each $\mathscr{G}' \in \mathbf{G}'$:, $g \in f_{des}(\mathscr{G})$ unless $\bot \in Cn(\mathcal{KB} \cup f_{des}(\mathscr{G}) \cup \{g\}$.

(3 *Vacuity:* For a set of valid* solution graphs $\mathbf{G}$, if for all $\mathscr{G} \in \mathbf{G}$ either $g \in f_{des}(\mathscr{G})$ or $\bot \in Cn(\mathcal{KB} \cup f_{des}(\mathscr{G}) \cup \{g\}$, then$\mathbf{G} \otimes_w g = \mathbf{G}$.

(4) *Minimal-change:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \otimes_w g = \mathbf{G}'$ it must be the case that for each $\mathscr{G} \in \mathbf{G}$ there exists a $\mathscr{G}' \in \mathbf{G}'$ where $g \in f_{des}(\mathscr{G}')$ or $\bot \in Cn(\mathcal{KB} \cup f_{des}(\mathscr{G}') \cup \{g\}$ and there does not exist a $\mathscr{G}''$ such that $g \in f_{des}(\mathscr{G}'')$ or $\bot \in Cn(\mathcal{KB} \cup f_{des}(\mathscr{G}'') \cup \{g\}$ and $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$, and for each $\mathscr{G}' \in \mathbf{G}'$ there exists a $\mathscr{G} \in \mathbf{G}$ and there does not exists a $\mathscr{G}''$ where $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$.

Given these properties the *weak add desired goal* ($\otimes_w$) operator is formally defined as follows.

Definition 23. *Given a well-formed goal model* $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$ *and a goal* g*, let the operator* $\mathbf{G} \otimes_w^* g$ *return a set (of sets of valid solution graphs), such that each* $\mathbf{G}' \in \mathbf{G} \otimes_w^* g$ *satisfies the properties (1)-(4). We define* $\mathbf{G} \otimes_w g = s(\mathbf{G} \otimes_w^* g)$*, where* s *is a function that makes a selection from elements in* $\mathbf{G} \otimes_w^* g$*.*

Example. *Given a goal model with two solution graphs as shown on the left-hand side of Figure 5.4 (i.e.* $\mathbf{G} = \{\mathcal{G}_{0.1}, \mathcal{G}_{0.2}\}$*), the desired goals* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{d}\}$*,* $\mathcal{KB} = \{dk_1, dk_2\}$ *and proximity instantiated by "Goal Graph Proximity", then* $\{\mathcal{G}_{1.1}, \mathcal{G}_{1.2}\}$ *is the outcome of weakly adding a goal* $\boldsymbol{i}$*.*

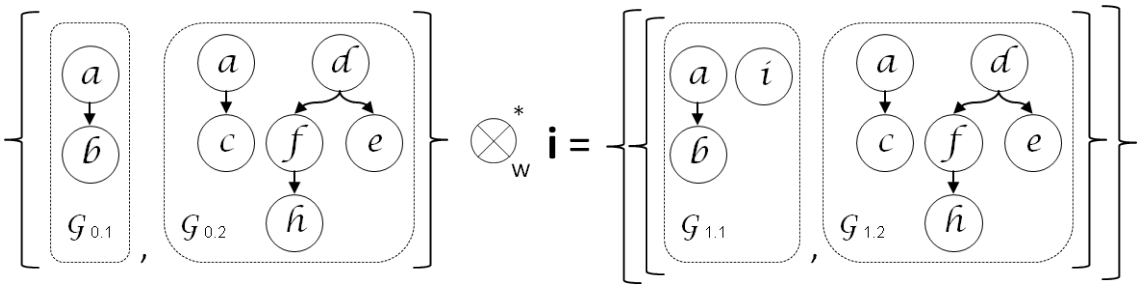

Figure 5.4: Example of the "weak add desired goal" operator.

The goal $\boldsymbol{i}$ *can be consistently added (as a desired goal) to* $\mathcal{G}_{0.1}$ *resulting in the revised solution graph* $\mathcal{G}_{1.1}$*. However, goal* $\boldsymbol{i}$ *is not satisfiable together with the desired goals* $\boldsymbol{a}$ *and* $\boldsymbol{d}$ *of the solution graph* $\mathcal{G}_{0.2}$*. Following the operators' success property, it therefore cannot be added, i.e. the solution graph* $\mathcal{G}_{0.2}$ *remains unchanged. Note, if the conflict would have been with a dependent goal of the solution graph* $\mathcal{G}_{0.2}$*, then the validity and success property of the operator would have required that* $\boldsymbol{i}$ *is added (as a desired goal) and the conflicting dependent goal to be removed. In other words, the goal to be added is always given precedence over all dependent goals in a solution graph.*

5.2.3 "Weak remove desired goal" Operator

We also require an operator to handle the case when stakeholders cease to desire a goal. A goal that the stakeholders have ceased to desire should be removed from the solution graphs *unless* it helps to satisfy a desired goal. The "weak remove desired goal" operator removes a given goal g from the desired goals of each solution graph (denoted by $\mathbf{G} \oslash g$). In other words, the operator demotes a goal g to a dependent

goal. Given a well-formed goal model $\mathbf{M} = \langle \mathbf{G}, LIB, \mathcal{KB} \rangle$, the set of solution graphs returned by the operator satisfies the following properties.

(1) *Validity:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \oslash g = \mathbf{G'}$ it must be the case that $\mathbf{G'}$ is valid.

(2) *Success:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \oslash g = \mathbf{G'}$ it must be the case that for each $\mathscr{G}' \in \mathbf{G'}$, $g \notin f_{des}(\mathscr{G}')$.

(3) *Vacuity:* For a set of valid* solution graphs $\mathbf{G}$, where for each $\mathscr{G} \in \mathbf{G}$ $g \notin f_{des}(\mathscr{G})$, it must be the case that $\mathbf{G} \oslash g = \mathbf{G}$.

(4) *Minimal-change:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \oslash g = \mathbf{G'}$ it must be the case that for each $\mathscr{G} \in \mathbf{G}$ there exists a $\mathscr{G}' \in \mathbf{G'}$ where $g \notin f_{des}(\mathscr{G}')$, and there does not exist a $\mathscr{G}''$ such that $g \notin f_{des}(\mathscr{G}'')$ and $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$, and for each $\mathscr{G}' \in \mathbf{G'}$ there exists a $\mathscr{G} \in \mathbf{G}$ and there does not exist a $\mathscr{G}''$ where $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$.

Note, demoting a goal g to a dependent goal may require additional removal of goals from the respective solution graph. For example, this is the case when g is a root goal and all its sub-goals are dependent goals and do not participate in a refinement of a desired goal. Also, demoting a goal g to a dependent goal may require additional *desired goals* to be added to the solution graphs, if consistently possible.

Definition 24. *Given a well-formed goal model $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$ and a goal g, let the operator $\mathbf{G} \oslash^* g$ return a set (of sets of valid solution graphs), such that each $\mathbf{G'} \in \mathbf{G} \oslash^* g$ satisfies the properties (1)-(4). We define $\mathbf{G} \oslash g = s(\mathbf{G} \oslash^* g)$, where s is a function that makes a selection from elements in $\mathbf{G} \oslash^* g$.*

Example. *Given a goal model with two solution graphs as shown on the left-hand side of Figure 5.5 (i.e. $\mathbf{G} = \{\mathscr{G}_{0.1}, \mathscr{G}_{0.2}\}$), the desired goals $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{d}\}$, $\mathcal{KB} = \{dk_1, dk_2\}$ and proximity instantiated by "Goal Graph Proximity", then $\{\mathscr{G}_{1.1}, \mathscr{G}_{1.2}\}$ is the outcome of weakly removing $\boldsymbol{d}$.*

5.2.4 "Not entail assertion" Operator

The "not entail assertion" operator minimally modifies the solution graphs in $\mathbf{G}$ such that a particular assertion (from now on we refer to this assertion as α) is strongly non-derivable (denoted by $\mathbf{G} \ominus \alpha$). In other words, α is not derivable from any solution graph in $\mathbf{G}$.

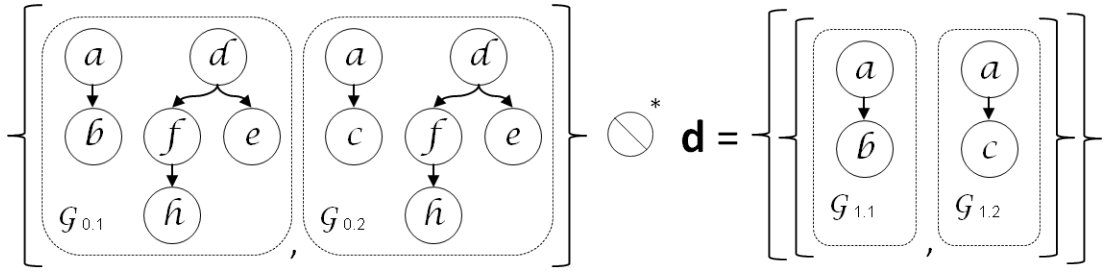

Figure 5.5: Example of the "weak remove desired goal" operator.

Before the operator properties are discussed, we give an extended definition of solution graph validity, which extends condition (3) and (4) of Definition 11 by requiring goals of a solution graph to not only be consistent with the $\mathcal{KB}$ but also be consistent with the negation of *alpha* (where α is the assertion to be strongly non-derivable by an application of the "not entail assertion" operator). We refer to this as *alpha-exclusion* (α-exclusion) and define it as follows.

Definition 25. *(α-exclusive solution graph)*
Given an assertion α, a solution graph $\mathscr{G} = \langle V, E \rangle$ is α-exclusive with respect to a goal library LIB, a domain knowledge base $\mathcal{KB}$ and the assertion α, if and only if:

(1) $V \subseteq LIB$ (library set membership) and

(2) for all $(e = (g, S)) \in E$ it holds that $g \in Cn(S \cup KB)$ and $g \notin Cn(S' \cup KB)$ for any $S' \subset S$ (correct refinements) and

(3) $\bot \notin Cn(V \cup KB \cup \{\neg\alpha\})$ (α-consistency) and

(4) there does not exist a set $DES \subseteq LIB^{des}$, such that $f_{des}(\mathscr{G}) \subset DES$ and $\bot \notin Cn(\mathcal{KB} \cup DES \cup \{\neg\alpha\})$ (α-desire maximality) and

(5) $f_{root}(\mathscr{G}) \subseteq LIB^{des}$ (desired root goals) and

(6) for all $(e = (g, S)) \in E$, there does not exist a $g' \in LIB$, where $g' \neq g$ and a set $S' \subset S$, such that $g' \in Cn(S' \cup KB)$ (immediate refinements).

Given a goal model $\mathbf{M} = \langle \mathbf{G}, LIB, \mathcal{KB} \rangle$, the set of solution graphs returned by the "not entail assertion" operator satisfies the following properties.

(1) *Validity:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \ominus \alpha = \mathbf{G'}$, if $\alpha \notin Cn(\mathcal{KB})$ then it must be the case that all $\mathscr{G}' \in \mathbf{G'}$ are α-exclusive solution graphs.

(2) *Success:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \ominus \alpha = \mathbf{G}'$, if $\alpha \notin Cn(\mathcal{KB})$ then it must be the case that $\mathbf{G}' \not\models \alpha$.

(3) *Vacuity:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \not\models \alpha$, it must be the case that $\mathbf{G} \ominus \alpha = \mathbf{G}$.

(4) *Minimal-change:* For a set of valid* solution graphs $\mathbf{G}$ and $\mathbf{G} \ominus \alpha = \mathbf{G}'$, it must be the case that for each $\mathscr{G} \in \mathbf{G}$ there exists a $\mathscr{G}' \in \mathbf{G}'$, such that $\alpha \notin Cn(\mathscr{G}' \cup \mathcal{KB})$ and there does not exist an α-exclusive solution graph $\mathscr{G}''$ where $\alpha \notin Cn(\mathscr{G}'' \cup \mathcal{KB})$ and $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$, and for each $\mathscr{G}' \in \mathbf{G}'$ there exists a $\mathscr{G} \in \mathbf{G}$ and there does not exist a $\mathscr{G}''$ where $\mathscr{G}'' <_{\mathscr{G}} \mathscr{G}'$.

The first two properties require that all solution graphs in $\mathbf{G} \ominus \alpha$ are valid and do not entail α, i.e. all solution graphs must be α-exclusive. Note, this requires that α is not derivable from the domain knowledge base, as otherwise there cannot exist an α-exclusive solution graph.[3] The operator is defined as follows.

Definition 26. *Given a well-formed goal model* $\langle \mathbf{G}, LIB, \mathcal{KB} \rangle$ *and an assertion* α*, let the operation* $\mathbf{G} \ominus^* \alpha$ *return a set (of sets of solution graphs), such that each* $\mathbf{G}' \in (\mathbf{G} - \alpha)$ *satisfies the properties (1)-(4). We define* $\mathbf{G} \ominus \alpha = s(\mathbf{G} \ominus^* \alpha)$*, where* s *is a function that makes a selection from elements in* $\mathbf{G} \ominus^* \alpha$*.*

The operator is interesting in various maintenance scenarios:

(1) The operator is useful in situations where it needs to be ensured that particular states of affairs are not actively pursued. An interesting instance of this is goal model compliance. Recent publications show (e.g. [145, 55, 83]) that compliance should not only be a concern during system design and execution, but also at the earliest point of system development, i.e. during goal-oriented requirements engineering. A goal model (i.e. a set of valid solution graphs) might be deemed to be non-compliant due to (a) explicitly stated assertions; or (b) assertions that are entailed by the solution graphs. The latter is particularly interesting as it requires a notion of entailment associated with the goal model (we have such an entailment relation i.e. k-level entailment). Given a non-compliant state of affairs, denoted by the assertion α, the "not entail assertion" operator allows us to identify modification to the goal model that makes the goal model compliant. There are also other instances (besides compliance) where stakeholders want to ensure that particular states of affairs are not actively pursued. Consider the following example.

[3] In such a situation, the $\mathcal{KB}$ can be revised using "vanilla" belief revision techniques.

Example. *Given an initial goal model with four solution graphs as shown on the top Figure 5.6 (i.e.* $\mathbf{G} = \{\mathscr{G}_{0.1}, \mathscr{G}_{0.2}, \mathscr{G}_{0.3}, \mathscr{G}_{0.4}\}$*), the desired goals are* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{i}\}$*,* $\{dk_1, dk_2\} \subseteq \mathcal{KB}$ *and proximity is instantiated by "Goal Graph Proximity". Let the assumption that at least one patient is in need of medical help and more than one ambulance is available at some point (*dk_3 = $\exists$ `p,p' ∈ HCProfessionals,` $\exists$ `pa ∈ Patients: RequiresHelp(pa)` $\wedge$ `Available(p)` $\wedge$ `Available(p')` $\wedge$ `p≠p'`*) also be part of the* $\mathcal{KB}$*.*

The above scenario permits situations where stakeholders intend (i.e. it is derivable from a valid solution graph) to dispatch more than one health care professional to a patient in need of medical help, i.e. the assertion α = $\exists$ `p, p' ∈ HCProfessionals,` $\exists$ `pa ∈ Patients, d, d' ∈ Dispatchers: Dispatch(d, p, pa)` $\wedge$ `Dispatch(d', p', pa)` $\wedge$ `p≠p'` *is derivable. Let us assume the stakeholders decide that such a situation is undesirable and hence there should not be any intention to bring it about. In other words,* α *should not be derivable from any valid solution graph.*

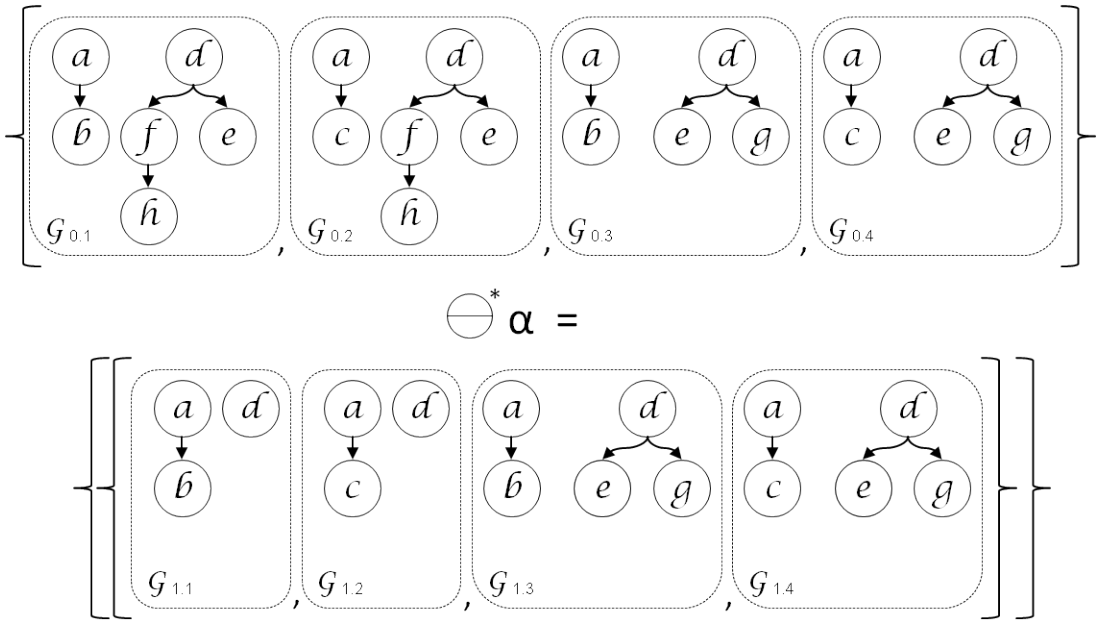

Figure 5.6: Example of the "not entail assertion" operator to avoid the realization of undesired (derivable) state of affairs.

The assertion α *is not entailed by the solution graphs* $\mathscr{G}_{0.3}$ *and* $\mathscr{G}_{0.4}$*, but 1-level entailed by* $\mathscr{G}_{0.1}$ *and* $\mathscr{G}_{0.2}$ *and therefore* $\mathbf{G}$ *does not strongly not entail* α*. The application of the operator on* $\mathbf{G}$ *results in a set* $\{\mathscr{G}_{1.1}, \mathscr{G}_{1.2}, \mathscr{G}_{1.3}, \mathscr{G}_{1.4}\}$ *(as shown on the bottom of Figure 5.6).*

Note, applying the "not entail assertion" operator does not force α to be non-satisfied, but merely ensures that there is no intention to satisfy α. In other words, applying the "not entail assertion" does not (necessarily) avoid situations in which more than one health care professional is dispatched to a patient in need, but ensures that such a situation is not *actively pursued*.

(2) The operator can also be used in situations where a goal becomes infeasible to realize. In such a situation, the "not entail assertion" operator can be used to excluded the goal from all valid solution graphs in $\mathbf{G}$. Note, this is different to an application of the "weak remove desired goal" operator, as it forces a particular goal to be removed, as opposed to simply demoting a desired goal to a dependent goal (in which case the goal may still be part of a valid solution graph if it is justified by a desired goal). The "not entail assertion" operator can hence be viewed as a stronger version of the "weak remove desired goal" operator.

Example. *Given an initial goal model with four solution graphs as shown on the top Figure 5.7 (i.e.* $\mathbf{G} = \{\mathscr{G}_{0.1}, \mathscr{G}_{0.2}, \mathscr{G}_{0.3}, \mathscr{G}_{0.4}\}$*), the desired goals are* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{i}\}$*,* $\{dk_1, dk_2\} \subseteq \mathcal{KB}$ *and proximity is instantiated by "Goal Graph Proximity".*

Let us assume that stakeholders have decided that the goal $\boldsymbol{e}$ *is infeasible to achieve and therefore wish to remove it. Applying the "not entail assertion" operator with parameters* $\mathbf{G}$ *and* $\boldsymbol{e}$ *a set of solution graphs* $\{\mathscr{G}_{1.1}, \mathscr{G}_{1.2}\}$ *(as shown on the bottom of Figure 5.7) is obtained.*

(3) The operator can also be useful in the realization of the "strong add desired goal" operator. To add a goal g as a desired goal to all valid solution graphs in $\mathbf{G}$ (i.e. applying the "strong add desired goal" operator), we first ensure that $\neg g$ is not derivable from any of the valid solution graphs (i.e. applying the "not entail assertion" operator with parameters $\mathbf{G}$ and $\neg g$). This guarantees that a goal g can be added to all valid solution graphs without causing conflict. In other words, the "strong add desired goal" operator can simply be implemented by first applying the "not entail assertion" operator, followed by a "simple" addition operation. This follows a well-known result in the belief revision literature. Levi and Harper showed that revising a knowledge base K with p is given by the contraction (i.e. the removal) of $\neg p$ from K followed by the expansion (i.e. the addition) of p to K[4].

Example. *Given an initial goal model with two solution graphs as shown on the top*

[4] A more detailed and pleasantly accessible summary of these results can be found in P. Gardenfors. [53] page 69

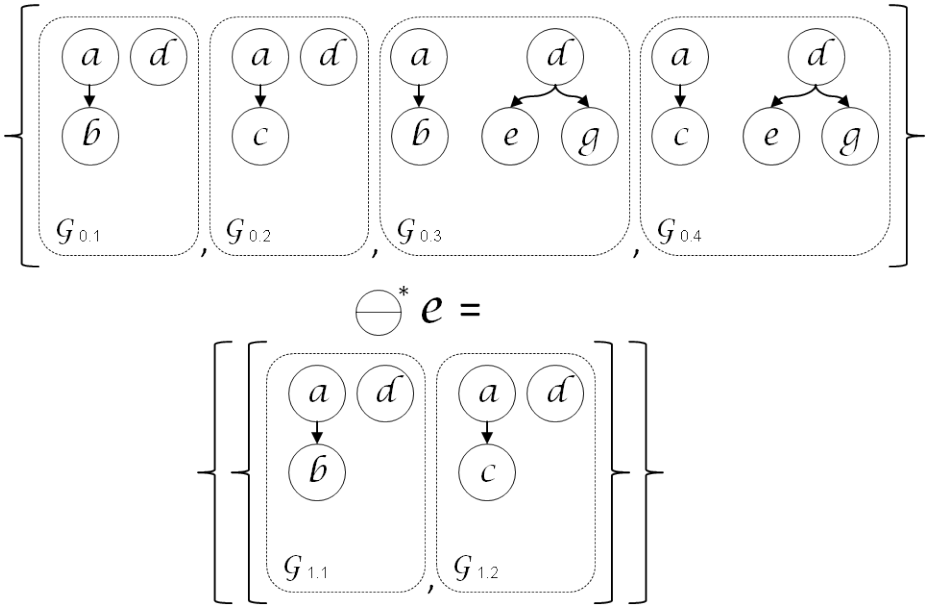

Figure 5.7: Example of the “”not entail assertion” operator to remove a goal from all solution graphs.

Figure 5.8 (i.e. $\mathbf{G} = \{\mathscr{G}_{0.1}, \mathscr{G}_{0.2}\}$*), the desired goals are* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{d}\}$, $\{dk_1, dk_2\} \subseteq \mathcal{KB}$ *and proximity is instantiated by “Goal Graph Proximity”.*

Let us assume that stakeholders have decided to add the goal $\boldsymbol{i}$ *to all solution graphs by applying the “strong add desired goal” operator. This is achieved by first applying the “not entail assertion” operator with* α *set to* $\neg\boldsymbol{i}$*. In Figure 5.8, each set of solution graphs on the right hand side corresponds to a correct outcome of the “not entail assertion operator”.*

Finally, adding goal $\boldsymbol{i}$ *to each solution graph in the respective set achieves the desired outcome of the “strong add desired goal” operator (see Figure 5.9).*

We acknowledge that this result does not hold for all possible instantiations of the “not entail assertion” operator class (more precisely all instantiations of the proximity relation $<_{\mathscr{g}}$). For example, the application of the “not entail assertion” operator may remove a (parent) goal, which in turn requires the removal of its (dependent) sub-goals. The addition of the respective goal to the solution graph does not “re-include” any of the removed sub-goals. However, one or more of the removed sub-goals may be a correct refinement (together with other goals in the graph) of the goal that was added and hence could be retained in the solution graph. Depending on how the proximity relation $<_{\mathscr{g}}$ is instantiated, the combination of the “not entail assertion” operator followed by a simple addition operation may return a set of solution graphs which

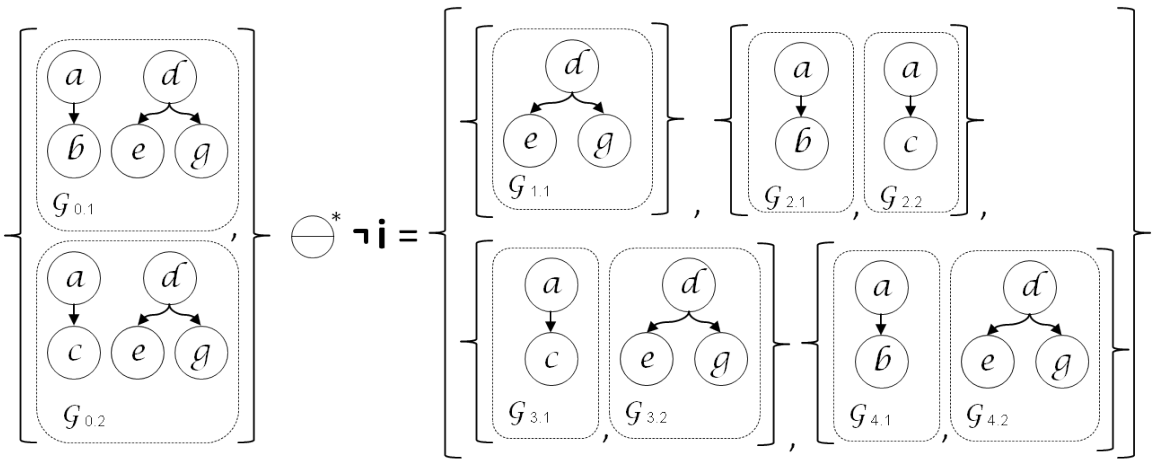

Figure 5.8: Example of the "not entail assertion" operator to implement the "strong add desired goal" operator.

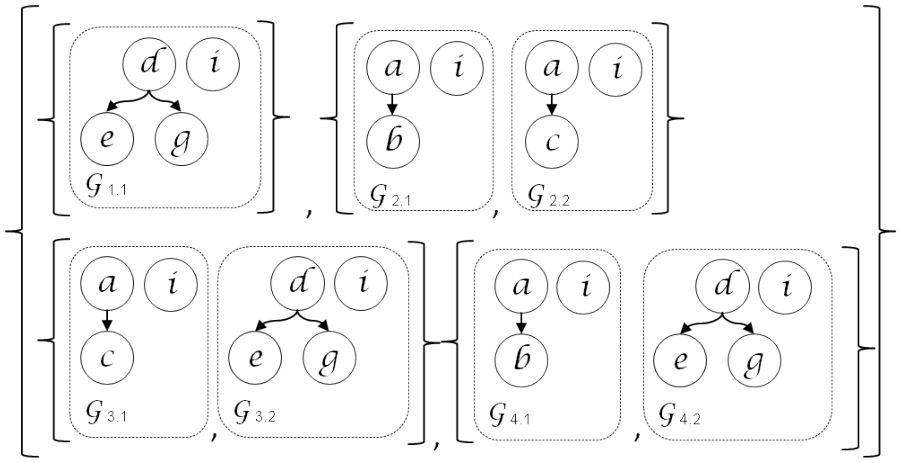

Figure 5.9: The goal model after the addition of goal **i**.

are non-minimally modified. Although the careful construction of such a scenario is possible, it is unlikely to occur in practice. Furthermore, the above example is not a problem if we define proximity by "Goal Graph Proximity".

The "not entail assertion" operator begs the question whether there is a need for a reverse operator - an "entail assertion" operator. We argue that such an operator is not required, since the stakeholder's *desire* to bring about particular state of affairs should be represented via a new desired goal. In other words, this change request should be implemented by the "strong add desired goal" operator.

The mindful reader may have noticed that the effect of the "not entail assertion" operator may be "forgotten" after the application of other operators. We believe that

this is not counter intuitive. However, there do exist situations in which it may be undesirable. For example, in a compliance setting (where compliance is maintained by ensuring that particular assertions remain non-derivable) it may be desirable to ensured that all specified assertions remain non-derivable. We briefly outline how our approach can be extended to avoid these situations, leveraging insights from the literature on iterated belief revision. Ghose [60] shows that an initial contraction of an agent's belief may be "forgotten" by the addition of a (seemingly) unrelated belief. The authors then show that this can be addressed by explicitly representing the contracted beliefs, i.e. the beliefs an agent is committed to not hold - these are referred to as *disbeliefs*. Following the idea of disbeliefs, the goal model could be extended by an additional set to keep track of assertions not to be entailed. In other words, every time the "not entail assertion" operator is applied, the respective assertion (not to be entailed) is added to this set. The definition of a valid solution graph as well as well-formed goal model would have to be adopted correspondingly - the idea being that any valid solution graph in $\mathbf{G}$ must not entail any assertion in the set.

As opposed to goals, for which there exists an incentive to achieve or maintain them, these assertions can be understood as state of affairs for which there is a *disincentive* to maintain or achieve them. We refer to these state of affairs as "anti-goals". It is worth pointing out that the concept of an "anti-goal" is distinct to the concept of an "avoidance goal", described in the KOAS framework[28]. An avoidance goal *requires* that particular state of affairs do not hold. For example, an avoidance goal of a shop owner could be to "avoid long queues". This is different to an anti-goal which merely ensures that there is *no intention* to bring about a particular state of affairs. In the shop example, an anti-goal may require that the shop owner *does not intend* to bring about state of affairs in which customers have to wait in long queues. As opposed to the given avoidance goals this still permits situations in which customers have to wait in long queues.

5.2.5 "Completion" Operator

We say that a solution graph is *complete* or *completed* if no more elements from the library can be added without violating its validity or making a particular assertion (i.e. α) derivable. The "completion" operator (denoted by $(\mathbf{G}, \alpha)\diamond$), takes as input a set of valid solution graphs $\mathbf{G}$ and an assertion α and completes each solution graph.

In the following, we say that the set $\mathbf{G}$ with a single and empty solution graph (i.e. a graph with an empty set of vertices and edges) is *pseudo well-formed*. We propose

the following properties.

(1) *Validity:* For a set of valid* solution graphs, or pseudo well-formed set $\mathbf{G}$ and $(\mathbf{G}, \alpha)\diamond$, it must be the case that each $\mathscr{G}' \in (\mathbf{G}, \alpha)\diamond$ is a valid* solution graph.

(2) *α-exclusion-persistence:* For a set of valid* solution graphs, or pseudo well-formed set $\mathbf{G}$, if $\mathbf{G} \not\models \alpha$, it must be the case that $(\mathbf{G}, \alpha)\diamond \not\models \alpha$.

(3) *Maximality:* For a set of valid* solution graphs, or pseudo well-formed set $\mathbf{G}$ and $(\mathbf{G}, \alpha)\diamond$, it must be the case that for each $\mathscr{G}' \in (\mathbf{G}, \alpha)\diamond$ there does not exist a valid solution graph $\mathscr{G}''$ such that $\mathscr{G}'' \not\models_k \alpha$ for any k and $f_V(\mathscr{G}') \subset f_V(\mathscr{G}'')$ or $f_E(\mathscr{G}') \subset f_E(\mathscr{G}'')$.

(4) *Inclusion:* For a set of valid* solution graphs, or pseudo well-formed set $\mathbf{G}$, for each $\mathscr{G} \in \mathbf{G}$, there exists a $\mathscr{G}' \in (\mathbf{G}, \alpha)\diamond$ such that $f_V(\mathscr{G}) \subseteq f_V(\mathscr{G}')$ and $f_E(\mathscr{G}) \subseteq f_E(\mathscr{G}')$[5].

(5) *Vacuity:* For a set of valid* solution graphs, or pseudo well-formed set $\mathbf{G}$, if $\mathbf{G} \not\models \alpha$ and for any $\mathscr{G} \in \mathbf{G}$ there does not exist a valid solution graph $\mathscr{G}'$ such that $f_V(\mathscr{G}) \subset f_V(\mathscr{G}')$ or $f_E(\mathscr{G}) \subset f_E(\mathscr{G}')$ and $\mathscr{G}'' \not\models_k \alpha$ for any k, then $\mathbf{G}\diamond = \mathbf{G}$.

We consider the particular case of a pseudo well-formed set because it allows us to construct (all or some) valid solution graphs from the library without the need to have identified a valid solution graph beforehand.

The α-exclusion-persistence property states that if α is not entailed by any solution graph, then α remains non-entailed after an application of the completion operator. Note, we are not forced to consider α and can simply "disable" it by setting it to $\bot$.

The maximality property states that each solution graph in the set $(\mathbf{G}, \alpha)\diamond$ must be a completed, i.e. no more elements from the library can be added without violating the validity of the solution graph or making the assertion α derivable.

The inclusion property states that the completed set $(\mathbf{G}, alpha)\diamond$ must be based on the original set of graphs, i.e. any solution graph of the original set $\mathbf{G}$ must be a sub-graph of at least one valid solution graph in the completed set. In other words, the completion of a set $\mathbf{G}$ never removes any element from $\mathbf{G}$. The operator is successful if the α-exclusion-persistence, maximality and inclusion property are satisfied.

[5] Recall that $f_V(\mathscr{G})$ and $f_E(\mathscr{G})$ return the set of vertices and respectively the set of edges from the graph $\mathscr{G}$.

Note, an empty solution graph is a subset of any valid solution graph and hence any completion of the pseudo-well-formed set satisfies the success property. In the case that the set $\mathbf{G}$ satisfies the maximality and inclusion property then the operation is vacuous.

Given these properties the "completion" operator is defined as follows.

Definition 27. *(completion operator)*
Given a set of valid solution graphs, or pseudo well-formed set* $\mathbf{G}$*, let* $(\mathbf{G}, \alpha)\diamond^*$ *return a set, such that each set* $\mathbf{G}' \in (\mathbf{G}, \alpha)\diamond^*$ *satisfies the properties (1)-(5). We define* $(\mathbf{G}, \alpha)\diamond = s((\mathbf{G}, \alpha)\diamond^*)$*, where* s *is a selection function that returns an element of* $(\mathbf{G}, \alpha)\diamond^*$*.*

The completion operator is a powerful operator and permits a wide range of usage scenarios.

(1) Given an empty set of solution graphs (i.e. a pseudo well-formed set), the operator allows systematic exploration of the space of solution graphs which are derivable from the goal library. In this context, the selection function can be encoded in various interesting ways. For example, it could be encoded such that $(\mathbf{G}, \alpha)\diamond$ contains the most preferred (or k-most preferred) solution graphs; and/or all solution graphs in $\mathbf{G}$ are realizable by an available description of functionality.

(2) In other scenarios, an application of the "completion" operator follows an application of one of the other maintenance operators, which may have returned a set $\mathbf{G}$ with *incomplete* solution graphs.

Example. *Consider a goal model with two solution graphs as shown on the left hand side of Figure 5.10 (i.e.* $\mathbf{G} = \{\mathscr{G}_{0.1}, \mathscr{G}_{0.2}\}$*), with the desired goals* $LIB^{des} = \{\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{i}\}$*,* $\mathcal{KB} = \{dk_1, dk_2, dk_3\}$ *and proximity instantiated by "Goal Graph Proximity".*

Let $\alpha =$ $\exists$ `p, p' ∈ HCProfessionals,` $\exists$ `pa ∈ Patients, d ∈ Dispatchers: Dispatch(d, p, pa) ∧ Dispatch(d, p', pa) ∧ p≠p' ∧ pa = pa`*,*
then $\{\{\mathscr{G}_{1.1}, \mathscr{G}_{1.2}\}, \{\mathscr{G}_{2.1}, \mathscr{G}_{2.1}\}, \{\mathscr{G}_{3.1}, \mathscr{G}_{3.2}, \mathscr{G}_{3.3}\}\}$ *are the outcome of* $(\mathbf{G}, \alpha)\diamond^*$*, where each set denotes a correct outcome of* $(\mathbf{G}, \alpha)\diamond$*.*

It is worth mentioning that an implementation of the "completion" operator is particularly amenable for an mixed initiative approach. In a mixed initiative approach, the tool (i.e. the software agent) and the requirements engineer (i.e. the human agent) work seemingly together to solve the problem of finding a valid outcome of the completion operator. For example, given an incomplete valid solution graph, the

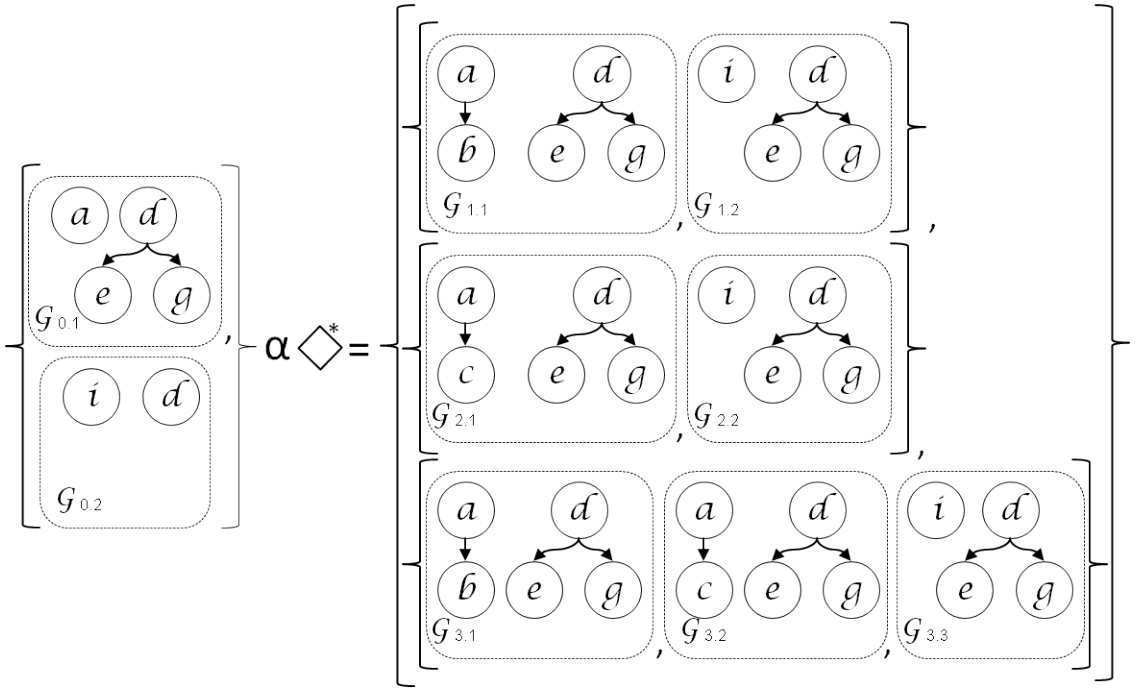

Figure 5.10: Example of the "completion" operator.

human agent could select one of the leaf goals for further refinement and the software agent could compute (using abductive reasoning techniques) correct and immediate refinements from goals in the goal library. The human agent would then select the most preferred refinement and the next goal to be refined. Such a mixed initiative approach has advantages. On the one hand, we benefit from the software agents capabilities to search a very large space of alternative refinements and a guarantee that each identified refinement is correct. On the other hand, the human selection of refinements according to his/her preference allows the search space to be pruned, i.e. the software agent does not have to compute refinements for goals of refinements that the human agent has ruled out.

5.3 Related Work

This section juxtaposes our approach to maintain goal models with existing ones. Like the work by Ernst and his co-authors [42, 44, 45], Ghose [64, 63], Zwhogi et al. [170], MacNish and Williams [113] our operators are inspired by belief revision techniques and principles. The emphasize of our comparison will hence be with respect to their work. There has also been work on revising models of intentionality in the area of

intelligent agents, which shares some principles with our work. We will provide a brief overview of this line of work at the end of the section.

As in the related work section of Chapter 4, will use the list of features suggested by Ghose [64] to guide the comparison.

Minimal Change: Minimal change is an important characteristic of most maintenance frameworks (exceptions are the work by Nuseibeh and his co-authors [126, 82] and Garces and his co-authors [52, 31]). This is particularly true for the belief revision inspired operators proposed in Ernst and his co-authors [42, 44, 45], Ghose [64, 63], Zwhogi et al. [170], MacNish and Williams [113], which all support a modular definition of minimal change. The proposed maintenance operators of our framework are no exception to this. Nevertheless, gaps and differences exit.

For example, we motivated, in detail, a feasible instantiation of minimal change which takes into account a solution graph's structure and the different ontological status of desired and dependent goals, while existing work (e.g. the work by Zwhogi et al. [170]) does not discuss how minimal change might be instantiated. On the other hand, in Ernst et al. [42] various intuitions of minimal change are discussed and concrete examples are given. However, in the work by Ernst et al. [42] minimal change is defined with respect to the specification (i.e. the set of tasks), not taking into account minimal change at the "goal level". In our framework, in principle, the definition of minimal change is not restricted to a particular type of element in the goal model.[6] We also believe that a sensible definition of minimal change should first take into account similarity between what stakeholders want (i.e. stakeholder goals) over how similar the realization (i.e. the set of tasks) is.

Additionally, as opposed to the work by Ghose [64, 63], Zwhogi et al. [170] and MacNish and Williams [113], we ensure that the minimal change property does not "protect" elements which have become superfluous, due to a change request. Given a goal and its refinement/decomposition into sub-goals and the removal of the parent goal, the maxi-adjustment transmutation used by MacNish and Williams [113] only ensures that one of the sub-goals is removed, resulting in (potentially) superfluous goals to remain in the goal model. In addition, Ghose [64, 63] does not consider the removal of requirements that have become "unjustified" due to the removal of their justification. More precisely, given a requirement $\alpha : \beta$, the "removal" of the requirement's justification (i.e. β), does not result in the consequent removal of α,

[6] Up to this point solution graphs only contain goals. In Chapter 6 we discuss solution graphs with service realization. However, this is not a problem, since minimal change is defined via a proximity relation between solution graphs and concerned about the elements in the solution graph.

since α remains part of at least one r-consistent set. This is not the case in our work, because our operators must return a valid solution graph and the definition of solution graph validity requires all dependent goals to be part of a refinement and all refinements to be correct. Hence, all goals in a solution graph are either desired or support desired goal, i.e. no superfluous goals can be part if a valid solution graph.

In Zowghi et al. [170], the contraction of a requirement is explicitly recorded in the set of constraints. This avoids situations where the contraction of a requirement does not persist after further maintenance steps. In the proposed maintenance framework, we do not record the contraction of a goal/requirement. This can result in situations where a goal that has been removed (e.g. due to the application of the "not entail assertion" operator) is re-included to a solution graph in later maintenance steps. This may be undesired in certain situations, and in Chapter 5.2.4 we briefly discussed how our framework could be extended in this direction. On the other hand, in situations where a goal has been contracted because stakeholders do not actively desire its realization, it may be too strict to ensure that the goal remains contracted, since the stakeholders may be willing to realize the goal if it supports the fulfilment of desired goals. In our work, the "weak remove desired goal" operator ensures that a given desired goal is demoted to a dependent goal, such that it has to be removed from the solution graphs unless it supports the satisfaction of desired goals.

Trade-off analysis: The work by Ghose [64, 63] and Ernst and his co-authors [42, 44, 45] permit the rejection of a change if the result would be less preferred. We argue that in some situations we are willing to accept a goal model that is less preferred then its predecessor (in terms of importance of requirements to the stakeholders). As pointed out before, this is the case when a change request is not driven by a shift in stakeholder preference, but driven by external factors like new regulations, which may force the commitment to a less preferred but compliant model.

Delayed Commitment: Ghose [64, 63] and Zowghi et al. [170] support a delayed commitment by not forcing to select one of the maximal consistent subsets that can be identified after a change to the requirements. Ernst and his co-authors [42, 44, 45] support a delayed commitment by delaying the application of the query operators. In our framework, the commitment to a particular set of solution graphs can be delayed by encoding the selection function of each maintenance operator in such a way that it returns the union of all sets of solution graphs (i.e. the merger of all valid operator outcomes).

Additionally, the work by Ghose [64, 63] and Zowghi et al. [170] supports a "lazy

evaluation strategy" [64]. Recall that in a "lazy evaluation strategy" the computation of maximal consistent sets is deferred as long as possible, i.e. up to the point where a decision needs to be made. This allows the users to make changes to the goal model, while omitting the computationally expensive exercise of generating maximal consistent sets. This feature is particularly useful at an early stage of the requirements engineering process, where the goal model changes frequently. Our proposed framework also supports a "lazy evaluation strategy", simply by working with an empty set of solution graphs. This allows changes to be made to the goal library as well as domain knowledge base, without the need to accurately reflect the changes in the solution graphs (i.e. without the need to maintain solution graph validity). At the point, where a decision has to be made, the "completion" operator can then be used to compute valid solution graphs from the goal library.

A disadvantage of a "lazy evaluation strategy" is that after changes have been made to the goal model, typically *all* solutions (i.e. maximal consistent sets) have to be recomputed. This is undesirable during a later stage in the requirements engineering process. At this stage, change tends to be less frequent and therefore solutions are expected to be available after each change request. Our approach (as opposed to the approaches presented in Ghose [64, 63] and Zowghi et al. [170]) avoids the need to recompute *all* solutions by only incorporating the change request in the solutions of interest (i.e. the elements in the set of solution graphs) via one of the proposed change operators. This often has computational advantages, especially when the set of solution graphs only consists of a few elements. However, as the number of elements in the set of solution graph increases the advantages are increasingly offset.

The idea of the "not entail assertion" operator is to modify the goal model in such a manner that a particular assertion is non-derivable. Operations with the same effect have been discussed in Ghose [64] and Nuseibeh and Russo [126]. In Ghose [64], it is ensured that a particular assertion α is not entailed by any r-maximal set by adding a new essential justified requirement $\varnothing : \neg\alpha$, while in Zowghi et al. [170] $\neg\alpha$ is added to the set of facts F. In both cases, the effect is that every maximal r-consistent set/scenario is consistent with $\neg\alpha$ and hence α not derivable. It is worth pointing out that in both approaches the effect of the change is persistent, until the constraints $\neg\alpha$ is removed. In Nuseibeh and Russo [126], abductive reasoning techniques are used to identify parts of the requirements specification from which an assertion α is derivable. Removing these parts from the specification ensures that α is non-derivable. This is a non-persistent change in the sense that the addition of new requirements to the

specification may (undesirably) make α derivable. An application of the "not entail assertion" is also non-persistent, but Chapter 5.2.4 we briefly discussed how this can be overcome by keeping track of all assertions not to be entailed.

There has also been work on revising models of intentionality in the area of intelligent agents that share some principles with our work. In Zhou et al. [169], the authors revisit and extend a family of partial implication relations and show how these can be used to define goal change operators. Roughly, a propositional formula P "partially implies" a formula Q, if for every case (i.e. possible truth assignment) of P there is a case of Q such that a *part of* the case of P is a part of the case of Q. The revision of a goal Q to a goal Q', such that P satisfies Q', is analogous to P partially satisfying Q. This can also be understood as a weakening of Q to Q' as described in [99]. In the work by Tettamanzi, Pereira and colleagues [27, 26, 151] an abstract goal revision operator is proposed to capture the arising of new desires. Like in our work, the operator is motivated by a set of normative properties and requires change to be optimal. However, as opposed to our work, an agent's goals are not structured and goals are required to be expressed in a limited propositional language (only conjunction and negation are considered as connectives). Bell and Huang [15] propose a hierarchical goal formalism and show how it can be revised when an agent's belief or motivation changes. Similar to that proposed by MacNish and Williams [113], an agent's goal hierarchy is given by a preference relation over a set of goals, where all goals in the goal hierarchy are required to be consistent and most preferred according to the preference relation. As mentioned by the authors, abstract high-level goals tend to be of highest preference and hence are more likely to persist after a change. Our definition of "Goal Graph Proximity", given in this chapter, follows this intuition. However, like the work by MacNish and Williams [113], Bell and Huangs [15] formalism does not consider dependencies between goals on a lower level and goals on a higher-level. In addition, Bell and Huang [15] do not permit reasoning with more than one consistent set of goals, as opposed to our formalization that permits reasoning with multiple consistent sets of goals (i.e. solution graphs).

5.4 Summary

In this chapter, we introduced a family of goal model maintenance operators which were demonstrated to address a multitude of change requests.

In the first part of this chapter (Chapter 5.1) we showed how minimal change can be defined in a modular manner by deploying an abstract proximity relation. The proximity relation allows us to assess the relative distance between two solution graphs with respect to a third graph (i.e. the original solution graph). We further motivated (what we believe to be) a sensible instantiation of the proximity relation. This instantiation takes into account stakeholder desire as well as the refinement structure of a solution graph.

In the second part of this chapter (Chapter 5.2) we introduced a set of goal model maintenance operators. The operators were defined as abstract operator classes, which permits their instantiation for different intuitions of minimal change and different languages in which goals may be specified. The following maintenance operators were introduced:

- The "*strong add desired goal*" operator minimally modifies the set of solution graphs such that the goal to be added is a desired goal of *all* solution graphs.
- The "*weak add desired goal*" operator is used to add a goal as a desired goal to those solution graphs for which it can be consistently included.
- The "*weak remove desired goal*" demotes a desired goal to a dependent goal and adapts the solution graphs respectively.
- The "*not entail assertion*" operator minimally modifies the set of solution graphs such that a particular assertion is strongly non-derivable.
- The "*completion*" operator takes as input a (potentially empty) set of solution graphs and completes each of these solution graphs by adding as many goals from the library as can be done without violating its validity.

Chapter 6

Addressing the Requirements Problem

The ultimate aim of requirements engineering is to find and maintain a solution to the *Requirements Problem*. It is therefore worth exploring how our approach can be extended to address the requirements problem in the context of inconsistent and evolving goals.

In Chapter 2.2.2, the requirements problem was summarized as the search for the most preferred specification, i.e. description of system behaviour, that satisfies (as close as is feasible) all compulsory- and as many optional goals as possible.

The requirements problem can be split into two parts. The first part is concerned with finding a description of system behaviour (e.g. captured by a set of service descriptions) that satisfy all mandatory goals and as many optional goals as is possible. This addresses the *functional aspects of the requirements problem* and is formalized as follows.

Given domain knowledge and assumptions $\boldsymbol{KB}$*, compulsory goals* $\boldsymbol{G}_C$*, and optional goals* $\boldsymbol{G}_O$*, a solution to the functional aspect of the requirements problem is a specification* $\boldsymbol{S}$*, such that* $\boldsymbol{KB}, \boldsymbol{S} \vDash \boldsymbol{G}_C, \boldsymbol{G}'_O$*, where* $\boldsymbol{G}'_O$ *is a maximal subset of* $\boldsymbol{G}_O$ *such that* $\boldsymbol{K}, \boldsymbol{S}, \boldsymbol{G}_C, \boldsymbol{G}'_O$ *is consistent.*

The second part is concerned with finding the most preferred description of system behaviour that satisfices all mandatory- and as many optional soft-goals as possible. This addresses the *non-functional aspects of the requirements problem.*

Given domain knowledge and assumptions $\boldsymbol{KB}$*, compulsory goals* $\boldsymbol{G}_C$*, optional goals* $\boldsymbol{G}_O$*, compulsory soft-goals* $\boldsymbol{Q}_C$*, optional soft-goals* $\boldsymbol{Q}_O$ *and let* $\boldsymbol{G}_O^*$ *be the most preferred maximal consistent subset of* $\boldsymbol{G}_O$ *and* $\boldsymbol{G}_O^*$ *be the most preferred maximal consistent subset of* $\boldsymbol{Q}_O^*$*, the specification* $\boldsymbol{S}$ *is a solution to the non-functional part of the requirements problem, if* $\boldsymbol{KB}, \boldsymbol{S} \vDash \boldsymbol{G}_C, \boldsymbol{G}_O^*$ *and* $\boldsymbol{S}$ *satisfices* $\boldsymbol{Q}_C, \boldsymbol{Q}_O'^*$*.*

A solution that satisfies both the functional- and non-functional parts of the requirements problem is a solution to the requirements problem.

This chapter has two main parts. In the first part (Chapter 6.1), we summarize our earlier work on maintaining goal models in the context of a service landscape [80] (in Chapter 6.1.1) and discuss how the results can be leveraged to addresses the *functional aspects* of the requirements problem in the context of inconsistent and evolving goals (Chapter 6.1.1.5). In essence, a solution to the functional aspect of the requirements problem is given by a descriptions of system functionality, e.g. by a set of service description, that satisfy all goals in a solution graph. We formally establish under which conditions a service description satisfies a goal in a solution graph (in Chapter 6.1.1.2).[1] We then (amongst others) outline a machinery that automatically searches through a pool of service descriptions to find the ones that satisfy a given goal (in Chapter 6.1.1.5). Such a machinery is important in the context inconsistent and evolving goals, as changes to the goal model may require the exploration of a revised set of service descriptions that satisfy the goals of the solution graph. In other words, this allows us to maintain an answer to the functional part of the requirements problem.

In the second part, we summarize our earlier work on Green Business Process Management [79, 78, 77] and discuss its benefits in exploring answers to the non-functional aspects of the requirements problem. In essence, an answer is a description of system behaviour, e.g. given by a set of business processes, for which it can be established that they are the most preferred ones (with respect to a given set of non-functional goals). As a first step, we show (in Chapter 6.2.1 and Chapter 6.2.2) how the non-functional performance of a design artifact (like a business process or service) can be estimated during design time by correlating it with resource models. This is followed (in Chapter 6.2.3) by a description of a process improvement machinery, which suggests process re-designs that improve the non-functional performance of the

[1] A service description is said to satisfy a goal in a solution graph if its formally described post-condition, together with the domain knowledge base of the goal model do entail the goal.

original process, but remain to satisfy the goal it is correlated to. Such machinery is very useful in situations where the performance objectives of stakeholders change and where we consequently seek to find new or revised descriptions of functionality that better address the performance objectives (see Chapter 6.2.4). Chapter 6.3 discusses related work, while Chapter 6.4 concludes this chapter.

6.1 Functional Aspects

In Chapter 6.1.1, we recapitulate our work on "Maintaining Motivation Models (in BMM[2]) in the Context of a (WSDL-S) Service Landscape" [80]. This work is based on a slightly different formalization of a goal model as well as "not entail assertion" operator, as discussed in the previous chapters. We stay true to the original paper [80], but highlight and discuss differences to the goal model formalization elaborated in this thesis. In Chapter 6.1.2 we show how this is the basis for addressing the *functional aspects of the requirements problem.*

6.1.1 Maintaining Motivation Models and in the Context of a Service Landscape

A motivation model (e.g. represented as a Business Motivation Model [70]) is an important artifact in an organizational context, as it encodes organizational intent and guides the maintenance of its service capabilities. Ultimately all service capabilities of an organisation should be traceable back to (and justified by) elements in the motivation model.

The ever-changing business context requires organisations to constantly adapt their motivation and service representations. For example, an organisation may change its vision in terms of modifying its goals or may have to give up (or adopt) services to remain compliant with changing regulations (which has recently been the focus of research). However, this poses challenges. Giving up a service capability may result in unrealized organisation motivation, and changes at the motivation level may render existing service capabilities superfluous or require the adoption of new service capabilities.

Existing work, which takes the motivation and service context into account, focuses on verification and service composition (e.g. [86], [21], [90], [112]), or techniques

[2]Business Motivation Model

for handling change at the service level (e.g. [2], [74]). Little has been done to handle (1) change at the motivation level (some exceptions are [64], [170], [42]) and its consequences to the service level and (2) compliance obligations at the motivation level (which has attracted recent research in (goal-oriented) requirements engineering e.g. [83], but only modelling aspects have been addressed so far). This is important in the service context, since non-compliant motivation models give rise to non-compliance at the service level. Both change- and compliance management have recently been identified as key challenges of service oriented computing [128].

We propose a framework for handling change at the motivation level and highlighting consequent impacts to the service level. In particular we formalize a change operator that is recommended for (but is not restricted to) changes driven by compliance, i.e. where a motivation model needs to be adapted to meet compliance obligations. The *manual* modification of a (motivation) model can be a time-consuming and error-prone exercise. For example, we might overlook inconsistencies or commit to sub-optimal modifications (often simply because of human cognitive limits on how much of the space of alternative modifications we can explore). We might interpret *optimality* in many different ways. One approach that we adopt, motivated by the need to protect existing investments in a project, is to minimize deviation from the prior motivation model.

We proceed as follows. Chapter 6.1.1.1 covers preliminaries and introduces a running example. Chapter 6.1.1.2 formalizes a motivation-service (F-hyper-) graph and introduces a hierarchical entailment relation with the graph. Chapter 6.1.1.3 formalizes a compliance sensitive operator via a set of AGM theory [1] inspired postulates. Chapter 6.1.1.4 discusses related work and Chapter 6.1.1.5 concludes the chapter.

6.1.1.1 Preliminaries

Business Motivation Model: A Business Motivation Model (BMM)[3], as standardized by OMG, is a hierarchical representation of organisational *motivation* with two key concepts: *means* and *ends*.[4]. Ends are used to describe what an organisation *wants to be*, while means describe how the organisation plans to achieve its ends (thus means *realize* the ends). In a BMM, both means and ends are hierarchically structured. In the end-hierarchy, the *vision* is the most abstract element and describes a state the

[3] http://www.omg.org/spec/BMM/1.1/

[4] Influencers and assessments are also concepts of a BMM model, but do not form part of our analysis

organisation would like to achieve. A vision is attained by a set of goals. A *goal* is a statement about a state or motivation the organisation seeks to maintain or bring about. Goals can be further refined to obtain *objectives*, which set measurable targets necessary for goal achievement. Objectives do not form part of our design-time analysis since their evaluation requires run-time information. In the means-hierarchy, a *strategy* represents an accepted upon course of action to achieve the ends. Strategies are implemented via *tactics*, which are narrower in scope. Figure 6.1 (a) shows some examples of a vision, strategy, goal and tactic and their hierarchical representation as a BMM in Figure 6.1 (b).

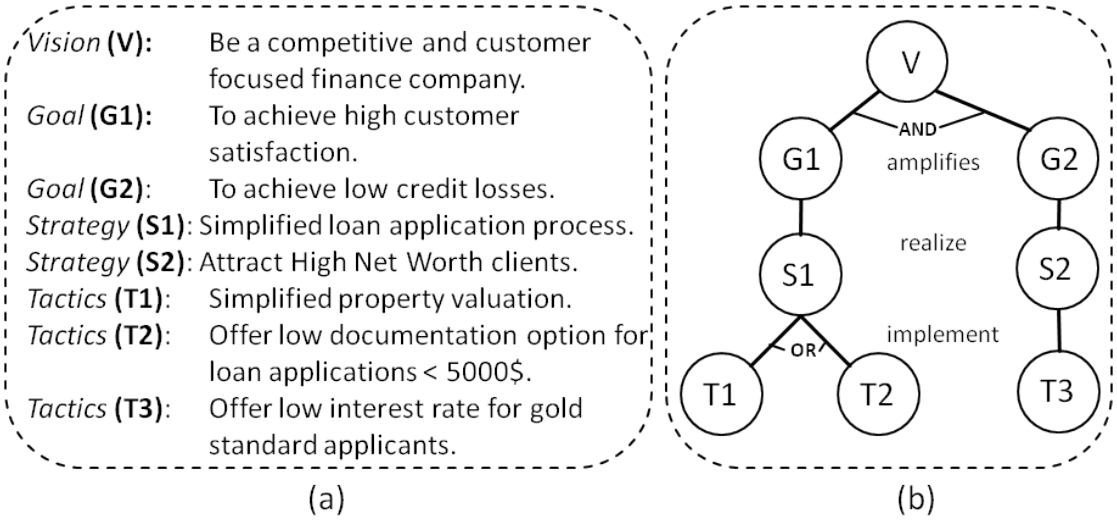

Figure 6.1: Example of a Business Motivation Model.

In the BMM of Figure 6.1 (b), the "AND" connection between goals "G1" and "G2" relating them to parent vision "V" indicates that *both* goals need to be realized to bring about the vision; the "OR" connection between tactics"T1" and "T2" and strategy"S1" denotes that either of the tactics can be pursued to implement the strategy.

Service representation: Semantic service description standards like WSDL-S and OWL-S enforce a degree of rigour and detail about how services are modelled, which in turn supports more sophisticated analysis. Most of these standards support the specification of pre- and post-conditions, which we leverage in our framework.

Our current work leverages WSDL-S, but, as noted above, other semantic service standards could equally well be used. For simplicity of exposition, we use the syntax of classical logic in representing pre- and post-conditions in the following. Figure 6.2

shows a WSDL-S fragment (of the pre- and post-conditions) of a "Premium Loan Service".

```
<operation name="PremiumLoan">
    <input .../>
    <output .../>
    <precondition  name="ExistingNamePrecond" expression="GoldStandardApplicant) ..."/>
    <effect name="PremiumLoan"  expression ="IncomeStatementReceived &
                    LoanApplicationHandled & GoldStandardDiscount"/>
</operation
```

Figure 6.2: WSDL-S fragment of a "Credit Check Service".

Service descriptions are maintained in a *service catalogue*. In addition to the "Premium Loan Service" service (denoted by Se3), our running example will refer to the following services in the service catalogue of a hypothetical organisation (for brevity we only provide their name and effect): (Se1) Simplified Property Valuation: `PropertyValued` $\wedge$ `SimplifiedPropertyValuation` (Se2) Small Loan Service: `LoanApplicationHandled` $\wedge$ `LowDocumentation.`

Note that our framework only requires services to be annotated with formal post-conditions. This has advantages in the case of business service descriptions, which often do not readily come with formal pre- and post-conditions, as it reduces the burden on the analyst to describing post-conditions.[5]

6.1.1.2 Formal Representation

We now propose a formal representation of BMM models and service catalogues that permits the application of (semi-) automated machinery for managing change of various kinds. The formal representation is based on a *motivation library* and *Motivation-Service graph*, which are described in detail as follows.

Motivation Library: A motivation library is a domain specific collection of feasible motivations, i.e. motivational elements that are feasible to realize (note that some states of affairs might be highly desirable for an organization, but not feasible to achieve) and which may (but do not have to) be part of an organisation's BMM. The library contains *at least* the elements that are part of the BMM.

[5] While our analysis does not require pre-conditions, pre-conditions have been successfully applied in many interesting scenarios ranging from correctness checking to adaptive service composition. For example, Weber et al.[158] use pre-conditions for checking the correctness of semantically annotated process designs, while Mukhija and Glinz[119] use them in the context of adaptive software architectures.

We require each element in the library to be represented by formal assertions of the language $\mathcal{L}$. In other words, the library is simply a set LIB of assertions, such that $LIB \subseteq \mathcal{L}$. The assertions provide a (machine understandable) description of the respective motivation element. Since elements of the BMM are part of the library, we require a formal assertion to be associated with them. We do this via annotations to the BMM elements. Annotation for elements representing "ends" (e.g. goals) describe the conditions that must hold for the "end" considered to be brought about. Annotations for the "means" (e.g. strategy, or tactic) describe the conditions that are expected to be brought about if the strategy or tactic were to be executed. Although "ends" and "means" (including services) are ontologically distinct, their formal representations are the same, i.e. they are expressed in the same formal language. For example, the list below provides the assertions for the respective BMM elements of Figure 6.1, as well as the strategy S3 and the tactic T4 (which do not appear in the current BMM but might have appeared in a previous one), as part of the library. Note that the assertions may be arbitrarily detailed, but are kept simple for ease of exposition.

V:	`Competitive` $\wedge$ `CustomerFocused`	G1:	`HighCustomerSatisfaction`
G2:	`LowCreditLoss`	S1:	`SimplifiedLoanProcessing`
S2:	`HighNumberOfHighNetWorthClients`	T1:	`SimplifiedPropertyValuation`
T2:	`LowDocumentation`	T3:	`GoldStandardDiscount`
S3:	`StrictCreditAssessment`	T4:	`IncomeStatementRecevied`

Within the motivation library, we refer to the motivational elements that an organisation desires to (i.e. would like to) adopt to its BMM as $LIB^{des} \subseteq LIB$. The motivational elements the organisation is actually *committed to* are given by the elements (denoted by their formal assertion) of the BMM model (denoted by $I \subseteq LIB$). Naturally, an organisation should be committed to all desired motivations. However, this may not always be possible. An organisation may not be committed to bringing about a desired motivation m, because m may be *inconsistent* with other (desired) motivations that have been accorded higher preference, or m may be *infeasible* to pursue with other desired motivations (e.g. doing so would cause a compliance violation).

Motivation-Service Graph: We begin by defining *F-hypergraphs*, which are generalizations of simple (directed) graphs that allow edges with a single source vertex but (potentially) more than one target vertices. An F-hypergraph is acyclic if and only if a path in the graph for which the "start"- and "end-" vertex are the same does

not exist. Formally, an acyclic F-hypergraph is a pair $\langle V, E\rangle$, where $V = \{v_1, v_2, ..., v_n\}$ is a set of vertices and $E = \{e_1, e_2, ..., e_n\}$ is a set of (directed) F-hyperedges, where each edge $e = (x, Y) \in E$ and $x \cup Y \subseteq V$. For any edge $e \in E$, let $S(e)$ denote the source vertex, and $T(e)$ the set of target vertices of e. $\langle V, E\rangle$ is acyclic if and only if there does not exist a path $(x_1, e_1, x_2, e_2, ..., e_q, x_{q+1})$, where $x_1 \in S(e_1)$, $x_{q+1} \in T(e_q)$, $x_i = S(e_i)$, for every i, and $x_{q+1} \in S(e_1)$ (this definition is adopted from [50]).

A *Motivation-Service-graph* (MS-graph) is an *acyclic and labelled F-hypergraph* representation of the BMM elements, services (from a service catalogue) and their hierarchical relationships. In an *MS-graph*, each vertex is associated with an element of either the motivation library or the service catalogue. From here on, we will also assume the ability to refer to the type of each element of the motivation library (via the type of associated vertex). Vertices associated with services will refer to the service post-conditions.

We use edges $e = (x, Y)$ to denote an AND-relation between a vertex x and a set of sub-vertices Y (note that edges represent *refinement* relationships between BMM elements or *realization* relationships between BMM tactics and services). For example, the edge "e1" of the MS-graph in Figure 6.3 (left hand side) is an AND-relation. In an OR-relation, we use $e = (x, Y)$ and $e' = (x, Y')$ to denote that the sets of vertices $Y \subset V$ and $Y' \subset V$ are distinct refinements or realizations of x. For example, the edge "e4" and "e5" of the MS-graph in Figure 6.3 (left hand side) are two distinct relations. Note, if we had defined the relationship between a goal and each of its refinements (realizations) individually (as we would be obliged to do in a simple graph), we would not be able to distinguish between goals belonging to alternative refinements (realizations). We believe that our formalization addresses many of the deficiencies in the way AND/OR (goal) graphs are formalized (most ignore the fact that such graphs are in fact hypergraphs and that AND edges are in fact F-hyperedges).

In addition, a *background knowledge base* $\mathcal{KB} \subseteq \mathcal{L}$ is used as an encoding of domain and organisation specific knowledge (which, for example, could be represented in RuleML). The $\mathcal{KB}$ may contain the knowledge that a low loan processing time results in high customer satisfaction (e.g LowLoanProcessingTime → HighCustomerSatisfaction), or that accepting credit applications with low documentation (i.e. without an income statement, etc.) is considered not to be a strict credit assessment (e.g. LowDocumentation → ¬ StrictCreditAssessment). The following rules are also considered in our example HighCustomerSatisfaction → CustomerFocused, StrictCreditAssessment → LowCreditLoss, LowCreditLoss → Competitive, LowDocumentation → Simplified-

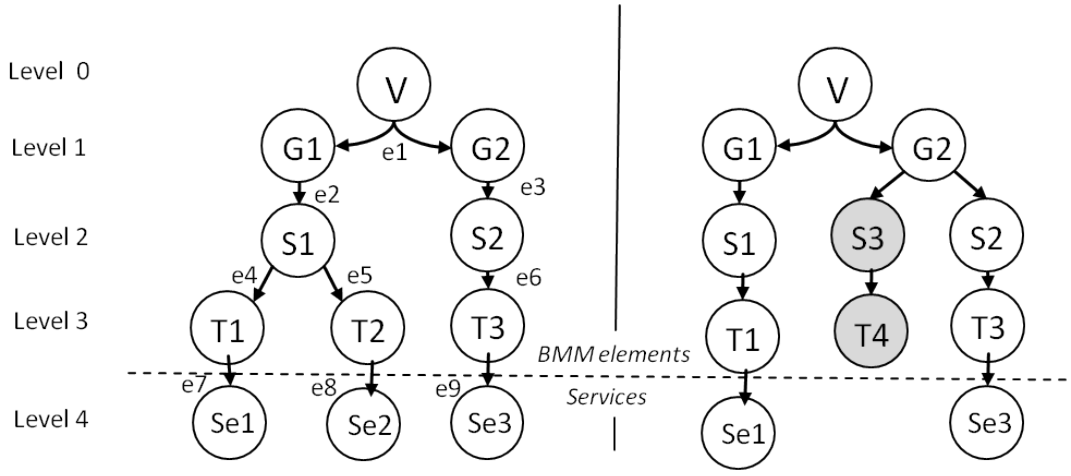

Figure 6.3: The original MS-graph (left) and its modification (right).

LoanProcessing, SimplifiedPropertyValuation → SimplifiedLoanProcessing, HighNumberOfHighNetWorthClients → LowCreditLoss, GoldStandardDiscount → HighNumberOfHighNetWorthClients, IncomeStatementReceived → StrictCreditAssessment.

Although we acknowledge that maintaining a formal representation of domain knowledge can be laborious, it should be emphasized that such an exercise has many advantages beyond this paper. The process of developing a domain knowledge base forces stakeholders to make their knowledge and assumptions (as well as terminology) precise, which can highlight potential inconsistencies, and promote their resolution by coming to a shared understanding. In the best case, domain specific knowledge is already available (in an accessible format) and the background knowledge does not need to be built from scratch. The increasing development of domain ontologies is a step in this direction.

Wellformed Motivation-Service Graph: We refer to an MS-graph that does not contain alternative refinements (or decompositions) as an *AND-MS-graph* (also referred to as solution graph in Chapter 4). Any *general MS-graph* (which may contain alternative refinements) can be represented as a set of distinct AND-MS-graphs. We use the function $\Delta_{AND}(G)$ to denote all maximal AND-MS-graphs that are sub-graphs of a given MS-graph G. Intuitively, a maximal AND-MS-graph is any maximal sub-graph of the original graph that retains the AND refined (realization) property. For example, the MS-graph of Figure 6.3 (left hand side) has *two* AND-MS- (sub) graphs (one includes T1 and Se1, the other includes T2 and Se2). In the following, we refer to a vertex v as a root vertex if and only if there does not exist a (hyper-) edge e in the MS-graph, such that v is an element of the targets of e. An AND-MS-graph is

wellformed if it satisfies the properties below.

1. All root vertices are desired and of type "vision". A vertex of type "vision" can only point to vertices of type "goal". A vertex of type "goal" can only point to vertices of type "goal" or "strategy". A vertex of type "strategy" can only point to vertices of type "strategy" and "tactic". A vertex of type "tactic" can only point to vertices of type "tactic" and "service".

2. The conjunction of the assertions associated with all vertices must be consistent with the domain knowledge base, since the organisation intends to concurrently realize all of them (i.e. there is no sequencing knowledge encoded that might suggest that one should be achieved before another). Formally, let $Cn(T)$ denote the closure under logical consequence of a set of sentences T (in $\mathcal{L}$) then $\perp \notin Cn(V \cup KB)$.

3. All target vertices of any edge must *minimally entail* the source vertex of the respective edge. In other words, the targets offer a more detailed description of the source and there is no redundancy in the additional detail (hence the minimality requirement). Formally, for all hyperedges $e = (x, Y)$, where $(x \cup Y) \subseteq LIB$ it holds that $x \in Cn(Y \cup KB)$ (entailment) and $x \notin Cn(Y' \cup KB)$ for all $Y' \subset Y$ (minimality). We have adapted this property from [30]. Observe, that in our running example the effects of service Se1, Se1 and Se3 respectively entail the tactics T1, T2 and T3 and are hence part of the MS-graph.

Although our discussion thus far does not refer to service pre-conditions, we could leverage these to flag potentially infeasible service-level realizations of BMM tactics - for instance, if a sequencing of these service elements such that the pre-condition of each service are derivable from the post-conditions of the preceding services does not exist.

A general MS-graph is well-formed if and only if all $\mathscr{G} \in \Delta_{AND}(G)$ are well-formed. In our example, the strategy S3 and tactic T4 (part of the motivation library) do not participate in the MS-graph of Figure 6.3 (left hand side), as doing so would result in a derivable AND-MS-graph which is not wellformed (i.e. it is inconsistent) and the organisation has given precedence to the other elements.

Discussion: The goal model elaborated in Chapter 4 differs from the proposed formalization of a BMM model (given above) in the following ways. First, instead of

maintaining a set of valid solution graphs (here we refer to them as well-formed AND-MS-Graphs), a single MS-graph is maintained for which all AND-MS-sub-graphs must be wellformed (the properties of a wellformed AND-MS-Graph are in many ways similar to the properties of a valid solution graph, e.g. both require all goals to be consistent, but do not need to contain a maximal consistent set of desired goals - we discuss the implications of this later.). While this essentially corresponds to a set of wellformed AND-MS-graphs, there are some important differences. (1) A single MS-Graph does not allow us to consider distinct root goals. (2) Solution graphs (in the set of solution graphs) are treated independently from each other, while wellformed AND-MS-sub-graphs are linked to each other via the MS-graph they are derived from and remain part of. This has disadvantages when the MS-graph needs to be maintained. Figure 6.4 shows a MS-graph G_1 and its two AND-MS-sub-graphs $G_{1.1}$ and $G_{1.2}$.

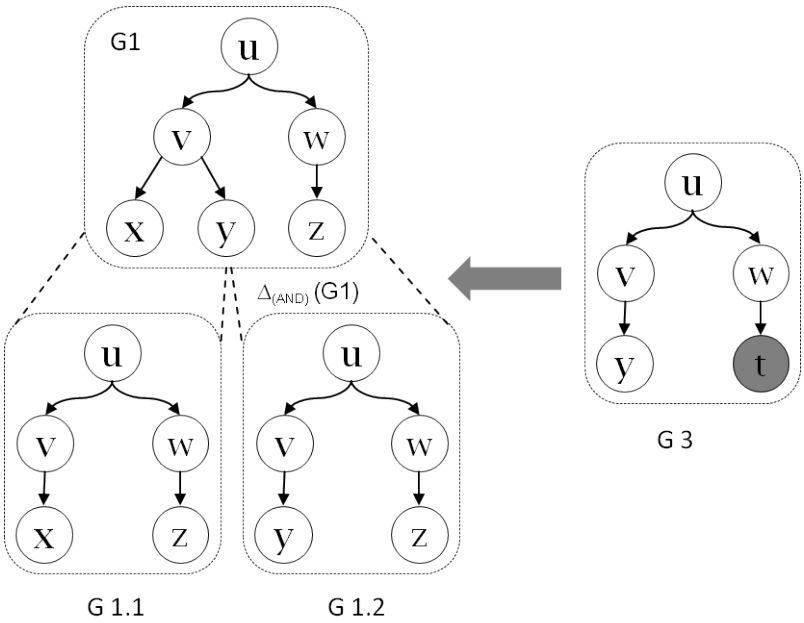

Figure 6.4: Advantage of goal mode formalization proposed in Chapter 4 (1).

*Let us assume that we want to consider the option/solution shown in the AND-MS-graph G_3 where **v** is refined by **y** and **w** by **t**. In the goal model formalization provided in Chapter 4, this is simply done by adding G_3 as another solution graph to the set of solution graphs. In the formalization of a BMM model, we would have to modify the corresponding MS-graph by adding a refinement from goal **w** to goal **t**. The resulting goal graph is denoted by the MS-graph G_2 shown in Figure 6.5.*

Observe that G_3 is part of all of the AND-MS-sub-graphs of G_2, but the AND-MS-

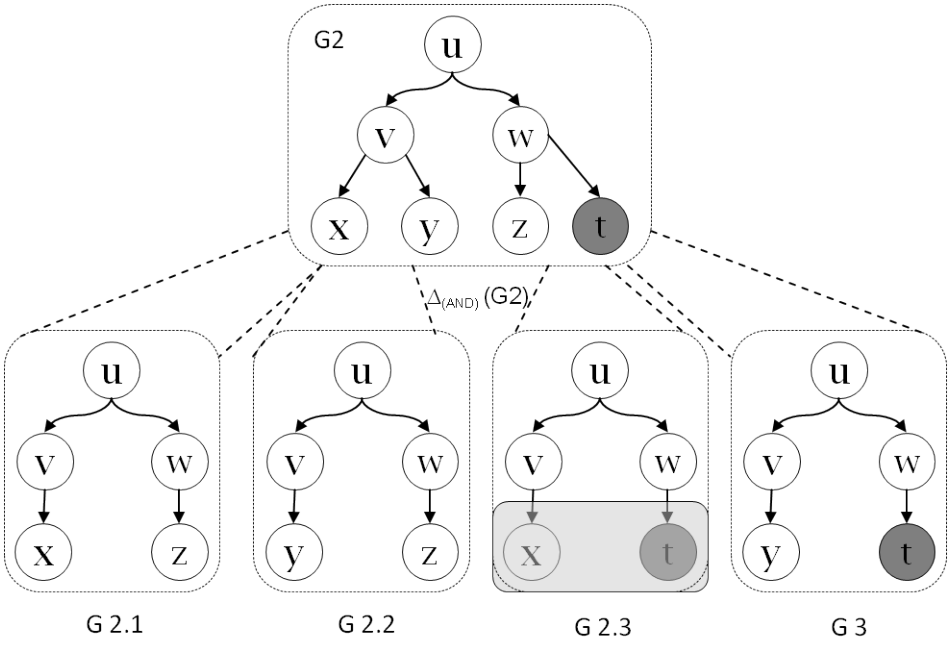

Figure 6.5: Advantage of goal mode formalization proposed in Chapter 4 (2).

graph $G_{2.3}$ for which the goals $\boldsymbol{x}$ and $\boldsymbol{t}$ may be inconsistent is also derivable. In this case, the MS-graph G_2 would not be well-formed (i.e. not all AND-MS-sub-graphs are well-formed). In fact, the given change request cannot be implemented without violating the wellformedness property of the MS-graph G_1. This is not a problem if the AND-MS-graphs (i.e. solution graphs) are treated independently of each other. Alternatively, we could have relaxed the definition of a wellformed MS-graph (which is admittedly too strict in some cases) such that not all AND-MS-sub-graphs are required to be well-formed. However, this would allow us to consider options (i.e. AND-MS-sub-graphs) that are non-wellformed, which is not the case in the more general goal model formalization described in Chapter 4.

k-level entailment: We will now define a hierarchical entailment relation for MS-graphs called k-level entailment. k-level entailment permits us to answer whether a particular assertion is derivable (or not derivable) from a particular level onwards in the MS-graph. This is useful in (but not restricted to) identifying and resolving non-compliance (i.e. inconsistencies with a regulation) in MS-graphs.

k-level vertices are a subset of MS-graph vertices, such that for each element v, there exists a path of length k from *a* root vertex to v, *or* v is a leaf vertex and there exists a path from a root vertex to v with a length less than k. The root vertices

themselves are 0-level vertices. For example, Figure 6.3 indicates the level of each vertex. Note that (although not the case in our example) the definition deliberately permits vertices to be associated with more than one level, such that a vertex can participate as the target of various relations across multiple levels.

For an *AND-MS-graph* $\mathscr{G}$, an assertion α is k-entailed if and only if all k-level vertices together with the $\mathcal{KB}$ entail the assertion and there does not exist a lower level (lower in terms of the value of k) in the MS-graph for which this is the case, i.e. k is the "earliest" (i.e. lowest) level at which the assertion is entailed. We use $\mathscr{G} \models_k \alpha$ to say that $\mathscr{G}$ k-entails an assertion α. For example the assertion `LowDocumentation` is entailed at level 3.

From the definition of a wellformed MS-graph, it follows that if an assertion α is k-level entailed, then α is entailed in all levels i that follow after k. We refer to this as α being *i-consequence*. For a general MS-graph G, an assertion α is *strongly k-entailed* (denoted by $G \models_k^{str} \alpha$) if and only if all AND-MS-graphs derivable from G have α as an i-consequence and $k \leq i$. Conversely, we say that a general MS-graph G *does not entail* an assertion α (denoted by $G \not\models^{str} \alpha$) if and only if there does not exist an AND-MS-graph derivable from G that entails α at any k.

6.1.1.3 Maintaining Motivation-Service Graphs

We now show how to maintain an organisation's motivation model (i.e. the BMM) and service capabilities (i.e. the catalogue) in their conjoint MS-graph representation. There can be many change drivers. Organizations change their motivations to respond to dynamic business contexts, e.g. by the addition of motivation elements (via an *add-vertex* operator) or the removal of motivations (via a *remove-vertex* operator) (the modification of a motivation can be viewed as a removal of the prior version followed by the addition of its modified version). Another situation where these same operations might be necessary is the need to modify motivational elements because their realizing service becomes unavailable (and no service replacement can be identified). An MS-graph may also have to be modified to meet compliance obligations, e.g. to ensure that an assertion is entailed by the graph (via an *entail-assertion* operator) or is not entailed by the graph (via a *not-entail-assertion* operator).

Discussion: As discussed in Chapter 5.2.4, a request to make a goal x derivable is best addressed by adding the goal explicitly to the graph - an "entail assertion operator" is hence not required.

The properties of the "not-entail-assertion operator are inspired by the AGM logic

of theory change [1] and its belief contraction postulates. A key insight from this framework is the need to minimize change to a body of knowledge encoded in an artifact such as an MS-graph. We therefore require means to assess the extent of change, which might be encoded in a notion of proximity between two MS-graphs.

Motivation-Service Graph Proximity: Given MS-graphs G, G' and G'', we say that $G' <_G G''$ if G' is "closer" to G than G''. The relation $<_G$ may be defined in various ways. From a *graph theoretic* perspective, we may view G' as closer to G if it has more vertices and edges in common with G. This intuition places an equal weighting on all elements of the graph. However, this is not sensible for MS-graphs, since some vertices justify others and hence they should be given precedence (e.g. a strategy is justified by the goal it aims to bring about). We prefer an MS-graph that preserves the original graph up to level $k+1$, over one that preserves the original graph up to level k. On the basis of this argument, one possible intuition for defining $<_G$ could be *k level set intersection cardinality*. MS-graph G' is preferred over G'' if G' shares more vertices with the original MS-graph G at level 0. In case of a tie, G' is preferred over G'' if it shares more vertices with G at level $k+1$, and so on.

"not entail assertion" operator: The "not entail assertion" operator (denoted by $\ominus$) minimally modifies an MS-graph to obtain one where an assertion is strongly not derivable. A key application is in compliance management. In the running example, if new regulations require that each loan must be backed up by sufficient documentation, the organisation must ensure that the assertion $\alpha =$ `LowDocumentation` is not derivable from the MS-graph (in the example α corresponds to a single assertion but may in general be a consequence of a set of assertions). Our formalization can handle this. We offer the following normative properties for the "not entail assertion" operator.

1. *Wellformedness:* For a wellformed MS-graph G, $G \ominus \alpha$ is wellformed.
2. *Success:* For a wellformed MS-graph G where $G \ominus \alpha = G'$, $G' \not\models^{str} \alpha$.
3. *Vacuity:* For a wellformed MS-graph G where $G \not\models^{str} \alpha$, $G \ominus \alpha = G$.
4. *Desire-inclusion:* For a wellformed MS-graph $G = \langle V, E, f \rangle$ where $G \not\models^{str} \alpha$, there does not exist a $g \in LIB^{des}$ such that $g \notin V'$ and the MS-graph $\langle V' \cup g, E', f \rangle$ satisfies the above properties. In other words, the resulting MS-graph should include as many desired conditions from the library as possible.
5. *Minimal-change:* For a wellformed MS-graph G where $G \ominus \alpha = G'$, there does not exist a G'' that satisfies the four properties above and $G'' <_G G'$. In other words,

the operation should return the MS-graph that satisfies the above conditions and is closest to the original graph.

Definition 28. *Given a wellformed MS-graph G and an assertion α, let the operation $G - \alpha$ return a set, such that each $G' \in (G - \alpha)$ satisfies the properties (1)-(5). We define $G \ominus \alpha = s(G - \alpha)$, where s is a selection function that returns an element in $G - \alpha$.*

In definition 28, $\ominus$ represents a class of operators, parameterized by the proximity relation $<_G$ and the selection function s. The removal of an element from G might permit the addition of previously inconsistent desired motivations from the library to G. In our example, the vertices T2 and Se2 of the MS-graph make α derivable, which might lead to their removal. This may permit the inclusion of the previously inconsistent library elements S3 and T4 to the MS-graph.

The MS-graph in Figure 6.3 (right side) denotes the outcome of the "not entail assertion" (with α = `LowDocumentation`) operator (instantiated by k-level intersection cardinality) with respect to the original MS-graph (right side)[6]. Tactic T4 can be included in the MS-graph but it is not realized by a service (or combination of services); this would have to be flagged to the user, who could then decide to adopt the required functionality, drop the unrealized options from MS-graph, or leave the model unchanged (i.e. we do not require that all motivation elements are realized by a service to avoid the exclusion of reasonable options). Furthermore, the service Se2 is not part of the MS-graph any more (and flagged as such to the user), as it is part of the service catalogue but does not contribute to bringing about any organisational motivation.

Discussion: The "desire inclusion" property is part of the properties of the "not entail assertion" operator for an MS-graph, and requires that the resulting MS-graph should contain as many desired goals as is consistently possible. In Chapter 5.2.4, we dropped the "desire inclusion" property from the "not entail assertion" operator, but instead required a solution graph to contain as many desired goals as is consistently possible for it to be valid. On first sight, this does not make much difference, since the "not entail assertion" operator described in Chapter 5 requires all solution graphs to be valid and hence contain a maximal consistent set of desired goals, but it has the following subtle consequences. First, the current (wellformed) MS-graph may not contain a maximal consistent set of desired goals, unless the last modification was a

[6] In our example all elements in the motivation library are desired, and recall that the desire inclusion property has primacy over the minimal change property.

successful application of the "not entail assertion" operator. Second, it allows us to consider solutions in which not all desired goals in the MS-graph are satisfied. For example, let us assume that in Figure 6.4, the goals $\boldsymbol{u}$,$\boldsymbol{v}$,$\boldsymbol{w}$, $\boldsymbol{x}$, $\boldsymbol{z}$ *are desired goals, then the solution denoted by the AND-MS-sub-graph* $G_{1.2}$ *does not satisfy all desired goals in the graph. This is handled by ensuring that desired goals are not part of any alternative refinements, and in doing so the MS-graph can be viewed as a commitment to a particular set of maximal consistent desired goals and their refinements (including alternative refinements). The goal model formalization described in Chapter 4 also addresses this deficiency, but also allows us to capture solutions with distinct sets of desired goals.*

6.1.1.4 Related Work

A growing literature studies services in the context of organisational motivation models (usually represented in i*[165], KAOS [28], BMM [70], or some modification or simplification of these). While most of the work focuses on (modelling and) verification techniques (e.g. [24], [86], [21], [90], [112]) and techniques for handling change at the service level (e.g. [2], [74]), little has been done to study the dynamics of changes to the motivation representation. The works by [170], [64], [42] are notable exceptions, but they do not explicitly take into account the service level. Zowghi et al. [170] address goal evolution by defining a belief revision operator that maps one (default) theory to another theory. However, the hierarchical relationships among goals are not considered. MacNish and Williams [113] revise hierarchical goal structures using a maxi-adjustment transmutation (a belief revision technique). Our work addresses some deficiencies highlighted by the authors. For example, maxi-adjustments only ensure that the sub-tree rooted at one of the conjuncts is retracted, while the wellformedness definition requires that all conjuncts are removed (if not otherwise justified). Ernst et al. [42] address the problem of finding minimal modifications to an existing requirements selection in the context of change. Their formalization emphasizes structure (similar to our work), however the formal assertion of the requirements are not the subject of their analysis (requirements are treated as black boxes) and therefore their analysis is of a different nature. The same holds for Liu et al. [110], who propose an (i* based) formalism for service requirement evolution. A (hierarchical) graph representation of goals and services is also considered in Levi et al. [106], but no formal representation provided. On the other hand, Galster and Bucherer [51] also provide a hypergraph representation of business motivation and service capabilities. However,

their graph representation does not relate formal assertions (but non-technical descriptions) and the amenability of (F-) hypergraphs to represent alternative refinements, so they are not considered. The correlation of services with BMM models has also been described in [112]. The correlation is established by annotating services with WS-Policies which make reference to elements in the BMM, and therefore require us to manually establish the relation. In our framework, relations between service and motivation elements are derivable from their respective formal assertion.

6.1.1.5 Conclusion

We showed how a Motivation-Service Graph (MS-graph) can be used to provide a combined representation of organisational motivation (represented in a BMM) and its services landscape and how this allows us to handle change at the motivation level and assess its impact to the service landscape. In particular, we discussed a maintenance operator class (the "not entail assertion" operator) for compliance driven change.

6.1.2 Discussion

In Chapter 6.1.1, we have shown how to maintain goal models correlated with service descriptions. We also highlighted and discussed the difference between the goal model formalization given in Chapter 4 and the one given above (Chapter 6.1.1). From now onwards, we will continue to work with the formalization of the goal model provided in Chapter 4, but will adopt the following insights from Chapter 6.1.1.

(1) Service descriptions can be linked via their post-conditions to leaf goals of one or more solution graphs (given that both are expressed in the same formal language). A service description can be linked to a goal in a solution graph (i.e. we say that the goal is realized by the service description), if the service's post-condition, together with the domain knowledge base, entail the goal. A solution graph is completely satisfied if for all of its leaf goals there exists a description of system behaviour, e.g. a service description, that realizes it (a solution graph for which only some goals are satisfied is consequently referred to as partially satisfied).

(2) Service post-conditions and goals are both sentences in a formal language and hence can be represented together in a solution graph.

(3) Given a service catalogue, represented as a set of services and one or more solution graphs, we can check whether a solution graph is completely satisfied by the service catalogue by testing whether, for each of its leaf goals, there exists a service in the catalogue whose post-conditions entail the goal. It is worth highlighting that this process can be automated.

Recall, in our formalization of a goal model, we only distinguish between desired and dependent goals and do not explicitly make the fine grained distinction between desired goals of compulsory and optional character. This permits situations in which a solution graph is valid, but does not contain all compulsory goals, since a compulsory goal may be conflicting with an optional goal and the latter has been given precedence. While this is true, the proposed goal model formalization is general enough to avoid these situations. Observe, (1) valid solution graphs that contain all necessary goals and as many optional goals as possible are a subset of the set of all valid solution graphs that can be constructed from the goal library; (2) the stakeholders are in full control over which solution graphs they want to consider, i.e. which solution graphs are part of $\mathbf{G}$. In a static setting (i.e. where there is no change), the above is simply addressed by restricting all solution graphs in $\mathbf{G}$ to the ones that contain all necessary goals and as many optional goals as possible. In a dynamic setting (when a maintenance operator is applied), the selection function of the respective operator can be encoded in such a way that a goal model with a set $\mathbf{G}'$ (for which all solution graphs contain all necessary and as many optional goals as possible) is selected from all goal models with corresponding sets $\{\mathbf{G}'...\mathbf{G}''\}$ that satisfy the respective operator properties. Note, this requires the existence of at least one set $\mathbf{G}'$ that satisfied this property. Note, if no such set exists, then the respective change request, e.g. adding a new necessary goal to all solutions graphs in $\mathbf{G}$, must be in conflict with the existing necessary goals. In the REKB [42, 44] and the REFORM framework [63, 64], this is handled by demoting some of the conflicting necessary goals to optional goals and then reapplying the change operation. In our work, the stakeholder simply chose one of the valid outcomes of the applied maintenance operator.

At the beginning of this chapter, we formalized the functional part of the requirements problem as follows.

Given domain knowledge and assumptions $\boldsymbol{KB}$*, compulsory goals* $\boldsymbol{G}_C$*, and optional goals* $\boldsymbol{G}_O$*, a solution to the functional aspect of the requirements problem is a specification* $\boldsymbol{S}$*, such that* $\boldsymbol{KB}, \boldsymbol{S} \models \boldsymbol{G}_C, \boldsymbol{G}'_O$*, where* $\boldsymbol{G}'_O$ *is a maximal subset of* $\boldsymbol{G}_O$ *such that*

$\boldsymbol{K}, \boldsymbol{S}, \boldsymbol{G}_C, \boldsymbol{G}'_O$ *is consistent.*

Given that the valid solution graphs in the set $\mathbf{G}$ are restricted to the ones that contain all necessary goals and as many optional goals as possible, it is easy to see that a set of services $\mathbf{S}$ which completely satisfies a solution graph $\mathscr{G} \in \mathbf{G}$ is a solution to the functional part of the requirements problem, since $\mathcal{KB}, \mathbf{S} \vDash f_{des}(\mathscr{G})$ (and $f_{des}(\mathscr{G})$ contains all necessary goals and as many optional goals as possible). We can hence reformulate the functional requirements problem as the search for a set of services that completely satisfies a solution graph in $\mathbf{G}$. Note, there can be more than one service catalogue subset which completely satisfies a solution graph. Each of these options can be captured by a solution graph (treating the respective set of services as alternative refinements of the original solution graph, as illustrated in Figure 6.6).

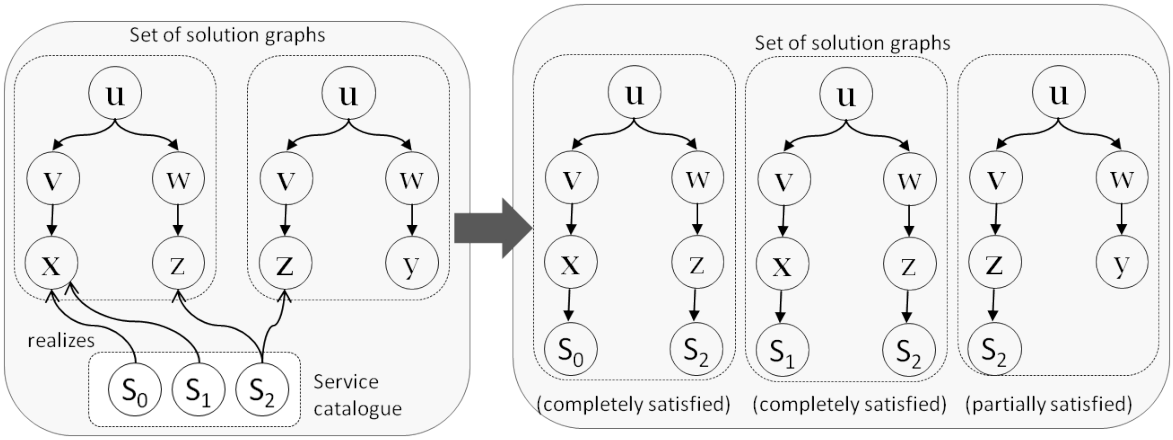

Figure 6.6: Solution graphs and their realization via services in the service catalogue.

This allows us to represent all solutions to the functional part of the requirements problem via completely satisfied solution graphs in $\mathbf{G}$. In the case that the set of solution graphs is initially empty, the "completion" operator can be applied to uncover the space of all valid solutions graphs which are then juxtaposed against the service catalogue to identify the solution graphs that are completely satisfiable.

So far, we have addressed the functional part of the requirements problem in a static context. In a dynamic context, i.e. when a maintenance operator is applied on a set of completely satisfied solution graphs, then the solution graphs of the returned set are not necessarily completely satisfied. There are at least three modalities to proceed from this point. (1) Solution graphs that are not completely satisfied as a consequence of applying the operator are removed. (2) The set of solution graphs

is left unchanged, i.e. we do nothing. (3) The completion operator is applied to search for goals in the goal library that allow the solution graphs to be refined so that the resulting solution graphs are completely satisfiable by services in the service catalogue. Of course a combination of (1), (2) and (3) is possible, as well. For example, we may want to remove solution graphs that cannot be completed by the "completion operator". Alternatively, we may want to retain these solution graphs (i.e. do nothing) in anticipation of future situations where additional goals in the goal library permit further refinements.

Note, in the case where there exists at least one completely satisfiable solution graph, after the application of a maintenance operator, then an answer/solution to the functional aspect of the requirements problem can be provided. In the case where a completely satisfiable solution graph in $\mathbf{G}$ does not exist, this can be (temporarily) accepted or the completion operator is used to search for a completely satisfiable solution graph. Note, while a completely satisfiable solution graph may not be part of $\mathbf{G}$, it may however be derivable from the goal library. This is due to the fact that the set $\mathbf{G}$ is not required to contain all valid solution graphs.

In the following, we critically discuss some of the assumptions we have made as well as other challenges that have not been addressed at this point. *First*, we have assumed that the service post-conditions are specified in the same language as the goals in the goal library. This may not always be a realistic assumption, since the ontological gap between the vocabulary used at the goal- and at the operational levels is hard to bridge. In [101], we have proposed an ontological-driven approach to match goals with services. Amongst others, this approach takes into account different degrees of ontological distance between service post-conditions and goals. As a last resort, correlations between services and goals can be manually established. This is also the suggested approach of Ernst et al. [42, 44] (Ernst et al. refer to descriptions of system behaviours in more general terms as "tasks"). *Second*, we assumed that the post-condition of a (business) service is deterministic. However, business services may encapsulate complex business processes which allow (per design) alternative execution paths and hence may result in different (i.e. non-deterministic) outcomes. In other words, a business service may have more than one post-condition. This situation has been considered in Koliadis et al. [94], where a goal is considered as realized by a business service (in their case a business process design) if the effects of all the alternative execution scenarios entail the goal. In the case that service descriptions have non-deterministic outcomes, they can be correlated to goals following Koliadis et

al. [94].

6.2 Non-functional Aspects

In this second part of the chapter, we summarize our previous work on Green Business Process Management [79, 78, 77] in Chapter 6.2.1, 6.2.2 and 6.2.3. In Chapter 6.2.4, we discuss how this is the basis for addressing the *non-functional aspects of the requirements problem.*

6.2.1 An approach to assess the (environmental) performance of business process designs

In the following, we recapitulate our work titled "Towards Green Business Process Management" [79]. Some of this material has also appeared in Ghose et al.[56] and is my contribution as a co-author. In particular, we show how resources and their usage-cost relationship can be modelled, and how linking these models to process designs allows us to derive the expected carbon-emission performance of processes at design time.

6.2.1.1 Introduction

There is a common a agreement that humanities global carbon footprint must be reduced. Due to external pressures such as legislative requirements [47, 11], as well as an increased awareness of the general public, e.g. choosing products from organizations with an environmentally sustainable profile, organizations are forced to understand the source of their carbon footprint and to minimize it where possible. We argue that by transforming the problem into the domain of Business Process Management (BPM) we can leverage the rich expertise in this field to address issues associated with identifying areas for improvement, understanding the implications and eventually reducing carbon emissions.

Business Process Management (BPM) is useful in understanding and improving an enterprise's processes, but can also be used to model and improve manufacturing and other "physical" processes. [56] To leverage BPM technology we first need to be able to assess the (environmental) performance of a business process *design.* This could be done by measuring the average performance of process execution instances and associating this data with the process design. However, this is not possible in

cases where the process design has not yet been instantiated, e.g. when we want to know the *expected performance of a process re-design*. In this paper, we show how the expected performance of a business process design can be derived during design time by modelling resources and relating them to activities of the process designs.

A commonly used environmental performance indicator/measure is carbon dioxide equivalent (CO2-e), which is an expression of other greenhouse gases in their carbon dioxide equivalent by their global warming potential (CO2 itself has a global warming potential of 1). Other measures including *waste generated* and *water consumed* must also be considered to get a complete picture of the environmental performance of a process, but for ease of elaboration we only consider one factor, i.e. CO2-e (we addressed the case of multiple heterogeneous measures in [78], which is summarize in Chapter 6.2.2 of this thesis).

In the following, we first describe a technique for accumulating emission values associated with activities of a process design to get the process cumulative emission values. Chapter 6.2.1.3 describes how resources and their usage-relationship can be modelled and how such models can be correlated with activities of a process design to infer/estimate its emission values. In Chapter 6.2.1.4 we report on our experiences of applying these techniques. Chapter 6.2.1.5 positions this work in the literature and Chapter 6.2.1.6 concludes it.

6.2.1.2 Accumulating carbon emission values across process designs

An *emission annotation* is a textual assertion associated with activities of business process designs and states the CO2-e value the activity would emit if executed. Associating this value with an activity is intuitive, since carbon emissions are either the direct *consequence of an action* (e.g. fugitive emission caused by digging for coal) or the result of *consuming resources* while performing an action (e.g. driving a van to deliver a package).

An emission annotation is either directly annotated to an activity, or (more interestingly) returned by a functional call to a correlated resource model. The latter is covered in Chapter 6.2.1.3 and for now we assume that the activities of a given business process design are already annotated with emission values. An emission scenario at a given point in a process model is the sum of the (cumulative) carbon annotations that would have been emitted up to that point. Accumulating emission annotations through a business process design is a non-trivial exercise since there might be various paths that can be traversed during process execution. For example, the simple

business process design shown in Figure 6.7 has two paths and hence two emission scenarios, i.e. one for traversing the "upper path" and one for traversing the "lower path".

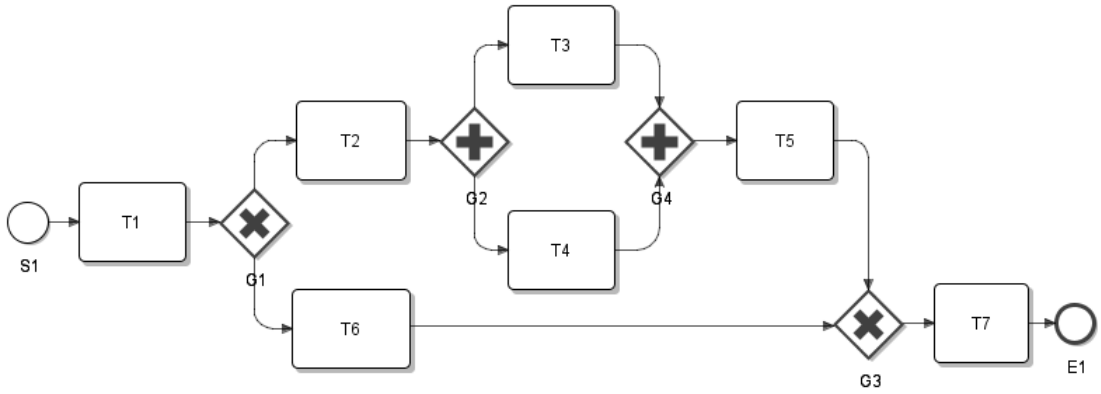

Figure 6.7: Example of a business process design in BPMN.

In the following, we describe a technique for accumulating emission annotations up to any selected point in the process design. This technique is based on the effect accumulation technique described in Hinge et al. [75].

Let Ti and Tj be a pair of contiguous activities connected by a control flow link. The set of (cumulative) emission scenarios at Tj consists of all distinct values $es(i) \oplus ea(j)$, where $es(i)$ is an emissions scenario associated with Ti, $ea(j)$ is the emissions annotation of Tj and '$\oplus$' is an accumulation operator. Each such distinct value constitutes an emissions scenario for Tj. We deal with AND merges (see Figure 6.7 label *G4*) in the following manner. If Ti and Tj are the only two activities immediately preceding an AND-merge, and Tm is the activity immediately following it, the set of emissions scenarios at Tm consists of all distinct values $es(i) \oplus es(j) \oplus ea(m)$ for every distinct pair $es(i)$ and $es(j)$ such that $es(i)$ is an emissions scenario associated with Ti and $es(j)$ is an emissions scenario associated with Tj, while $ea(m)$ is the emissions annotation associated with Tm. In the preceding setting, if we replace the AND-merge with an XOR-merge (see Figure 6.7 label *G3*), we proceed by $es(i) \oplus ea(m) = es(m')$ and $es(j) \oplus ea(m) = es(m'')$, where $es(m')$ and $es(m'')$ are distinct.

This technique allows us to compute all emission scenarios for process design. The range of values between carbon emission scenarios can be large, i.e. there can be scenarios with comparatively "low" and scenarios with comparatively "'high" emissions. Using these values in the parent-process can result in an even wider range. Due to this potential issue and ease of communicating emission figures, it is of interest to the

users to have a single emission figure for each process design. The approach of computing an average of all emission scenarios was considered. This approach essentially places equal weighting on all scenarios. However, we feel that this approach might be erroneous due to the fact that some rarely executed scenarios might skew the mean emission value. To avoid this issue, the process designs can be enriched with probability figures. This can be done, for example, by evaluating process logs, which is generally a powerful technique enabling an enterprise to evaluate and understand the performance aspect of the qualitative attributes associated with their processes. If no process log is available, one can also report on the experience of process users.

Probability figures are annotated for all *outgoing exclusive gates* (XOR-split see Figure 6.7 label *G1*)[7] of the process design. The sum of the probability annotations of all outgoing paths of a XOR-split has to be 1. We do not have to annotate sequence flows and parallel splits since they are always executed (by design). We accumulate the probability annotations across the process design to find the probability of an emission scenario. This can be done by using the technique described previously, substituting addition with multiplication.

6.2.1.3 Resource Nets and their correlation with process designs

The amount of carbon emission emitted by an activity is primarily influenced by the resources it uses. For example, driving a large SUV car (resource) causes more emission than driving a small car (resource) for a "delivery to customer" activity. Another factor is *how the resource is used*. Driving a car with many accelerations causes more emission than foreseeing driving with less accelerations. Furthermore, the *intensity with which a resource is used* has an influence on the emission impact. Staying with the previous example it makes a difference whether we use a car for 100km or 200km. In addition, a resource might require another resource to be used and the carbon emission figure is also influenced by how it is used, with which intensity, and their sub-resources. For example, a sub-resource could be the fuel used for combustion and the associated carbon emission for gathering and transporting the fuel to the petrol station. To sum up, the factors are:

1. What resource/type of resource is used?
2. How is the resource used?

[7] In the following we do not consider OR-gateways. Semantically, an inclusive OR can be viewed as a combination of an XOR and an AND

3. With which intensity is it used?
4. Which sub-resources does it use?

There is a clear interplay between resources and activities and their impact on a system's environmental performance. An approach to fathom this interplay of resources with other resources and activities is modelling it. We briefly describe how the resource model can be captured as a directed acyclic graph, provide guidance for instantiating it and finally show how it can be used to derive the environmental performance of a business process during design time.

Resource Nets: The usage-cost relationship among resources is represented by a directed acyclic graph, which we refer to as *Resource Net*. Vertices are used to denote resources and their usage modes (to account for the fact that resources can be used in different ways), and are associated with a cost per scale value (e.g. 1 kg CO2-e *per* kilometer, or 10Wh *per* hour). Resources are linked via directed edges to denote that the edge's target resource uses the edge's source. Each relationship between resources is associated with a number that denotes the *usage intensity*, e.g. to denote that the car/van is used for 25km.

Formally, a Resource Net is an acyclic graph $\langle V, E\rangle$, where elements in V are resources, each denoted by a tuple $\langle ID, UMS, C, CAP, \rangle$ where ID is the unique identification of the resource, C is a set of cost types (e.g. CO2-e, dollars, etc.), and CAP is a set of capabilities associated with a resource.[8] and UMS is a non-empty set of usage modes. Each usage mode UM is a tuple $\langle id, Scale, cap, \Omega\rangle$, where id is the unique identification of a usage mode, $Scale$ is a tuple $\langle unit, quant\rangle$ where $unit$ denotes a measure (e.g. kilometre, litre, piece, etc.) and $|unit| = 1$ and $quant \subseteq R^+$ quantifies the unit (e.g. 1km, 0.1L, 1piece, etc.). $\Omega : Scale \times C \rightarrow \mathcal{R}^+$, is a function and quantifies the cost for the scale of a usage mode while cap is a set of capabilities required by the usage mode, where $cap \subseteq CAP$. E is a set of directed labeled edges of the form $\langle\langle u, v\rangle, \langle UM_u, UM_v, I\rangle\rangle$, such that (1) u and v are resources, where $u, v \in V$, (2) $UM_u \in UMS$ denotes a usage mode of resource u, (3) $UM_v \in UMS$ denotes a usage mode of resource v, (4) the order of u and v dictates that a directed edge points from u to v, subsequently the interpretation of the corresponding usage model (UM_u and UM_v) is UM_v requires UM_u with the intensity I. [9], and (5) $I \in \mathcal{R}^+$ denotes that

[8] For example a car might have the capability to transport five people

[9] This implies that there can be several directed edges between two resources - one for each usage cost relationship between two usage modes.

$UM_v.Scale.quant$ requires $UM_u.Scale.quant \times I$.

An example of an instantiated Resource Net (graph) is given in Figure 6.9. The following procedure provides guidance in instantiating a Resource Net.

1. Select a resource.
2. Identify all relevant usage modes of the selected resource by asking: (a) Is this resource used in different ways? (b) Do the costs of the identified usage modes vary considerably? [10]
3. Focus on usage modes that potentially result in high costs.
4. For a selected usage mode (a) identify its scale - how the usage intensity can be measured; (b) identify all relevant cost types (e.g. carbon emission, waste, dollar); and (c) quantify the cost types for the usage mode's scale (e.g. x kg CO2-e per km).
5. Identify all sub-resources required by each usage mode.
6. Repeat the procedure until all resources and their usage modes are identified.
7. Associate each resource's usage mode with the required sub-resource usage mode and specify the usage intensity.

Note, finding a required sub-resource and its provided usage mode could be partially automated by searching for appropriate capabilities, associated with a usage mode, across a store of existing resources.

Correlating Resource Nets with process designs: Processes can be captured at different levels of abstraction, ranging from abstract process designs to decomposed sub-processes and finally to detailed atomic activities. We suggest the following bottom-up procedure to identifying the level of abstraction at which the Resource Net should be correlated with the business process design:

1. Identify all process designs.
2. Order them according to their level of abstraction. The general rule applies that processes are more abstract than their sub-processes, and atomic activities (that cannot be decomposed any further) are on the lowest level of abstraction.
3. Select the process designs at the lowest level of abstraction.
4. Check whether resources can be associated in reasonable time and effort.

[10] We are aware that the answer can be hard to quantify exactly, but note that an educated guess can provide enough insights for the moment.

5. If no resources could be found, consider the next level of granularity and repeat the previous step.

Having identified an appropriate level of abstraction, we correlate a given process design with a Resource Net by annotating its activities with a *functional call*. This call states the resource, the leveraged usage mode, and its usage intensity, for example *use(Printer, Scanning, 5 pages)*. The expression is evaluated and returns the corresponding emission figure, as emission annotation, back to the process design. Note, the process analyst can also directly annotate the emission annotation. This is necessary in situations where carbon emissions are not the result of using a resource, but rather the direct consequence of the activity (e.g. CO2-e impact of cutting a tree). The emission annotations can be extrapolated to the higher level processes (bottom-up) by repeated accumulation of the emission annotations across the process designs. This results in generating a carbon emission value at the process environment level (see Figure 6.8) and can be the starting point for process analysis and optimization.

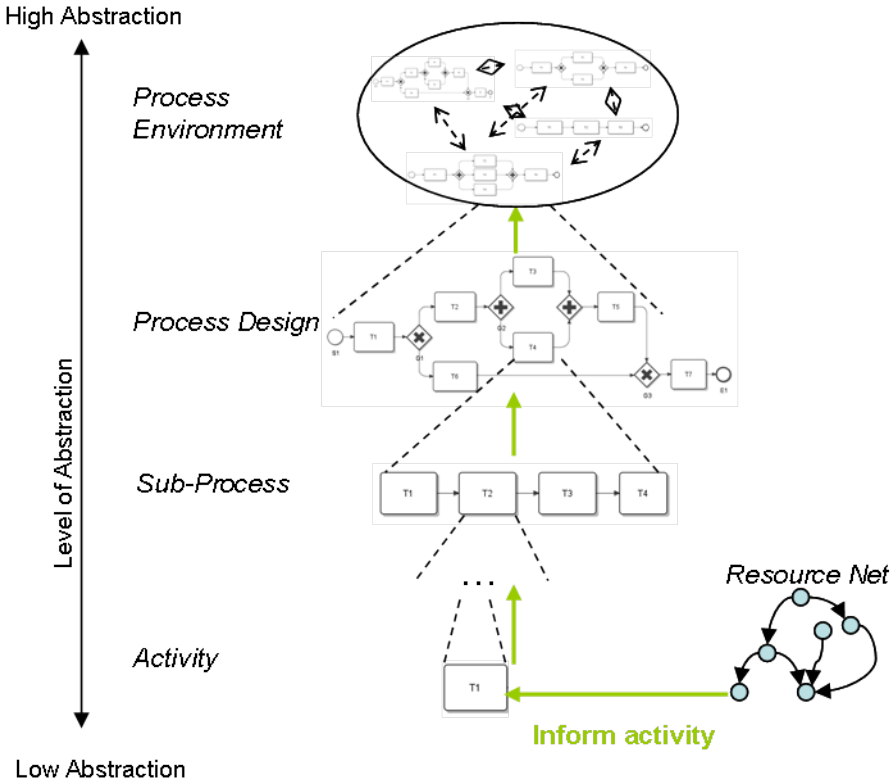

Figure 6.8: Business processes at different levels of abstraction.

Although the introduced Resource Net should be capable of modelling a wide range of resources and their usage-cost-relationships, we note that the user should focus

on relevant resources that are measurable and contribute considerably to the carbon footprint of the business. The general rule applies that the more resources (starting with the most relevant) are considered and modelled, the more accurate the carbon emission figure will be. However, the more resources are modelled, the higher the costs for designing the Resource Net, mainly due to the information gathering process and possible maintenance and management costs of the Resource Net. Accordingly, these two factors have to be evaluated against each other. It is important to note that we are not in the game of accounting carbon emission, but rather seek to enable carbon-aware business process design decisions. To identify all relevant resources, we provide the following guide:

1. Select business process design.
2. Select each activity and identify all relevant resources by asking: (a) What tools are used that require fossil fuels or electricity? (b) What materials are consumed?
3. Keep the relevant resources and drop non relevant resources by checking whether: (a) the associated costs are relevant, (b) they can be associated and quantified, (c) it can be done in reasonable time and money.
4. Identify all sub-resources of the identified resources and repeat the previous step.

We now provide an **example** of how a Resource Net can be instantiated and used. The example shows how an electrical device (in this case a modern printing device) could be represented using a Resource Net. The laser printer prints in black and white and provides scanning functionality. First, we identify all usage modes of the printer, being "print" and "scan". The scale in which both usage modes are measured is piece of pages (short: "page"). For simplicity, we assume that there are no direct costs associated with both usage modes. Note, if we would want to consider the carbon emission that has been emitted for producing the printer and transporting it to its current place, we could have done so by dividing the number of emission associated with producing and transporting the printer by the average number of papers that are printed or scanned during the printer's lifetime. However, the printer requires other resources (sub-resources) like paper for printing and/or electricity for scanning. For simplicity we assume that the resource paper is only used for printing, resulting in the usage mode "print". Another usage mode could have been "burn", denoting that the paper is combusted for heat and light. The scale of the paper is also captured in pieces of pages (short: "pages"). Again, to keep the example simple, we do not include the costs of producing and delivering the paper. The resource "electricity" is more

abstract and could denote a specific electricity provider and its offerings, or an average electricity provider of the country in which the business is located. We identify the usage modes "energy mix", "solar energy", "atom energy", and "coal energy", but seek to focus on the energy mix. The scale of the usage mode "energy mix" is Watt hours (Wh) and the costs are 2g CO2-e per Wh (this information is generally provided by the electricity provider or by the government as a regional average). Note that we can also associate other cost types with a usage mode, for example monetary costs. This results in the following representation of resources (R) and their usage modes (UM):

- R: $\langle Paper, \{print\}, \{\varnothing\}, \{\varnothing\}\rangle$;
  UM: $\langle print, \langle page, 1\rangle, \{\varnothing\},\ 0\rangle$
- R: $\langle Electricity, \{energy\ mix\}, \{\varnothing\}, \{\varnothing\}\rangle$;
  UM: $\left\langle energy\ mix, \langle Wh, 1\rangle, \{\varnothing\}, 2gCO2 - e_{/Wh}\right\rangle$
- R: $\langle Printer, \{print, scan\}, \varnothing, \{b\&w\ print, scanning\}, \rangle$;
  UM: $\langle print, \langle page, 1\rangle, \{b\&w\ print\},\ 0\rangle$

Having specified all resources, their usage modes and the associated costs, we associate them by defining the edges:

- E: $\langle\langle Paper, Printer\rangle, \langle UM_{print}, UM_{print}, 1\rangle\rangle$
  To print *one* page, one piece of paper is required.
- E: $\langle\langle Electricity, Printer\rangle, \langle UM_{energymix}, UM_{print}, 3\rangle\rangle$
  To print *one* page 3Wh are required.
- E: $\langle\langle Electricity, Printer\rangle, \langle UM_{energymix}, UM_{scan}, 2\rangle\rangle$
  To print *one* page 2Wh are required.

Figure 6.9 (below) shows a Resource Net instantiation (solid line), correlated with Activity "A", of the example described above. The dotted lines show possible other resources and activities using/extending the Resource Net. Activity "A" in the example uses the printer to print 15 pages and states this with the following functional annotation "*use(Printer, print, 15)*" (see bottom right corner of Figure 6.9). The function returns the emission annotation "*90g CO2-e*", denoting that 90g CO2-e are emitted when performing the activity (assuming this is the only resource used by the activity and it does not have any further user annotated emission impact).

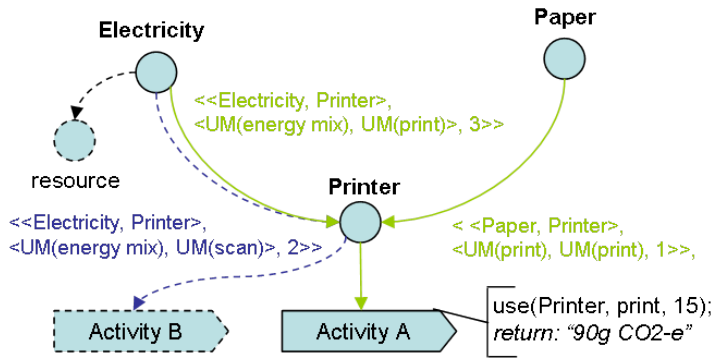

Figure 6.9: Example of a Resource Net.

6.2.1.4 Evaluation

We applied our framework on handcrafted examples similar to the provided examples, existing models published in the literature [61, 58, 117] as well as ca. 10 BPMN business process designs with up to 50 activities from the medical as well as financial domain. We were able to identify resources and associate these resources with their corresponding activities. Using [12] as our primary source for converting electricity consumption and fuel combustion into CO2-e, we correlated cost with the identified usage modes associated with a single activity. Finding the costs of resources that are possibly *used by several activities* at the same time (for example an information system, or the building in which the activities are executed) was somewhat more challenging, since the proportional break-down of costs cannot be easily identified. We handled this problem by utilizing the principles of (time-driven) activity based costing (ABC) [91], which is a well known and widely applied costing approach to trace indirect costs to cost objects. Specifying usage modes as cost objects made it possible for us to compute and trace indirect costs. However, alternate approaches are required. Finally, using the ProcessSEER machinery we were able to successfully accumulate the identified emission figures across the process designs.

6.2.1.5 Related Work

Capturing the resources, leveraged by a business process, through the usage-cost relationship, is an important factor for informing a business process design with its emission impact. As an overview, the current resource modelling literature does not give this relationship sufficient attention. The BPMN 1.2 specification does not consider resource models and their correlation with process designs. The *proposed* BPMN

2.0 specification [127] does, but provides no deeper insight into the aforementioned usage-cost relationship. Podorozhny et al. [130] introduce a general approach and key concepts in resource modeling and management, and apply it to activity and agent coordination. Of particular interest is the "requires relationship", which has some similarity to the proposed usage-cost relationship. However, the "requires relationship" only specifies that another resource is required, and further details such as the usage intensity are not captured or described. Zu Muehlen [118], [116] describes the role of resources in the context of workflow systems. A resource is considered to be a workflow participant that can *actively* contribute to the goals of an activity. Passive resources like material or information are explicitly not considered. On the other hand, Kwan and Balasubramanian [97] have a more abstract definition of a resource, also considering passive resources. They define a resource as an entity that is involved in the execution of an activity. This view is aligned with ours. Russell et al. [138] list the various ways in which resources are represented and utilized in workflow systems. A resource is described as an entity that is capable of doing work. The authors distinguish between human and non-human resources, where a non-human resource can be durable or consumable. To identify when a consumable resource is completely consumed, the concept of "rate of usage" for a consumable non-human resource is introduced to the resource meta-model.

6.2.1.6 Conclusion

A key challenge of today's organizations is to understand and optimize their environmental impact. We argued that BPM technology is the right starting point to address this problem, but needs to be extended to be able to estimate the emission of a business process at *design time*. We showed that this can be done by modelling resources and their usage-cost relationships as a Resource Net and by correlating it to activities of a business process design via textual annotations. The ability to estimate the emission performance of a business process is the basis for carbon-aware process re-design and optimization, which is the next logical step towards Green BPM.

6.2.2 A framework for comparing and accumulating heterogeneous measures

In Chapter 6.2.1 (summarizing our work in [79]), we have discussed a technique for accumulating emission values across a business process design. However, we did not

consider the case of multiple and potentially heterogeneous measures of qualitative and quantitative natures. In [78], we showed how this deficiency can be addressed via the c-semiring framework [18]. We now recapitulate this work.

6.2.2.1 Introduction

In practice, quality of service (QoS) measures are often not commonly agreed upon and are assessed on heterogeneous and multidimensional scales, which reduces their applicability (e.g. different machineries for each QoS measure have to be defined). Following the work in [57], we show how the algebraic "c-semiring" [18] structure is used to combine and compare both qualitative and quantitative QoS factors. In particular, we exemplify the usefulness of this approach when representing and reasoning with "green" performance measures.

The c-semiring structure generalizes many useful scales (real-valued, fuzzy-valued, probabilistic, qualitative and so on) and permits the integration of multiple heterogeneous scales into a single composite scale with the same abstract properties. Thus, while we might assess the carbon footprint of a process (or process step) on a real-valued scale, we might assess the impact of a process on flora/fauna on a qualitative scale of $\{high, medium, low\}$.

A c-semiring, which adds additional properties to the classic mathematical definition of a semiring, is defined as follows:

Definition 29. *(C-Semiring [18])*
A semiring is a 5-tuple $\langle A, \oplus, \otimes, 0, 1\rangle$ *such that* A *is a set of preference values, where* $0, 1 \in A$. *0 denotes the "worst" element and 1 denotes the "best" element in* A. $\oplus$ *is a cumulative, associative and idempotent comparison operator with 0 as identity element* ($a \oplus 0 = a$) *and 1 as absorbing element* ($a \oplus 1 = 1$), *closed over* A. $\otimes$ *is an associative and commutative combination operator with 0 as its absorbing element* ($a \otimes 0 = 0$) *and 1 as identity element* ($a \otimes 1 = a$), *closed over the set* A. $\otimes$ *distributes over* $\oplus$.

The "$\oplus$" comparing operator is used to define a partial order $\geq_S$ over the set of preference values A, enabling us to compare different elements of the semiring. On principle, $a \geq_S b$ if and only if $a \oplus b = a$, denoting that the $\oplus$ operator chooses a over b. The $\otimes$ operator is used to combine elements of the set A. We have $a \otimes b \geq_S a$ (1 is the maximum element of A), denoting that combining more elements of A leads to a "worse" result with respect to $\geq_S$. We require each activity to be annotated with its specific (local) QoS measures, represented by a vector $\langle m_1, m_2, ..., m_k\rangle$ where each

m_i is an element of A, of the c-semiring associated with the *i-th* QoS measure. We accumulate QoS values across a process design, using the "⊗" combination operator, to receive the cumulative QoS values for each distinct QoS measure of the process design (see Chapter 6.2.2.4 for more details).

6.2.2.2 Green QoS Measures

Our particular interest is in green QoS measures. In the following we provide an (incomplete) list of green scales and illustrate their instantiation in the c-semiring structure. Other QoS measures instantiated in the c-semiring structure can be found in [57].

1. *Water consumption*: $\langle R^+, min, +, \infty_+, 0\rangle$
2. *CO2-e*[11] *emission*: $\langle R^+, min, +, \infty_+, 0\rangle$
3. *Waste generation*: $\langle R^+, min, +, \infty_+, 0\rangle$
4. *Damage to fauna and flora*: $\langle \{low, ..., high\}, min, max, high, low\rangle$
5. *Air quality*: $\langle \{normal, ..., dangerous\}, min, max, dangerous, normal\rangle$
6. *Environmental performance*: $\langle \{AAA, ..., D\}, min, max, D, AAA\rangle$

While "Water consumption", "CO2-e emission" and "Waste generation" are generally quantifiable, other measures like "Damage to fauna and flora" are assessed in a qualitative scale since their "real impact" can only be assessed in long term studies. Therefore, these values are often determined by an educated guess of some expert. On the other hand, "Air Quality" and "Environmental Performance" are qualitative measures, representing a combination of measures. In the following, we show how different heterogeneous c-semiring scales can be combined.

6.2.2.3 Combining C-Semirings

Each QoS scale is of individual interest. However, there is often the need to combine different measures to assess the overall environmental performance of a process design, considering all or some subset of the green QoS scales. This results in a multidimensional QoS assessment. The resulting composite scale can be shown to have the same abstract properties (it is a c-semiring as well). More details and the corresponding proofs can be found in [18].

[11] Carbon dioxide equivalent (CO2-e) is an expression of other greenhouse gases in their carbon dioxide equivalent by their global warming potential (CO2 itself has a global warming potential of 1)

Definition 30. *(Composition of C-Semirings)[18]*
Given n semirings $S_i = \langle A_i, \oplus_i, \otimes_i, 0_i, 1_i \rangle$*, for* $i = 1, ..., n$*, let us define the structure* $Comp(S_1, ..., S_n) = \langle \langle A_1, ..., A_n \rangle, \oplus, \otimes, \langle 0_1, ..., 0_n \rangle, \langle 1_1, ..., 1_n \rangle \rangle$*. Given* $\langle a_1, ..., a_n \rangle$ *and* $\langle b_1, ..., b_n \rangle$ *such that* $a_i, b_i \in A_i$ *for* $i = 1, ..., n$*,* $\langle a_1, ..., a_n \rangle \oplus \langle b_1, ..., b_n \rangle = \langle a_1 \oplus_1 b_1, ..., a_n \oplus_n b_n \rangle$*, and* $\langle a_1, ..., a_n \rangle \otimes \langle b_1, ..., b_n \rangle = \langle a_1 \otimes_1 b_1, ..., a_n \otimes_n b_n \rangle$.

Accordingly, the order $\geq_S$ over $\langle A_1, ..., A_n \rangle$ is $\langle a_1, ..., a_n \rangle \geq_S \langle b_1, ..., b_n \rangle$ if and only if $\langle a_1 +_1 b_1, ..., a_n +_n b_n \rangle = \langle b_1, ..., b_n \rangle$. Since we have only defined the least upper bound ($\langle 0_1, ..., 0_n \rangle$) and greatest lower bound ($\langle 1_1, ..., 1_n \rangle$), not all elements of the composed c-semiring structure can be compared or combined. For example, consider the following composition of c-semiring instantiations for CO2-e and damage to fauna and flora. $\langle \langle R^+, \{l, m, h\} \rangle, \oplus, \otimes, \langle \infty_+, h \rangle, \langle 0, l \rangle \rangle$. Let us first compare the elements $\langle 5, m \rangle \oplus \langle 4, l \rangle = \langle 4, l \rangle$, hence $\langle 4, l \rangle \geq_S \langle 5, m \rangle$ $(a + b = a)$. However, $\langle 5, l \rangle \oplus \langle 4, h \rangle = \langle 4, l \rangle$ $(a + b = c)$ is not comparable, since there is no order for these two tuples. We can solve this by defining a new order $\geq'_S$ over each tuple in $\langle A_1, ..., A_n \rangle$ such that there does not exist a situation where $a \geq_S b$ and $a \neq b$ and $b \geq'_S a$ (the orders contradict each other).

On the other hand, we can map each tuple of the combined c-semiring into a set A of another c-semiring and define the order over this new structure. For example, scales for "fine particles in the air", "ground-level ozone", and "carbon monoxide" (besides others) can be composed and the resulting composition be mapped into the "Air Quality" c-semiring. Such a mapping is particularly appealing when a combination of different scales have to be communicated to stakeholders like the government, business partners or costumers. Similarly, the measure "Environmental performance" $\langle \{AAA, ..., D\}, min, max, D, AAA \rangle$, listed in Chapter 6.2.2.2, subsumes other environmental scales for ease of communication, where AAA denotes the "best" environmental performance and D the "worst" (similar scales are used to rate credits in the financial sector).

6.2.2.4 QoS Measure Accumulation

We now show how the QoS values (annotated at each activity) for distinct QoS measures can be accumulated across a process design to receive cumulative QoS values for each path through the process design.

Let S be a set of c-semirings, each denoting a distinct QoS measure for a given process design. We accumulate annotated QoS values, where each value corresponds to an element of the set A of each $s \in S$, between a pair of contiguous activities Ti and Tj (Ti pointing to Tj) as follows. The (cumulative) QoS values of a measure $s \in S$

at Tj consists of $a_i^{cu}(s) \otimes_s a_j^{lo}(s)$, where $a_i^{cu}(s)$ is a cumulative QoS value associated with Ti, and $a_j^{lo}(s)$ is a local QoS value of Tj. The operator '$\otimes_s$' is the corresponding combination operator for c-semiring $s \in S$. We deal with *XOR merges* (see Figure 6.7 label *G3*) in the following manner. If Ti and Tj are the only two activities immediately preceding an XOR-merge, and Tm is the activity immediately following it, we proceed by $a_i^{cu}(s) \otimes a_m^{lo}(s) = a_m^{cu}(s)$ and $a_j^{cu}(s) \otimes_s a_m^{lo}(s) = a_m^{cu}(s)'$. To deal with *AND merges* (see Figure 6.7 label *G4*) we have to add an additional *parallel combination operator* "$\circledast$" to the semiring structure. This operator allows us to specify how the QoS measures are propagated along a parallel branch and combined together at the merge gateway. For example, such an operator is needed when dealing with cycle time, where the "worst" time value of each branch in a parallel environment is picked. When we let Ti and Tj be the only two activities immediately preceding an AND-merge, and Tm be the activity immediately following it, we proceed by $(a_i^{cu}(s) \circledast_s a_j^{cu}(s)) \otimes_s a_m^{lo}(s) = a_m^{cu}(s)$. In the case of cycle time, the $\circledast$ operator (choose max value) is idempotent, therefore $\otimes$ distributes over $\circledast$.

6.2.2.5 Conclusion

We described how the abstract algebraic c-semiring structure can be used to represent, compare and accumulate heterogeneous and multidimensional measures across a business process design. This allows us to compare business processes and is the basis for environmental-aware business process improvement.

6.2.3 Semi-Automated Business Process Improvement

We now summarize our previous work on "Business Process Improvement in Abnoba" [77], in which we proposed a semi-automated process improvement machinery. Roughly, the machinery splits a semantic effect annotated business process design into fragments and tries to replace these fragments with more environmentally friendly fragments from a process fragment repository in a manner that ensures that the resulting business process can be provisioned by the existing resource context and has the same desired outcome as the original process.

6.2.3.1 Introduction

To meet the climate change challenge, we need to improve *operational efficiency* in a pervasive fashion. The notion of a *business process* is a particularly useful unit of

analysis in this context, as it allows us to model the as-is behaviour and to identify re-designs with a lower carbon-emission profile.

Various process improvement techniques have been described in the literature (see Reijers and Mansar [133] for a survey). While existing work primarily provides process re-design guidelines to the analyst, less attention is given to automated process re-design/improvement machineries. Existing machineries for automated process improvement (e.g. [123]) mainly focus on parallelizing activities. While parallelizing activities can have a positive impact on the process cycle time, the resulting process re-design is not necessarily superior with respect to other criteria like carbon dioxide emission. Furthermore, the number of applicable process re-designs is limited by the size of the search space constituting all possible process re-designs, which can be devised by parallelizing activities of the process design under consideration. By extending the size of the search space, we can increase our chances to find more preferable process re-designs. We extend the search space by leveraging a library of process snippets which constitute best practice process designs and fragments for a given domain. Essentially, we seek to replace process fragments of an as-is process design with other fragments (drawn from the library) in a manner that ensures that the *original goals of the process are still realized* (the same desired functional outcome is achieved), but the sustainability profile of the resulting process design is improved. Such a machinery requires the process designs (and fragments in the library) to be annotated with semantic effects and machinery for effect accumulation. For this, we build upon the ProcessSEER framework [61, 59, 75]. However, when introducing new process fragments (from the library) to the as-is process design, we must also ensure that the process re-design can still be executed in the organizational *resource context*. In this context, we describe a rich resource model, a machinery for correlating resources with process designs, and finally the resulting concept of (abstract and concrete) process provisioning. These results extend and improve upon earlier work on resource modeling [130, 97, 138, 118, 116]. In addition, it must be ensured that the process re-design does not violate any *compliance requirements*. Business process compliance for semantic annotated process designs can be found in [59, 61].

In Chapter 6.2.3.2), we first describe work done on resource modelling and introduce the concept of process provisioning before we propose a machinery for process improvement in Chapter 6.2.3.3. We conclude in Chapter 6.2.3.4.

6.2.3.2 Resource Modelling & Process Provisioning

Following [130], we extend our resource model proposed in [79] (summarized in Chapter 6.2.1). In addition to a "usage-cost relationship" (denoting that one resource is used by another with a certain intensity and associated cost), we allow relationships of type "IS-A" and "Whole-part", which enables us to talk about resources on different levels of abstraction. A *IS-A relation* is used to denote that entities can share a set of attributes, where the attributes of one entity are inherited by all specializations for that entity. The *Whole-part relation* is another abstraction mechanism where the linked resources constitute part of an aggregated resource. The *Use relation* (Podorozhny et al. uses the term "requires relation") denotes that a resource entity requires another entity to be used. Also, a resource can be of type "schedulable resource class", "unschedulable resource class" and "resource instance. A *resource instance* is a unique representation of an entity from the physical environment of discourse. It therefore denotes a specific resource in the organization. A *schedulable resource class* is a set of resources that are substitutable for each other and can be allocated to an activity. An *unschedulable resource class* is used to structure resources with similar properties that are not schedulable. We use this class to describe resources like electricity, which cannot directly be used by an activity, but are rather indirectly used via electrical devices (i.e. resources).

We refer to the *correlation of resource models and process designs* as the provision of the process design by the resource model. We consider *process provisioning* in two different ways. In *abstract process provisioning* we correlate an activity in the process design with a resource in the resource model. The resource in question might be arbitrarily abstract, e.g. a "printer" resource, or a more specific "inkjet printer" resource, or an even more specific "inkjet printer model x of manufacturer y" resource. By establishing an *abstract provisioning relation* between a task in a process design (say "print report") and any of these resources, we merely assert a provisioning relation (e.g. the "print report" activity shall use the "inkjet printer" resource, or an even more specific resource). In *concrete process provisioning*, we annotate the abstract provisioning relation with specific quality of service (QoS) requirements. For instance, we might establish a *concrete provisioning relation* between the "print report" activity and an "inkjet printer" resource which admits a configurable image resolution setting by stating the minimum image resolution requirements.

Technically, we correlate process designs with resource models by annotating each activity of a process design with an *expression*. The annotated expression references

one or more resource entities, where each reference is accompanied with a (potentially empty) set of QoS requirements. Each QoS requirement is described by a triple $\langle QoS, value, 1\rangle$, where QoS denotes the QoS measure of consideration, $value$ the required value, and 1 the most preferred value (as in the algebraic c-semiring structure [18], which we use in [78] to handle multidimensional qualitative and quantitative measures). The existence of a resource model permits us to explore a range of provisioning options. An *activity can be provisioned* by a set of resources R if (1) it requires R' resources and all elements of R' are also an element of R or of a specialization in R. (2) R' meets all QoS requirements stated in the provisioning relation. A resource meets a QoS requirement if the QoS value is equal or between the required value and the most preferred value (1). A *process design can be provisioned* if all its activities can be provisioned.

It can be observed that the provisioning relations serve as constraints, restricting the space of applicable process designs and applicable resource models. This is particularly important in the face of process re-design and optimization, where it must be ensured that the *process re-design can still be provisioned* by the resource context. Similarly, for any changes on the resource context (e.g. exchanging resources, or deploying a process design in a new resource environment), provisioning must be ensured. In the following we will focus on process design improvement, omitting resource optimization, which we seek to address in future work.

6.2.3.3 Business process improvement

Process improvement must involve a process re-design to obtain processes that achieve the same (functional) goals, while minimizing the environmental impact (and potentially other non-functional criteria as well). To achieve this, we need (1) the ability to annotate process designs with detailed specifications of functional effects (2) the ability to correlate organizational goals with process designs, and (3) the ability to search for process re-designs through a space of alternative process designs.

Preliminaries We now provide preliminaries on annotating and accumulating functional effects, correlating goal models with process models, and change patterns for design time changes on process designs.

A process design can be semantically enriched by specifying the outcome of each activity. This is done by annotating each activity with its immediate effects (post conditions), denoting the consequence of executing the respective activity in some state in the environment. Immediate effects are represented as logic statements in

conjunctive normal form. We pairwise accumulate effects to receive a cumulative effect. Pairwise accumulation is done by set union of the effects of two sequentially linked activities, such that the combined set of effects is consistent (we can view sentences in conjunctive normal form as sets of clauses, without loss of generality). If the resulting set is not consistent, the cumulative effects consist of the effects of the second activity and as much effects of the first activity as can be consistently included (effects are overwritten/undone by the second activity). Furthermore, all cumulative effects must be consistent with a background knowledge base (KB) containing rules (e.g. for compliance). Note that this KB can also be used to state pre-conditions (conditions that must hold before the activity can be executed) in a centralized manner. Effect accumulation in parallel environments is done by pairwise accumulating the cumulative effects of each branch with the effects of the activity directly following the AND-join and combining these sets via set union. A detailed description of this machinery can be found in [61, 59, 75]. The functional outcomes of a process are its cumulative effects at the end-event. A process design can have multiple cumulative effect outcomes one for each distinct execution path.

A goal is an objective the organization seeks to achieve or maintain. It can be formulated at different levels of abstraction ranging from abstract high-level goals to more refined operation goals. [152] A rich body of knowledge on goal modeling can be found in the goal-oriented requirements engineering literature. We list, besides others, [167, 152]. Correlating process models with goal models is crucial to ensure that the process environment, being under change, maintains to satisfy the organizational objectives. Koliadis and Ghose [93] propose the GoalBPM methodology for relating business process models (in BPMN) to functional goals (in KAOS [152]) via satisfiability links. These links are drawn between the process model and goals of the goal model, denoting the goals satisfied by the process design. A goal is represented as a logic statement and is, in its simplest form, a single literal α, or conjunction of literals $\{\alpha \wedge \beta\}$, for example "package send" and "bill send". A process design *satisfies* the organizational objectives if it satisfies *all* correlated goals. A correlated goal is satisfied by a process design if all its cumulative effects at the end-event entail the goal. For example, the cumulative effect $\{\alpha \wedge \beta \wedge \ \gamma\}$ entails (denoted by $\vDash$) the goal $\{\alpha \wedge \beta\}$). The implication of requiring all cumulative effects to entail a goal is that each path through the process design must lead to goal satisfaction. It is obvious that a path that does not satisfy the objectives of a process is superfluous. Note that a correlated goal does not have to denote "positive" outcomes only. For example,

a process "credit card application" might have the goal "application processed" and the background KB tells us that either the cumulative effect "credit card issued" or 'credit card declined" satisfies the goal.

Weber et al. [157] introduce high-level change patterns and change support features to assess existing process aware information systems in their ability to deal with process change. The high-level change patterns are divided into patterns supporting structural process adaption (adaption patterns) and patterns for built-in flexibility at predefined regions (e.g. to add information during run-time). We focus on the adaption patterns, since our focus is on design time optimization. Equally to Weber et al., we use the term *process fragment* to denote atomic activities, hammocks (sub-process graphs with a single entry and exit point), and complex activities (sub-process). The change patterns *delete process fragment* from process design and *replace process fragment* by another fragment, are of interest to us. Applying these change patterns on a process design we are able to discover potential process re-designs. Due to the nature of process fragment (single entry and exit point), the well-formedness of the process re-design is ensured.

A Machinery for Process Improvement We now introduce a machinery for discovering process re-designs using a library of process fragments. Figure 6.10 provides an overview of the procedure. The original process design (upper left corner) is (1) disassemble into its process fragments (lower left corner). (2) The change pattern "delete process fragment" is applied to delete potentially obsolete fragments. (3) Search for substitutable fragments in the library (lower right corner). (4) Replace substitutable fragments and check whether the resulting process re-design achieves the same desired functional outcome. (5) Order process re-designs according to their environmental profile. (6) Repeat previous steps until termination criteria is met.

In the first step, we *disassemble the as-is process design* into all its process fragments. An efficient technique for fragmenting a business process into single entry single exit fragments has been proposed in [155].

In the second step, we *identify and delete obsolete fragments* by checking, for each identified process fragment, whether it can be deleted, such that the resulting process design still satisfies the correlated goals (desired functionality). We do this by accumulating the semantic effects to get the cumulative effects. We then check whether the cumulative effects entail all correlated goals (as described before). If the cumulative effects entail the correlated goals, the process design is kept and the previous step repeated until no more fragment can be deleted.

In the third step, we seek to *find a substitutable process fragment* for each remaining

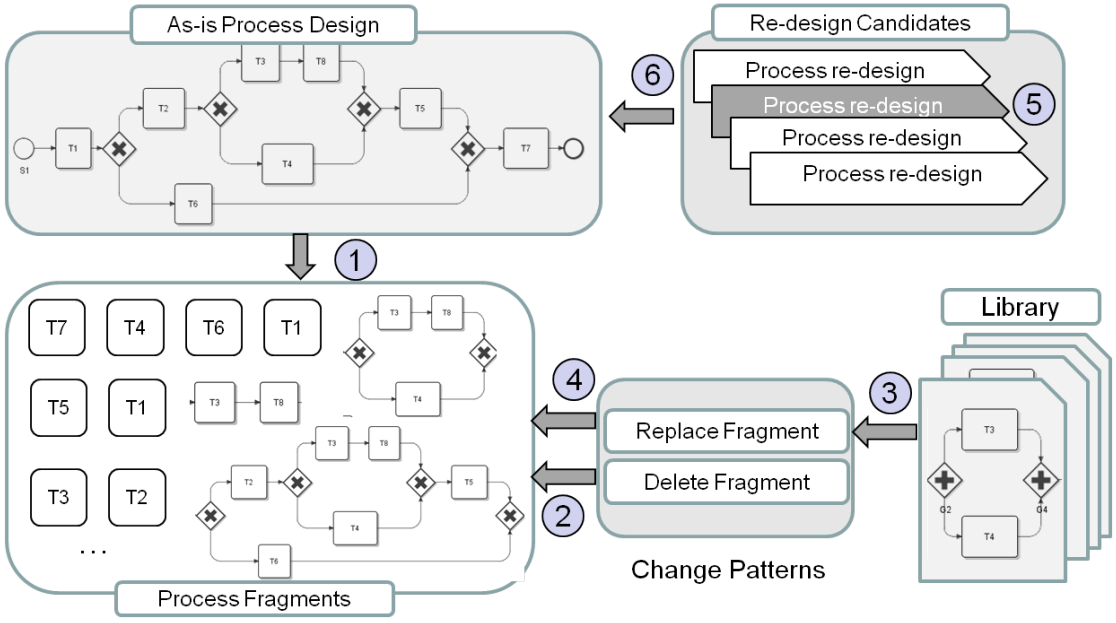

Figure 6.10: Overview of the business process improvement procedure.

process fragment from a library of process fragments. The library is constituted of a set of (semantic- and provisioning constraint annotated) process design snippets, denoting "best practice" solutions. The library is not restricted to process snippets, but could also include services (which can be seen as complex activities), derived from a service broker. Searching for replacements in the library can hence be utilized as an instrument for business functionality outsourcing. A process fragment p' is a potential substitute for another process fragment p if and only if every terminal effect scenario of p is entailed by some terminal effect scenario of p'. For example, consider a process fragment with two sequentially linked activities T_n and T_m with a cumulative effect of $\alpha \wedge \beta$ at T_m. This process fragment is substitutable by a process fragment in the library with one activity T_n and the effect $\alpha \wedge \gamma \wedge \beta$, since $\{\alpha \wedge \gamma \wedge \beta\} \vDash \{\alpha \wedge \beta\}$.

In the fourth step, we *replace each process fragment of the as-is design with a substitute in the library* (if one exists). Each such replacement denotes a potential process re-design, such that the number of identified substitutable fragments denotes the number of potential process re-designs. Due to the nature of process fragments, the well-formedness of each such process re-design can be guaranteed. However, it is not guaranteed that the potential process re-design candidate has the intended functional (cumulative effect) outcomes or whether it violates any compliance requirements (details omitted for brevity). This is due to the fact that a fragment from the library can unintentionally add additional effects to the process design (e.g. γ in the example

given above), which might conflict with effects of other activities, resulting in unintended cumulative effects and finally in non-satisfaction of correlated goals. A process fragment p' is defined to be a *viable process re-design* for another process fragment p of a process design P if and only if: (1) For every cumulative effect outcome s' in process design P' is obtained by replacing p with p' in P, $s' \vDash G_p$, where G_p is the goal of process P. (2) P' does not violate any compliance requirements. (3) P' can be provisioned. We use the ProcessSEER machinery to get cumulative effects. Finally, all *potential* process re-designs that satisfy the correlated goals, can be provisioned, and do not violate any compliance requirements are added to a *set of process re-designs*. Please note that the effect accumulation procedure for parallel environments assumes that the cumulative effects of each branch are consistent - there exists no execution order of activities in a parallel environment that leads to inconsistency. As pointed out by [75], such a scenario only occurs if the process is designed erroneously. Modeling activities in parallel, we implicitly accept that the execution order between activities on different branches does not have an impact on the functional result of the process. However, in our case we cannot guarantee that the assumption still holds. We therefore have to determine all interleavings of a parallel environment to identify execution orders leading to undesired cumulative effects. This can be a computationally expensive task for large process models. However, we can reduce the computational costs for most cases by devising a pre-checking procedure that catches *potential conflicts*. This is done as follows: Let us devise a set Ω for each branch $n, ..., m$ in a parallel environment, where Ω_n is the set of all effect elements of a branch n. A *potential conflict* is identified if $\Omega_n \cup ... \cup \Omega_m \cup KB \vDash \bot$. We only have to determine the cumulative effect of all possible interleaving of the parallel environment if there exists a potential conflict. Although we cannot always exclude this scenario, we note that we are in the game of design time improvement, where time constraints are less strict.

In the fifth step, we *order the process re-designs according to their sustainability profile*, such that the most preferred process design can be identified. In [78] we leverage the algebraic c-semiring structure [18] to compare "green" QoS values of both *qualitative and quantitative natures*. Essentially, a comparison operator "$\oplus$" is defined such that $a \oplus b = a$ if a is more preferred as b ($a \geq b$, where $\geq$ is a partial order over all values of a measure). The process is straight forward for a *single measure* and a process re-design with a *single execution* path. For process designs with *multiple execution paths* we end up with multiple values for a single measure, one for each path. This is handled as follows. (1) For a QoS measure of *quantitative nature* we use

the average value over all paths. This approach essentially places equal weighting on all paths. However, this approach might be erroneous due to the fact that some rarely executed scenarios might skew the mean value. To avoid this issue, we can consider the probability (if available) of taking a path through the process design during execution. Such figures can be derived from process logs or user experience. (2) For a QoS measure of *qualitative nature* we either choose the "best" or "worst" value as the corresponding value. In cases where we want to compare process re-designs with respect to *multiple QoS measures*, we compose each QoS measure (denoted by a single c-semiring), where the composite structure is a c-semiring as well (proofed in [18]). Hence we can use the "$\oplus$" comparison operator of the composite structure to compare the values. For a more detailed discussion see [18] and [78].

In the sixth step, the procedure is *repeated for all identified process re-designs* until a termination criteria is met (e.g. number of iterations, duration, no improvement after x amounts of iterations in a row). Please note that the described procedure is not complete in the sense that we cannot guarantee that the best possible process re-design has been identified. Nonetheless, we believe that the procedure can already provide significant improvements after a few iterations. Furthermore, the procedure can be interrupted at any time providing the user with the current order over the already identified process re-design suggestion.

6.2.3.4 Conclusion

We elaborated a machinery for process improvement and the concept of process provisioning. The process improvement machinery makes use of a library of (best practice) process snippets to discover potential process re-designs, while ensuring that functional, compliance and process provisioning requirements are met. This machinery will allow us to provide tool support for environmentally aware semi-automatic process improvement. The described machinery behaves as *semi*-automatic because it does not consider interconnections (e.g. message flows) across and between pools and hence requires the analyst to correctly place these links. Nevertheless, we believe that the workload for the analyst can be decreased, and that further (potentially overlooked) process re-design alternatives can be revealed.

6.2.4 Discussion

Given our work in Green Business Process Management, important aspects for addressing the non-functional aspects of the requirements problem are summarized as follows.

(1) We showed that the non-functional performance of a business process (we focused on CO2 emissions) can be assessed *at design time* by modelling resources and the usage-cost relationship amongst them and the business process. This allows us to estimate the non-functional performance characteristic of a process design that has not yet been executed, e.g. a business process *re-design*.

(2) We showed that providing immediate effect annotations to activities of a business process (describing the immediate outcome of performing the activity) can be used to compute the cumulative effect scenarios of the process (the expected cumulative outcomes, if the process were to be executed) - again at *design time*. This allows us to (1) identify from a pool (e.g. repository) of business process designs the ones that realize a given goal and (2) ensure that modifications to a given process design result in a re-design that remains to realize a given goal.

(3) We showed how this can be used to improve business processes in a semi-automated manner. A process design is improved by searching a repository of process snippets to replace with snippets of the process design in a manner that improves on the non-functional performance of the process, but ensures that it remains able to realize the correlated goal (in this case we treat the realization relation between the business process and the goal as normative).

At the beginning of this chapter, we stated the non-functional part of the requirements problem as follows.

Given domain knowledge and assumptions $\boldsymbol{KB}$*, compulsory goals* $\boldsymbol{G}_C$*, optional goals* $\boldsymbol{G}_O$*, compulsory soft-goals* $\boldsymbol{Q}_C$*, optional soft-goals* $\boldsymbol{Q}_O$ *and let* $\boldsymbol{G}_O^*$ *be the most preferred maximal consistent subset of* $\boldsymbol{G}_O$ *and* $\boldsymbol{G}_O^*$ *be the most preferred maximal consistent subset of* $\boldsymbol{Q}_O^*$*, the specification* $\boldsymbol{S}$ *is a solution to the non-functional part of the requirements problem, if* $\boldsymbol{KB}, \boldsymbol{S} \vDash \boldsymbol{G}_C, \boldsymbol{G}_O^*$ *and* $\boldsymbol{S}$ *satisfices* $\boldsymbol{Q}_C, \boldsymbol{Q}_O'^*$.

Given a solution to the functional part of the requirements problem, the aim is to find, from amongst these solutions, the one that (1) satisfices all mandatory- and as

many optional *soft*-goals as possible and (2) is the one which is most preferred by the stakeholders. In the following, we elaborate on and address each of the two conditions in more detail.

Recall, a soft-goal, as opposed to a goal, does not have a clear cut satisfaction criteria, i.e. it does not permit a boolean evaluation. For example, the statements "*minimize CO2 emission*, "*minimize process cycle time*", "*maximize customer satisfaction*", or "*maximize profit*" do not have clear cut satisfaction criteria, since they may always be further minimized or maximized. In operations research these statements are referred to as "objective functions" (e.g. in [150]). Soft-goals can describe both functional- and non-functional aspects of a system, but most often are used to describe the latter (the examples given above all describe non-functional aspects). The following discussion is restricted to soft-goals defined as objective functions on non-functional aspects of a system, i.e. we treat soft-goals as statements that require a non-functional system property to be either minimized or maximized.[12] The conceptualization of soft-goals as objective functions is not new and has also been discussed in Letier and Lamsweerde [103] as well as Jureta et al. [88].

One way to handle soft-goals is to "convert" them to goals by establishing a threshold, e.g. "CO2 emissions are reduced by 20% in the year 2015." has a clear cut satisfaction criteria - either emissions are reduced by 20% in the year 2015, or they are not. Alternatively, a goal is considered as *satisficed* (as opposed to satisfied) if it has been sufficiently addressed, e.g. if there exist means that positively contribute to the realization of the goal, as suggested by Mylopoulos et al. [121].

We follow the intuition given in Mylopoulos et al. [121] in many ways. We say that a description of system behaviour S contributes more positively than a description S' to a soft-goal g, if S minimizes/maximizes g more than S' does. For example, let the system behaviour be described by a set of business process designs. The business processes in set S contribute more positively than the processes in set S' to a soft-goal "minimize CO2 emission", if the expected cumulative CO2 emission of processes in S is lower then the expected cumulative CO2 emission of processes in S'. Our previous work on Green Business Process Management can be used to establish such an ordering over a given set of business process design collections by allowing us to get an estimate on the non-functional performance values (e.g. CO2 emissions) of an activity and/or sub-process and aggregate these values for the process/parent-process level.

[12] This is also the reason why we consider the fulfilment of soft-goals as part of the non-functional aspect of the requirements problem.

We say that a description of system behaviour S satisfices a soft-goal g if it is the description that most positively contributes to g. The previous sentence can be interpreted in different ways. We may interpret it as the search for the system description that is the "optimal" of *all possible system descriptions*. Unfortunately, this is not realistic in practice, since the space of all possible alternatives is unknown, potentially infinite, and at least very large. Alternatively, we can interpret the above sentence as the search for a set of optimal business process designs within certain boundaries. For example, in the process improvement procedure, summarized in Chapter 6.2.3, the process fragment library acts as a bound, i.e. there is only a finite number of business process designs that can be "derived" from the process fragment library. It is also worth pointing out that optimality can be defined in a modular manner with respect to a particular process improvement procedure. For example, given a process improvement procedure p (like the one summarized in Chapter 6.2.3) then a business process could be said to be p-optimal, if it cannot be further improved using the procedure p. Consequently, a soft-goal could be said to be p-satisficed by a business process design, if the process design is p-optimal. In a setting with more than one process design, i.e. in a "process eco-system" [95], this may not be sufficient. Optimizing each process design individually (i.e. locally) may not result in an optimal process eco-system (i.e. "global" optima), since (amongst others) processes are "interrelated" [96] and "interoperating" [95], such that one optimal process configuration may not permit an optimal configuration of another business process. [96] A soft-goal could hence said to be satisfied, if the process eco-system cannot be further improved with respect to the soft-goal. To automate this analysis the results presented by Kurniawan and his co-authors [96, 96] could be leveraged.

So far, we have only considered the case where a single soft-goal is to be satisficed. The case of multiple soft-goals poses additional challenges, since soft-goals may be conflicting. Two soft-goals (i.e. objective functions) are in conflict if, by improving one, we would have to eventually give up on improving the other. For example, we may seek to satisfice the soft-goal "*minimize CO2 emission*" and the soft-goal "*increase profit*", but there may not exist a description of system behaviour that can satisfice both, since a system that minimizes CO2 emission may be more expensive to maintain and therefore would decrease profits, while a system that is more affordable to maintain may increase CO2 emissions. In other words, there may not exist a description of system behaviour that can satisfice all soft-goals. One way to handle this is to restrict the analysis to a consistent subset of soft-goals. More precisely, we may only want to

consider a maximal consistent set of soft-goals, which contain all necessary soft-goals and as many optional soft-goals as is possible. Alternatively, instead of requiring that all soft-goals are to be satisficed, a different solution concept could be considered. A commonly used solution concept for a non-trivial multi-objective optimization problem is *pareto optimality*, which is due to Vilfredo Pareto. Roughly, given a set of conflicting objective functions (i.e. soft-goals), a solution (i.e. a description of system behaviour) is pareto optimal if none of the objective functions (i.e. soft-goals) can be further improved by another solution (i.e. description of system behaviour) without worsening one or more objective functions (i.e. soft-goals). The set of all pareto optimal solutions is referred to as pareto frontier (or pareto set) [20]. The pareto frontier is useful for further decision making since it excludes sub-optimal system descriptions and allows the decision makers to choose from the set of (pareto) optimal system descriptions.

The second part of the non-functional aspect of the requirements problem requires that we commit to a specification that is most preferred. More precisely, we are looking for a valid and completely satisfiable solution graph for which the set of desired goals include all necessary goals as well as the most preferred optional goals and for which the correlated description of system behaviour satisfices the most preferred set of soft-goals.

It is worth pointing out that a solution (graph) that satisfies the preferred set of goals and, at the same time, satisfices the preferred set of soft-goals may not exist. In such a case, a lexicographical ordering over the preference relations can be established. In the following, we consider the case where the stakeholders wish to consider solutions that satisfy the preferred set of goals (and mandatory goals) and then, amongst these, find the ones that satisfice (i.e. are optimal with respect to) the preferred set of soft-goals. There can be more than one solution graph that conforms with the above conditions, and by restricting the set of solution graphs to these it is ensured that returning any element from the set is an answer to the requirements problem. Preferably, we may wish to consider a set of solution graphs that also includes incomplete and non-preferred answers to the requirements problem in anticipation of future situations where they can be completed and become preferable. In the latter case, we need to ensure that a query for a solution to the requirement problem returns the right solution graph(s) from the set of solution graphs. Note, the set of solution graphs in many cases does not (and is not supposed to) contain all valid solution graphs (that are constructable from the goal library). This has two important consequences. (1) Omitting a solution is already an expression of preferential attitude and

therefore it must be consistent with the preference defined over goals and soft-goals. (2) A change in stakeholder preference may result in the most preferred solution not being part of the set of solution graphs. In such a situation, we would be required to re-compute solution graphs (in the worst case all solution graphs) from the goal library via an application of the completion operator. The computational advantages of explicitly maintaining a subset of all solutions must therefore be balanced with the cost of recomputing solutions.

In the following, we critically discuss and recapitulate some of the assumptions we have made, as well as other challenges that we have not addressed at this point. First, we treat soft-goals as objective functions, i.e. statements to minimize or maximize a non-functional property of a system specification. This representation of a soft-goal may not be appropriate for soft-goals of functional character. Typically this is not a problem, since (1) soft-goals usually refer to non-functional aspects and (2) a functional soft-goal is often a sign that a goal has not yet been refined enough and hence should be handled by further refining it (i.e. by turning it into a goal). Second, we assumed a lexicographical ordering that gives precedence to the satisfaction of preferred goals over satisficement of soft-goals. Jureta et al. [89] do not give one particular ordering precedence over the other. However, the applied ordering has computational advantages in the context of our proposed framework, since it reduces the number of business process designs that need to be considered by the process improvement procedure[13]. In other words, we only have to consider business process designs that realize goals of solution graphs that satisfy the set of preferred goals.

6.3 Related Work

The maintenance approach presented in Zwhogi et al. [170], Nuseibeh and his co-authors [126, 82] or Garces and his co-authors [52, 31] address the evolution of a requirements specification (i.e. goals are not considered) and hence can be argued to directly maintain an answer or multiple answers (in the case of Zwhogi et al. [170]) to requirements problem. It is worth highlighting though that these approach does not take into account non-functional aspects. In the REFORM framework [64, 63], Ghose distinguishes between goals and requirements of functional and non-functional character. Furthermore, the framework supports reasoning with descriptions of system

[13] In such a case, a soft-goal is defined to be satisficed if a given set of business processes cannot be further improved by a process improvement procedure (e.g. the one summarized in paragraph 6.2.3)

behaviour via their critical states (referred to as system trajectories). However, the primary focus is on whether the system states are consistent with a maximal r-consistent set of goals/requirements, rather than identifying descriptions of system behaviour that realizes a particular goal. Similar to the work by Ghose, the GSA framework, proposed by McNish and her co-authors [113, 38] supports reasoning system states (referred to as scenarios). However, as opposed to the work by Ghose, their approach does not only ascertain that all states are consistent with goals in the goal model, but also analyses whether all goals are satisfied (i.e. realized) by at least one state. The latter is similar to the approach we took in [80], where we correlate semantic service descriptions via their expected post-conditions to goals in the goal model and require all goals of a solution graph to be satisfied by at least of service. However, the GSA approach does not take into account stakeholder preference as well as fulfilment of soft-goals.

In Ernst et al. [43] (building on the previous work of Sebastiani et al. [143]), the authors use local search to find solutions to the requirements problem. More precisely, "TABU search" [67] is used to search an AND/OR (goal) graph for a specification that satisfies all mandatory- and the most preferred set of optional goals. As local search techniques do not guarantee optimality, the techniques proposed in Ernst et al. [43] may return incomplete and/or sub-optimal solutions to the requirements problem.

Furthermore, the work by Ernst et al. [43] did not address the requirements problem in a dynamic setting, e.g. where goals and/or preferences may change. The latter was explicitly taken into account into account in subsequent work by Ernst et al. [42], which we have discussed in Chapter 3.1. Recall that in their work, a goal model is represented in a requirements knowledge base REKB, where a change request (e.g. to add or remove a goal or to update domain assumptions) is reflected via basic "tell" and "untell" operations. The functional part of the requirements problem is addressed in a dynamic context via the query operator "GET_MIN_ CHANGE_ TASK_ENTAILING", which takes as input a set of goals and returns sets of tasks that realize the goals and minimize a distance function. In other words, the operator returns sets of tasks that are minimal modifications to the set of tasks that was selected prior to the change. It is important to emphasize that the operator is only successful if there exists a solution to the requirements problem. Therefore, the operator does not support reasoning with partial solutions. Our proposed framework permits reasoning with partial solutions (we referred to them as not completely satisfied solution graphs) by explicitly representing these in the set of solution graphs.

Furthermore, the work by Ernst et al. [42] assumes that the realization links (or more generally "traceability relationships" [148]) between tasks and goals are manually established. This has disadvantages in a maintenance setting, since changes in the goal model, as well as changes in the description of system behaviour (i.e. tasks, services, processes) may render these links as outdated. This requires the human engineering (in a laborious exercise) to continuously update the realization links, or (worse) may result in erroneous links that are overlooked. An advantage our our language agnostic framework is that goals can be specified in arbitrary detail (i.e. the goal semantics can be made precise), as opposed to the work by Ernst et al. [42] where goals are required to be atoms of a propositional language. In particular, we have showed how this allows to reason with post-conditions of semantic service description as part of the goal model, as well as to apply automated business process improvement machinery that takes into account a business process' execution semantics. Furthermore, this also the basis for applying techniques that support the incremental construction of software specifications from goals (e.g. as described in Letier and Lamsweerde [102]) or the construction of business processes from (e.g. as described in Ghose et al. [62]).

Overall, we believe that the ability to reason with partial solutions as well as the ability to derive realization links between (semantic annotated) business processes/services and goals in the goal model, further eases the exploration of solutions to the requirements problem in the context of our framework.

6.4 Summary

In this chapter, we have discussed how the exploration of answers to the requirements problem can be supported in the context of our goal model maintenance framework. In particular, we have shown how the body of work that was presented in earlier publications [80, 79, 78, 77] can support this exercise.

In the first part of this chapter (Chapter 6.1), we addressed the functional part of the requirements problem. In particular, we showed (in Chapter 6.1.1) how services described in a semantic service description standard and part of an organisation's service catalogue, can be represented via their post-condition as part of solution graphs. This allows us to assess the impact of changes at the goal level to the underlying service catalogue by highlighting unfulfilled goals (i.e. missing services) and superfluous services (i.e. services that do not fulfil a goal). An answer to the functional part of

the requirements problem corresponds to a solution graph that is valid and completely satisfiable (i.e. all goals are fulfilled by a service description).

In the second part of this chapter (Chapter 6.2), the non-functional part of the requirements problem was addressed. We first showed (in Chapter 6.2.1 and Chapter 6.2.2) how the non-functional performance of a design artifact (like a business process or service) can be estimated during design time by correlating it with resource models. This was followed (in Chapter 6.2.3) by a description of a process improvement machinery, which leverages a library of process fragments to make process modifications that improve the non-functional performance of the process, but also ensure that the process re-design remains to fulfil the goal it is correlated to. We then discussed (in Chapter 6.2.4) how this allows us to reason about the satisficability of soft-goals. Roughly, a (non-functional) soft-goal is considered satisficed by business process/service if the process design cannot be further modified in a manner that improves its non-functional performance.

Chapter 7

Implementation

In this chapter, we present the key aspects of an implementation of the "not entail assertion" operator. We begin by motivating a practical instantiation of the operator and show how finding a solution to the operator can be decomposed into independent sub-problems. After a brief summary of SAT and partial Max-SAT preliminaries, we show how each of these sub-problem can be cast as a state space search problem and efficiently solved using A* search.

7.1 Instantiation of the "not entail assertion" Operator Class

An implementation of the "not entail assertion" operator (class) requires us to commit to a particular language $\mathcal{L}$, a particular definition of minimal change (i.e. we have to instantiate the proximity relation $<_G$), as well as an instantiation of the selection function s. Our implementation is with respect to the following instantiations.

1. The language $\mathcal{L}$ is a classical *propositional language*.
2. The proximity relation $<_G$ is instantiated by "*Goal Graph Proximity*" (see Chapter 5.1).
3. The selection function s selects the first available solution. In other words, the first set of solution graphs that satisfies the properties of the "not entail assertion" operator is returned.

A classical propositional language was chosen because it is a good balance between expressiveness and computability, and therefore would be a realistic choice in a practical scenario. Computational advantages of a propositional language include *decidable procedures* for satisfiability (SAT) checking [139]. In addition, over the past decades *efficient* SAT solvers have emerged. These allow us to handle problems with tens of thousands of variables and constraints[1]. It is also worth mentioning that theories in more expressive logics like many sorted first order logic (as used in the running example) can be "propositionalized" [85] (i.e. grounded to propositions) and hence the same techniques can be leveraged.

As argued in Chapter 5.1, we believe that the chosen instantiation of proximity is good intuition of minimal change. It also has computational advantages as it restricts the refinements of the modified graph to those of the original one, which has a positive impact on the search space, i.e. it decreases the number of valid solution graphs considered in the search.

The chosen instantiation of the selection function does not impose any *additional* stakeholder preference on the set to be returned by the operator. In the following we show that this has significant advantages, as it allows us to maintain each solution graph in $\mathbf{G}$ separately, i.e. we can decompose the problem into smaller independent sub-problems.

7.2 Decomposition into independent sub-problems

We start by defining a "not entail assertion" operator for an individual solution graph, i.e. a "local not entail assertion" operator. The "local not entail assertion" operator takes a valid solution graph as input, and returns a valid solution graph that does not entail α and minimally deviates (with respect to chosen instantiation of the proximity relation) from the original graph.

Definition 31. *("local not entail assertion" operator)*
Given a valid solution graph $\mathcal{G}$ and an assertion α, let $\mathcal{G} \ominus_{local} \alpha = \mathcal{G}'$ such that $\mathcal{G}'$ is α-valid and there does not exist an α-valid solution graph $\mathcal{G}''$, such that $\mathcal{G}'' <_{\mathcal{G}} \mathcal{G}'$.

An application of the "local not entail assertion" operator on each solution graph in $\mathbf{G}$ results in a set of solution graphs that satisfies all the properties of the "not entail assertion" operator. Recall, the operator $\mathbf{G} - \alpha$ returns sets of solution graphs

[1] see SAT 2011 competition http://www.cril.univ-artois.fr/SAT11/results/benchinfo.php?idev=58

where each set in $\mathbf{G} - \alpha$ satisfies the normative properties of the "not entail assertion" operator.

Theorem 2. *Given a set of valid solution graphs* $\mathbf{G}$*, an assertion* α *and the operator* $\mathbf{G} - \alpha$*, then* $\{\mathscr{G} \ominus_{local} \alpha | \mathscr{G} \in \mathbf{G}\} \in (\mathbf{G} - \alpha)$.

Proof: Follows trivially from the minimal change property of the "not entail assertion" operator.

Given Theorem 2, an instantiation of the "not entail assertion" operator class where the selection function does not encode any additional stakeholder preference (i.e. any set $\mathbf{G}$ which satisfies all the properties of the "not entail assertion" operator is accepted) permits us to maintain each solution graph in $\mathbf{G}$ individually and find a solution faster.

Since the "not entail assertion" operator can be implemented by multiple applications of the "local not entail assertion" operator, in the following, we focus on an implementation of the "local not entail assertion" (proximity being instantiated by "Goal Graph Proximity").

7.3 SAT and partial Max-SAT Preliminaries

An implementation of the "local not entail assertion" operator requires means to (1) determine the derivability of an assertion from a set of goals, as well as to (2) identify maximal consistent subsets of goals. Since $\mathcal{L}$ is a propositional language we can cast the former as a "Boolean Satisfiability Problem" (SAT-Problem) and the latter as "Partial Maximum Boolean Satisfiability Problem" (partial Max-SAT-Problem). This allows us to draw upon a rich body of knowledge and leverage off the shelf tool support (we use SAT4J[2]). In the following, we briefly describe SAT related concepts and terminology used throughout the remainder of this thesis.

Most SAT solvers require formulae to be represented in conjunctive normal form (CNF), i.e. as a set of clauses, where each clause is a disjunction of literals. In the following, we use the function $C(x)$ to denote the CNF representation of a formula $x \in \mathcal{L}$, i.e. $C(x)$ is a set of clauses that is satisfiable (or not satisfiable) whenever the formula x is satisfiable (or not satisfiable)[17]. It is noted that a propositional formula can be converted into CNF in linear time [144].

[2] http://www.sat4j.org

A clause is *satisfiable* if at least one of its literals evaluates TRUE by a truth assignment. A set of clauses C is satisfiable if all of its clauses are satisfiable. For a set C of (potentially unsatisfiable) clauses, a set $C' \subseteq C$ is maximal satisfiable if all clauses in C' are satisfiable and there does not exist another truth value assignment such that a strict super set of C' clauses is satisfiable. This is known as the maximum satisfiability (*Max-SAT*) problem. In a partial maximum satisfiability (*partial Max-SAT*) problem, a maximum set of soft-clauses is to be identified that can be satisfied by a truth assignment that *satisfies all hard clauses*. We use $\overline{C}$ to denote that all clauses in the set C are hard clauses.

A "block" [10] is a set of clauses which is considered satisfied if and only if all its clauses are satisfied. Like for individual clauses, blocks can be defined as "hard-blocks" and "soft-blocks". Correspondingly, a partial Max-SAT problem for blocks is a maximum number of soft-blocks that can be satisfied by a truth assignment that *satisfies all hard-blocks*. [10] We use $\overline{C}$ to denote that all clauses in the set C are hard clauses. An efficient technique for solving Max-SAT problems with blocks has been discussed in [10].

7.4 An A* search encoding

We now show how the search for a valid outcome of the "local not entail assertion" operator can be formulated as a *state space search problem* and efficiently solved using *A* search* [72]. Roughly, each *state* in the state space corresponds to a distinct valid solution graph for which α is non-derivable. A *successor function* is used to encode the available transitions between states. The aim is to search the state space, starting from the *start state* (i.e. a modification of the original graph which does not entail α), applying the successor function to transition to neighbouring states, until a *goal state* (i.e. valid outcome to the operator) is found. *A* search* uses context specific heuristics to prune parts of the search space, but also guarantees that a goal state will eventually be found (provided that the heuristic is admissible and consistent [72]).

The formulation of the "local not entail assertion" operator as a state space search problem has various advantages: (1) it allows the problem (of computing the operator outcome) to be viewed in a well understood framework; (2) it allows a dynamic search for either one, more, or all valid outcomes of the "local not entail assertion" operator; (3) it allows (to some extent) language specific techniques (i.e. the language in which the goals and the $\mathcal{KB}$ are formally expressed) to be abstracted, only requiring that

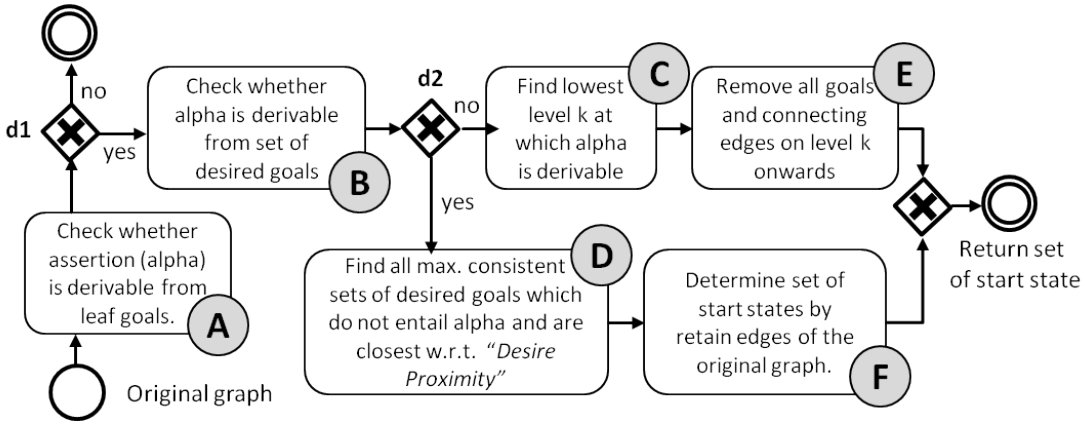

Figure 7.1: BPMN design of the pre-processing procedure.

given subroutines are implemented in a language dependent manner.

An A^* search encoding requires us to define a *start state*, *successor function*, *heuristic function*, and *goal test function*.

7.4.1 Start State

A* search (in the worst case) is known to face an exponential growth of visited states in the length of the solution. [139] The length of the solution is given by the minimum number of states that connect the start state to the goal state in the state space. Shortening the length of the solution decreases the number of visited states and thereby improves the performance of the search. Therefore, it is not uncommon to leverage problem specific techniques (in a pre-processing procedure) to compute a start state, which shortens the solution length, if doing so reduces the overall computational effort.

In our encoding of the search problem, the start state is *not* given by the original solution graph, but determined by a pre-processing procedure. The outcome of the pre-processing procedure is (a set of) start state(s) that correspond to a valid solution graph for which α is not derivable and which is closest to the original graph with respect to *Desire Proximity*. In other words, the actual A* search is reduced to finding a solution graph that is closest with respect to *Refinement Proximity*. In the following, we describe the pre-processing procedure (see Figure 7.1) in detail.

Figure 7.1 visualizes the pre-processing procedure via a (BPMN[3]) process diagram. The pre-processing procedure takes a valid solution graph $\mathscr{G}$ as input, and consists of seven steps (labelled "A", "B", ..., "F") and two decision points (labelled "d1" and

[3] `www.bpmn.org`

"d2").

Step A: In the first step, the procedure checks whether the assertion α is derivable from the set of leaf goals. Given the original solution graph $\mathscr{G}$, we can encode this as a SAT-Problem via the following set of clauses.

$$\phi_1 = C(\mathcal{KB}) \cup C(f_{leaf}(\mathscr{G})) \cup C(\neg\alpha).$$

If ϕ_1 is satisfiable, then α is not derivable, else α is derivable. From Corollary 1, it follows that any assertion not derivable at the leaf level is not derivable at any level in the goal graph. Therefore, if α is not derivable at the leaf level (decision point "d1"), the original solution graph is a valid outcome of the operator, i.e. we have found the solution.

Step B: In case α is derivable from the set of leaf goals, the procedure checks whether α is derivable from the *set of desired goals*. The SAT encoding is as follows.

$$\phi_2 = C(\mathcal{KB}) \cup C(f_{des}(\mathscr{G})) \cup C(\neg\alpha).$$

If ϕ_2 is satisfiable, then α is not derivable, else α is derivable. Note, if α is non-derivable from the set of desired goals, then retaining the set of desired goals of the original graph trivially guarantees that "Desire Proximity" is satisfied. However, if α is derivable from the set of desired goals, we first have to compute a maximal consistent set of desired goals that is closest to the set of desired goals of the original graph (i.e. following the lower path at decision point "d2" in the process). In step D, we show how this can be cast as a partial Max-SAT problem, for which reasonable efficient solvers have been developed over the past years.

Step C: In the case that α is not derivable from the set of desired goals, we identify the earliest level of the original solution graph at which α is derivable (following the upper path at decision point "d2"). Since we know that a k-entailed assertion is also entailed at higher levels (in terms of k), we can identify k using an efficient binary search (see Algorithm 1 below). The idea is as follows. Given $depth(\mathscr{G}) = i$[4], we check whether α is a $i/2$-consequence (i.e. entailed at level $i/2$), if true, the "earliest appearance" of α must be between level 0 and $i/2$, otherwise it must be between level $i/2$ and i. We repeat until we find the earliest level at which α is entailed.

Note, in Algorithm 1, *mid* may be the earliest level in which α is entailed, but this is not known until we have searched through all options. This has two consequences:

[4] Recall that $depth(\mathscr{G})$ returns the longest path between any two vertices in G.

Algorithm 1: Step **C**

input : $\mathscr{G}$ (a valid solution graph),
α (the assertion not to be entailed).

int $min = 0$.
int $max = depth(\mathscr{G})$.
while $min < max$ **do**
  int $mid = ((min + max)/2)$.
  if $\mathscr{G}_{mid}$ *entails* α **then**
    `//Recall that` $\mathscr{G}_k$ `denotes all goals of` $\mathscr{G}$ `at level` k.
    $max = mid$.
  else
    $min = (mid + 1)$.
return min.

(1) as opposed to most text book algorithms, we do not subtract 1 from mid, as mid may be the solution; (2) the number of required steps (i.e. number of consistency checks) in the worst- and best case is identical (i.e. $log_2(depth(\mathscr{G})) + 1$).

Step E: The start state is identified by removing all goals and connecting edges of the original goal graph from level k onwards (where α is k-entailed). More precisely, for each refinement/edge of the original goal graph for which the target goals are on level k or higher, all target goals are removed from the set of vertices and the edge itself is removed from the set of edges of the original goal graph.

Note, the resulting graph is guaranteed to be valid, does not entail α, is closest to the original one with respect to "*Desire Proximity*", and is guaranteed to be a sub-graph of any valid outcome of the "local not entail assertion" operator.

Step D: In case α is derivable from the set of desired goals (decision point "d2"), we (1) compute all maximal sets of desired goals that are consistent with $\neg\alpha$ (we referred to this as α-consistency) and (2) find amongst these the ones that share as many desired goals with the original solution graph as possible.

This first step can be encoded as partial a Max-SAT problem. Recall, a solution to a partial Max-SAT problem is a satisfiable set of clauses that contains all hard clauses and as many *soft-clauses* as possible. The above problem can be encoded as shown in ϕ_3, i.e. clauses in $C(KB)$ and $C(\neg\alpha)$ are captured as hard clauses, while clauses in $C(LIB^{des})$ are captured as soft-clauses.

$$\phi_3 = \overline{C(\mathcal{KB})} \cup \overline{C(\neg\alpha)} \cup C(LIB^{des}).$$

We compute solutions to ϕ_3 using the algorithm and encoding described in Liffiton and Sakallah [108]. This technique allows us to efficiently compute *all* solutions by incrementally searching a single search tree (as opposed to solving multiple optimization problems as is the case for many off-the-shelf Max-SAT solvers (e.g. SAT4J)).

In the case that all goals of the goal library were represented by a *single clause* (i.e. a singleton set), each solution to ϕ_3 includes a maximum consistent set of desired goals. In the case that goals are expressed by more than one clause, then the clauses of each goal are treated as a block. In other words, a solution to ϕ_3 contains a goal g, if and only if all clauses of g are part of the solution. We identify sets of goals by searching through the solutions that have been computed for ϕ_3 and check, for each goal, whether all its clauses are contained in the respective solution. Alternatively, we could leverage the techniques described in Argerlich and Manyà [10] to directly compute solutions for blocks. The approach proposed by Argerlich and Manyà [10] is also more efficient, since (roughly speaking) all clauses of a block can be ignored if one of its clauses cannot be satisfied. We are currently extending our implementation towards this direction.

Given sets of maximum (α-)consistent desired goals, we need to identify the ones which share as many desired goals as possible with the original solution graph. This is done by sorting the solutions according to the following weighting scheme. Each goal in a solution that is part of the original solution graph has a weighting of "1", otherwise "0". Summing up these values provides us with a cumulative weighting for each solution, according to which we efficiently sort all solutions in descending order (e.g. using "quicksort" [76]). The top and equally weighted elements of the ordered list are the outcome of this step.

It is worth highlighting that the list of maximum α-consistent sets of desired goals can be *reused* when applying the "local not entail assertion" operator to other elements of the set of solution graphs (only the ordering has to be re-determined).

Step F: All maximum α-consistent sets of desired goals returned by the previous step are captured by a new solution graph. However, each of these solution graphs is unstructured, i.e. its set of edges is empty. This can be resolved (without any additional checks) by retaining all edges/refinement relations of the original graph, for which the source and all target goals are an element of the respective solution graph. Each of these created solution graphs is a start state to the search. Note, the search can start with more than one element in the queue (i.e. the ordered list of states to be visited).

Observe that the outcome of the lower and upper path after decision point "d2" in the pre-processing procedure is a set of solution graphs (the upper path results in a singleton set), where each graph (1) is valid, (2) does not entail α, and (3) is closest to the original graph with respect to Desire Proximity. However, the solution graph may not be closest with respect to *Refinement Proximity*, since further refinements (of the original graph) may be added without causing an inconsistency or reintroducing the derivability of α. In other words, a valid outcome to the "local not entail assertion" operator can be found by adding refinements of the original graph (and their respective target goals) to the solution graph in each start state. Finding the solution graph which is closest with respect to *Refinement Proximity* is done in the A* search. We use $f_{start}(\mathscr{G})$ to denote the set of all start states that were computed in the pre-processing procedure for the original solution graph $\mathscr{G}$.

7.4.2 Successor Function

Given a state s, a successor function $f_{succ}(s)$ is used to generate as set of connected (successor) states. The successor function has to be devised such that a goal state (i.e. a valid outcome of the "local not entail assertion" operator) is guaranteed to be part of the state space. This can be done by identifying minimal sequences of modifications (i.e. adding and removing vertices and edges) of the solution graph in state s, such that the resulting graph is valid (and α remains non-derivable). Recall that (due to the pre-processing procedure) removing elements cannot bring the search any closer to a goal state. Furthermore, the only way elements can be added to a given solution graph is via a refinement of a goal already in the graph.[5] Therefore, each refinement (of the original graph) that can be added to the solution graph $\mathscr{G}'$ in a state s, such that the resulting solution graph is valid and the assertion α remains non-derivable, could be considered as a successor state. However, this results in a large set of successor states[6] and consequently a very bad branching factor.

Fortunately, we can *decrease the branching factor considerably.* Instead of considering the addition of a single refinement (from the original graph), we consider

[5] A goal cannot be added (i.e. as a root) because the definition of a valid solution graph requires all root goals to be desired goals, and the graph denoted by the current state already contains a maximal consistent set of desired goals (since it is valid) and therefore no more desired goal, i.e. no more root goal, can be consistently added.

[6] In the worst case, the number of successor states for a solution graph in a state s corresponds to the number of its leaf goals that can be further refined.

the addition of as many refinements of the next immediate level as possible. More precisely, given a solution graph $\mathscr{G}$ in some state s, we identify a maximal set of refinements (from the original graph) which can be directly added to *considered leaf goals* (we elaborate on this shortly) of $\mathscr{G}$ such that their target goals do not introduce any inconsistencies or make α derivable. A leaf goal g of the solution graph $\mathscr{G}'$ in state s is *not considered* if g was an element of a complementary set of goals that were added to the solution graph in predecessor state s' of s. Since the addition of any element in the complementary set would necessarily result in a non-valid solution graph, we do not need to consider their addition at any following state. The set of *considered leaf goals* will also be required to compute the heuristic value of state s.

For each refinement of the original solution graph that can be directly linked to a considered leaf goal of the solution graph in the current state s, we capture the clauses of its target goals as a block (the idea being that a refinement can only be added if all clauses of all of its targets goals together do not make α derivable). Let $\mathcal{E}_+$ denote this set of blocks. Further, let N denote the set of desired goals that were added as a result of (taking the lower branch in) the pre-processing procedure (in the case where the upper branch was taken, then the set N is empty), then we compute all maximal subsets $\mathcal{E}_+$ via a call to the partial Max-SAT solver with the following encoding.

$$\phi_5 = \overline{C(\mathcal{KB})} \cup \overline{C(\neg\alpha)} \cup \overline{C(N)} \cup C(\mathcal{E}_+).$$

We consider the goals in N, since these goals were not part of the original goal graph and may be (together with other goals) in conflict with goals in $\mathcal{E}_+$. We use the function $pMSAT(\phi_5)$ to return all maximum clause subsets of ϕ_5.

It is worth mentioning that, instead of calling a partial Max-SAT solver at each application of the successor function, we could also determine the maximal consistent subsets of *all* goals of the original goal graph via a *single* call to the solver. This would give us a bag of maximal consistent sets of goals, which we could use to determine which goals on a particular level are consistent. However, as we would have to include *all* goals of the original goal graph, the corresponding partial Max-SAT problem is "larger" (i.e. has more variables and constraints) than the partial Max-SAT problem we have to solve when applying the successor function at each state (where we only take into account the goals on a particular level). As the worst case time complexity of partial Max-SAT is known to be NP-hard [17], solving multiple "small" problems is often more efficient than solving a single "large" problem.

7.4.3 Heuristic Function

A heuristic function is used to guide the search to "promising"[139] areas in the search space. In our context, the heuristic function should guide the search towards states corresponding to solution graphs that are closest to the original goal graph with respect to *Weighted Refinements Inclusion*. A heuristic function that is *admissible* and *consistent* is guaranteed to find a solution (i.e. a solution graph that is closest to the original one) while pruning away parts of the search space that are guaranteed to not contain a solution.

As with standard A* search, the heuristic function is the sum of the *actual distance* of state s from the start state and the *estimated distance* of s to a solution (i.e. the goal state). To assess distance, we use a *weighting scheme* that involves assigning values to elements in a solution graph. This leads to a cumulative weight (value) for each graph considered during search, with the idea being that a graph with a higher weight is "closer" to the original graph.

To capture proximity defined by *Refinement Proximity*, we assign a value of 1 to refinements that refine goals on level $depth(\mathscr{G}) - 1$ and assign a value to each refinement that refines a goal at level j which is strictly higher than the sum over the value assignments to refinements that refine goals on level k (where $j < k$).

Given this weighting scheme, the *actual weight* of the solution graph $\mathscr{G}'$ in state s is simply given by the sum over the weight of all refinements in $\mathscr{G}'$. The *estimated weight* is an optimistic guess about the sum of the weight of refinements that can be further added to the solution graph in s. A solution to a relaxation of the original search problem is guaranteed to return an optimistic estimate. [139] We relax the search by dropping the condition that a solution graph must be consistent and not entail α. In other words, the estimated weight value for a solution graph at state s is given by the sum over the value assignments to refinements that can be added to leaf-goals that are relevant and to their consequent sub-goals.

The *heuristic function* (f_h) at a state s is given by the sum of the actual weight and estimated weight.

7.4.4 Goal Test Function

A state s is a *goal state* (i.e. $f_{goal}(s) = \text{true}$), if $f_h(s) = 0$. Observe that $f_h(s) = 0$ if no more refinements of the original goal graph can be added to the solution graph in the current state s. Since the heuristic function is admissible, the first state s for

which $f_h(s) = 0$ must be the optimal solution (i.e. the solution graph which minimally deviates from the original one).

7.4.5 A* Algorithm

Given the set of start states $f_{start}(\mathscr{G})$, the successor function f_{succ}, the heuristic function f_h, and the goal test function f_{goal}, the (best first search) Algorithm 2 (adopted from Russel and Norvig [139]) computes a valid outcome to the "local not entail assertion" operator. In Algorithm 2, the function $f_{que}(queue, states)$ inserts a set of *states* into the *queue* and orders all states in the *queue*, such that the one with the highest heuristic value (with respect to f_h) is in the front of the list. The subroutine $RemoveFront(queue)$ returns and removes the front element of the *queue*.

Algorithm 2: A* search

```
input : ⟨f_start(𝒢), f_succ, f_h, f_goal⟩
queue ← f_start(𝒢)
while queue ≠ ∅ do
    s ← RemoveFront(queue)
    if f_goal(s) = true then
        return s
    queue ← f_que(queue, f_succ(s))
return failure
```

7.5 Correctness

The implementation is correct if the A* search returns a solution graph as specified by the "local not entail assertion" operator. The A* search must return a correct outcome if we can show that it cannot return a solution graph that (1) is not α-valid[7] or (2) not a minimal modification of the original solution graph.

First Part (returned solution graph must be α-valid):
The solution graph $\mathscr{G}'$, returned by our implementation, is not α-valid if at least one of the following is true:

(a) one or more of its refinements are incorrect, or

[7] Recall, a solution graph that is α-valid, amongst others, is guaranteed to not entail α.

(b) its goals are α-inconsistent[8], or

(c) a desired goal from the goal library can be α-consistently added, i.e. the desired goals are not α-maximal, or

(d) one or more of its root goals are *not* desired goals (i.e. dependent goals), or

(e) one or more of its refinements are not immediate refinements.

We now show that none of the above conditions can be true. All refinements must be correct (ruling out (a)) and immediate (ruling out (e)), since the solution graph returned by the implementation only includes refinements of the original solution graph and the original solution graph is required to be valid. Goals of $\mathscr{G}'$ must be α-consistent (ruling out (b)), since (1) the goals of all solution graphs returned by the pre-processing procedure are α-consistent and (2) the successor function only adds refinement to a solution graph in state s, that retain the α-consistency of the solution graph. In other words, all states in the state space correspond to α-valid solution graphs. The set of desired goals must be (α-)maximal, i.e. no more desired goal from the library can be α-consistently added. Note, no solution graph element (i.e. goal or refinement) is ever removed from a solution graph as a result of applying the successor function. Therefore, the desired goals of a solution graph returned by an application of the "local not entail assertion" operator are guaranteed to be α-maximal if the desired goals of all solution graphs retuned by the *pre-processing procedure* are α-maximal. The pre-processing procedure has three execution paths $e_1 = \langle A \rangle$, $e_2 = \langle A, B, C, E \rangle$, $e_3 = \langle A, B, D, F \rangle$ (see Figure 7.1). In execution paths e_1 and e_2, all desired goals of the original solution graph can be retained and therefore guaranteed to be α-maximal. In execution path e_3, step "D" identifies sets of desired goals that are α-maximal and step "F" only returns solution graphs for which the desired goals correspond to one of the sets identified in step "D". To sum up, the desired goals of the solution graph, returned by the "local not entail assertion" operator, must be *alpha*-maximal (ruling out (c)). Last, all root goals of $\mathscr{G}'$ must be desired goals, since the root goals of the original solution graph are desired goals, and dependent goals are only added as refinements to existing goals in the solution graph (ruling out (d)).

Second Part (minimal change):
Let us assume that $\mathscr{G}'$ is not a minimal modification of the original solution graph $\mathscr{G}$, so another solution graph $\mathscr{G}''$ which is α-valid and is preferred with respect to "Goal

[8] Recall, a set of goals is α-consistent if all goals are conjointly consistent with $\neg\alpha$.

Graph Proximity"[9] must exist. However, this cannot be the case since (1) the solution graphs returned by the pre-processing procedure are closest to the original graph with respect to "Desire Proximity" and (2) the A* search returns (amongst these) the one that is also closest with respect to "Refinement Proximity".

The solution graphs returned by the pre-processing procedure are closest to the original graph with respect to "Desire Proximity", since execution paths e_1 and e_2 of the pre-processing procedure return solution graphs that retain all desired goals of the original solution graph and are therefore guaranteed to be closest with respect to "Desire Proximity". In execution path e_3, step "D" identifies the sets of desired goals that share as many desired goals with the original solution graph as is possible, and step "F" only returns solution graphs for which the desired goals correspond to one of the sets identified in step "D".

For the A* search to return a solution graph that is also closest with respect to "Refinement Proximity", the heuristic function must be admissible and consistent. The heuristic function $f_h(s)$ is admissible if the estimated weight for any state s is never underestimated (the intent is to maximize weight). Recall, the estimated weight is given by the sum over the weight assignment to refinements that can be further added to the solution graph in state s, regardless of whether this causes inconsistencies or makes α derivable. In other words, we relax the original search problem by dropping the two conditions. Since the weight of a solution to the relaxed problem can never be smaller than (i.e. never underestimate) the weight to the solution of the non-relaxed problem, the heuristic must be admissible. The heuristic function is consistent since the actual- and estimated weight is identical for any path taken through the state space to get to a state s.[10]

7.6 Summary

This chapter presented key aspects of an implementation of the "not entail assertion" operator.

In Chapter 7.1, we discussed and motivated (what we believe to be) a realistic instantiation of the "not entail assertion" operator. In this instantiation, goals are specified in a propositional language, proximity is instantiated by "Goal Graph Prox-

[9] Recall, that "Goal Graph Proximity" is defined as a lexicographical ordering over "Desired Proximity" and "Refinement Proximity"

[10] As a side note, the state space is a tree (i.e. we are doing tree-search) and therefore the function $f_h(s)$ would not be required to be consistent for the A* search to be optimal.

imity" and the selection function does not enforce any additional restrictions. We then showed (in Chapter 7.2) how the problem of finding a solution to the operator can be decomposed into independent sub-problems. More precisely, we showed how finding a minimally modified set of solution graphs can be treated as finding a minimal modification for each solution graph in the set. We then continued to show how finding a minimally modified solution graph (that does not entail a particular assertion) can be formulated as a state space search problem and efficiently solved using A* search. A discussion on the correctness of the presented implementation was given in Chapter 7.5.

Chapter 8

Experimental Evaluation

This chapter discusses and presents experimental results of a prototype implementation of the "not entail assertion" operator. The purpose of this experimental evaluation is to show that our approach can be used to provide algorithmic maintenance support. This is done by showing that an prototype implementation can scale up to goal models of a size that can be expected in a practical setting.

First, we motivate and discuss various performance indicators and drivers. In Chapter 8.2, we discuss experimental preliminaries, including the procedure used to randomly generate solution graphs. We also give a brief description of an uninformed version of the A* search (which will provide a baseline to compare the A* search against). The experimental results are presented and discussed in Chapter 8.3.

8.1 Indicators and Drivers

We can draw conclusions on the scalability of an implementation by observing a set of meaningful indicators with respect to a change in value for a set of relevant drivers. We propose the following *indicators* and *drivers*.

8.1.1 Indicators

An important indicator is execution time (denoted by I_{Time}), i.e. the time it takes to compute a valid outcome to the "local not entail assertion" operator. The execution time is largely influenced by calls to the SAT- as well as partial Max-SAT solver, which are known to be in the class of NP-complete [25] and NP-hard problems, respectively. To abstract from the performance of a particular solver as well as hardware config-

uration, we also observe the number of calls to the SAT- (denoted I_{nSAT}) as well as partial Max-SAT solver (denoted by I_{npMSAT}). Another useful indicator is the number of states that are visited during the A* search, which in the worst case is known to face an exponential growth of visited states in the length of the solution[1] [139]. Note, the number of visited states is indirectly measured by I_{npMSAT}, since the partial Max-SAT solver is called each time the successor function is applied (and once in situations where step "D" of the pre-processing procedure is executed), and the successor function is applied each time a state is visited (except that state is a goal state).

8.1.2 Drivers

In the following, we discuss and motivate a set of drivers that impact the listed indicators.

The most important driver is the *"size" of a solution graph*, which is given as an input to the "local not entail assertion" operator. The size of a solution graph is assessed by its number of levels (i.e. the graph's depth). (1) The number of levels impact the number of required calls to the SAT solver, since it directly impacts the size of the binary search in step "C" of the pre-processing procedure. (2) The number of levels impact the length of the solution in the A* search, which impacts the number of visited states and consequently the number of calls to the partial Max-SAT solver. (3) The number of calls to the SAT- and partial Max-SAT solver in turn impact execution time. We use D_{Levels} to denote this driver.

In situations where step "D"[2] of the pre-processing procedure is executed, the number of desired goals in the goal library drive the total number of clauses given to the partial Max-SAT solver. In other words, as the number of desired goals increases, the number of clauses increases. However, there is no strict correspondence, since each goal may be formulated by an arbitrary number of clauses. We take care of this by directly considering the number of clauses (denoted by $D_{nclauses}$). This also allows us to take into account domain knowledge bases with various numbers of clauses, in a single driver.

We say that there is a conflict if a set of goals in a solution graph is α-inconsistent. There can be multiple conflicts in a solution graph and more than one option to resolve

[1] The length of a solution is given by the minimum number of states that connect the start state to the goal state.

[2] Recall, in step "D" all maximal (α-)consistent subsets of the LIB^{des} are computed if and only if α is derivable from desired goals of the given solution graph.

a conflict. The more options there are to resolve a conflict the more states need to be visited during the A* search. This in turn impacts the number of calls to the partial Max-SAT solver. The number of options to resolve a conflict is hard to set, but can be approximated by a combination of the following two drivers. Let $S = \{g_l, g_m, ..., g_n\}$ be a minimum (with respect to set inclusion) conflicting set of goals. Since S is minimal conflicting, removing any element from it makes the set non-conflicting. It follows that there are $|S|$ number of options to minimally resolve the conflict. It is easy to see that, as the cardinality of the set S increases, the number of options to resolve the conflict increases. We use D_{nCG} to denote the cardinality of the minimal conflict set of goals (at the same level) in a given solution graph. There can be more than one set of minimal conflicting goals, which increases the total number of options to resolve conflicts. Consider two distinct sets of conflicting goals S and S', then both S and S' have goals from the same level, or from distinct levels in the solution graph. In the first case, there are *up to* $|S| \times |S'|$ options to (minimally) resolve the conflict. In the latter case, there are *up to* $|S|$ options to resolve the conflict at level i (potentially each resulting in a successor state). For each of these successor states there can be *up to* $|S'|$ options to resolve the conflicts at the following level. In other words, there are also *up to* $|S| \times |S'|$ options to resolve the conflict. We use D_{nCS} to denote the number of sets of minimal conflicting goals.

The earliest level at which α is entailed also has an impact on the length of the solution and therefore the number of calls to the partial Max-SAT solver. More precisely, if α is earliest entailed at the leaf level, then the length of the solution is *one*. As the level at which α is earliest entailed decreases (up to level 0), the solution length increases. This is taken into account during the random generation of solution graphs, i.e. the earliest level at which α is entailed varies randomly (as can be expected in a practical scenario). Since we consider reasonably large sample sizes, the probability of each scenario should be evenly distributed.

Finally, our evaluation is with respect to the "local not entail assertion" operator (i.e. with respect to a single solution graph). Therefore, the number of solution graphs in $\mathbf{G}$ is irrelevant. However, the number of solution graphs does have an impact on the number of applications of the "local not entail assertion" operator to implement the "not entail assertion". The results of our evaluation can be projected to $\mathbf{G}$ of arbitrary size.[3]

[3] In our experiments, we only consider solution graphs that entail α. In a practical setting, not all solution graphs in $\mathbf{G}$ may entail α, in which case the "local not entail assertion" operator is vacuous and we can terminate after the first step in the pre-processing procedure.

8.1.3 Summary

To sum up, we suggest assessing the scalability of the proposed implementation by measuring

- execution time (I_{Time}) in seconds,
- number of calls to SAT solver (I_{nSAT}),
- number of calls to partial Max-SAT solver (I_{npMSAT}),

with respect to the drivers:

- number of levels in solution graph ($D_{nLevels}$),
- number of clauses ($D_{nclauses}$),
- number of options to make α non-derivable (approximated by D_{nCG} and D_{nCS}).

8.2 Experimental Preliminaries

The random generation of solution graphs and SAT-problems is described below, followed by a modified formulation of the A* search as an uninformed search.

8.2.1 Random Generation of Solution Graphs

We did not have access to large goal models and therefore solution graphs were randomly generated. We believe that this is reasonable for the purpose of assessing scalability (it also has advantages, since it allows us to create large numbers of solution graphs of a chosen size). In the following, we elaborate on some preliminary technicalities before briefly describing the random generation procedure itself.

Preliminary Technicalities: To be able to generate large and valid solution graphs in reasonable time[4], each goal in the generated solution graph corresponds to a distinct unit clause. We acknowledge that this has a positive impact on the execution time, but point out that it does not have an impact on the number of calls to the SAT- or partial Max-SAT solver. We generate a consistent set of goals by ensuring that each goal in the set corresponds to a unique variable. We generate a *correct and immediate refinement* of a goal g_0 by a set of sub-goals $\{g_1, g_2, ...g_n\}$ by adding $(g_1 \wedge g_2 \wedge ... \wedge g_n) \rightarrow g_0$

[4] Generating solution graphs is computationally expensive, since it (amongst others) requires the application of adductive reasoning techniques to infer correct goal refinements.

to the $\mathcal{KB}$ (i.e. $g_1, g_2, ...g_n$ do minimally entail g_0). The fact that goals are expressed by unit clauses still allows us to create scenarios with multiple conflicts. To create a scenario where α is derivable from a combination of goals $\{g_1, g_2, ...g_n\}$ we add $(g_1 \wedge g_2 \wedge ... \wedge g_n) \rightarrow \alpha$ to the $\mathcal{KB}$ (i.e. removing any of the goals $g_1, g_2, ...g_n$ would make α non-derivable - unless it is derivable from another combination of goals).

Random Generation Procedure: In the following, all parameters of the random generation procedure are denoted by n_x, where x identifies the parameter. The random generation procedure creates n_r number of root goals and marks them as desired goals. For each root goal, the procedure creates a refinement with at least *one*- and at most n_s number of sub-goals. The procedure continues to create refinements for the respective sub-goals until the resulting graph reaches a depth of n_{Levels}. Given this generated graph, the procedure randomly picks a level and then randomly selects n_{nGC} number of goals from that level, i.e. a set $\{g_1 \wedge g_2 \wedge ... \wedge g_n\}$, adds $(g_1 \wedge g_2 \wedge ... \wedge g_n) \rightarrow \alpha$ to the $\mathcal{KB}$. The previous step is repeated n_{nCS} number of times.

8.2.2 Random Generation of SAT Problems

A SAT (or partial Max-SAT problem) corresponds to a set of clauses. A set of clauses is randomly generated, as follows. A set Var of n_{var} distinct variables is generated. A clause is a set of positive and/or negative variables. A clause c of size $n_{clausesize}$ is generated by $n_{clausesize}$-times randomly selecting a variable of Var and adding it with an equal probability as a negative- or positive variable to c. This step is repeated $n_{clauses}$ number of times to generate a set of clauses of size $n_{clauses}$.

8.2.3 Modification of the A* search as Uninformed Search

To provide better understanding of the performance of the A* we require a baseline against which it can be compared. We have chosen to compare the A* search against an implementation of its modification as an uninformed search. An uninformed search does not make use of the problem specific knowledge encoded in the heuristic function. In other words, instead of ordering the *queue* (see Algorithm 2) after the addition of the successor state, the current order of the list retained and the successor states are either added on top of the list (for *depth first search*) or at the end of the list (for *breath first search*). Note, it does not matter whether the state space is searched in a breath- or depth-first manner, since it is finite and exhaustively searched. The search space must be exhaustively searched, since otherwise we cannot be guaranteed that an

optimal solution has been found (unless we make use of context specific knowledge). The start and successor states are determined in the same way as in the A* search. It is worth mentioning that, although the uninformed search does not make use of the heuristic function, it greatly benefits from the pre-processing procedure as well as the successor function.

8.3 Experiments

The experimental results are presented and discussed in the following, but first the experimental process and set-up is described in more detail.

8.3.1 Experimental Process & Set-Up

We have split the experiment into two parts.

In the first part, we evaluated step "D" of the pre-processing procedure. This was done by randomly generating sets of clauses for $D_{nclauses} \in [100, 500, 1000, 1500, ..., 3500]$. The number of distinct variables was set to be equal to the number of clauses. Each generated clause had between one and ten variables. We only considered sets of clauses that were non-satisfiable (i.e. in case a satisfiable set was generated, it was omitted and a new set was generated). To obtain average values, we generated 50 sets for each instantiation of the driver.

In the second part, we evaluated the "local not entail assertion" operator for situations where the upper path at decision point "d2" of the pre-processing procedure is taken (i.e. where α is not derivable from the set of desired goals and step "D" is not required to be executed). We randomly generated solution graphs for $D_{Level} \in [2, ..., 16]$, $D_{nCS} \in [2, ...7]$ and $D_{CG} \in [2, ..., 7]$ by setting $n_{level} = D_{nlevel}$, $n_{nCG} = D_{nCG}$ and $n_{nCS} - D_{nCS}$ as input parameters to the random generation procedure (the other two parameters are set to $n_r = 1$ and $n_s = 2$). A solution graph with 16 levels is a realistic upper bound, since it has on average more than 2000 goals (if each refinement has at most two sub-goals). This is well within the size of solution graphs that can be expected in a practical setting (in [153] the authors report that they encountered goal models with up to 640 goals). We considered up to 7 conflicting sets, each with up to 7 goals. We believe that in a practical setting it is not uncommon that a solution graph contains a single conflict and therefore we believe that the considered range is not too optimistic.

To be able to obtain average results, we generated 50 solution graphs for each driver instantiation. We have chosen to generate 50 solution graphs as it allowed us to get representative average values, as well as conduct the experiments in a timely manner. For each generated solution graph, we ran the A^*- as well as *uninformed search*. Average values were obtained for each indicator by summing up the values for each individual run, divided by the number of runs (i.e. 50). To be able to perform the experiments in a timely manner, we aborted each execution of the uninformed search after 20 minutes and recorded a "time out". The experiments were conducted on an Intel Core Due 2.66Ghz with 3GB of RAM.

8.3.2 Experimental Results

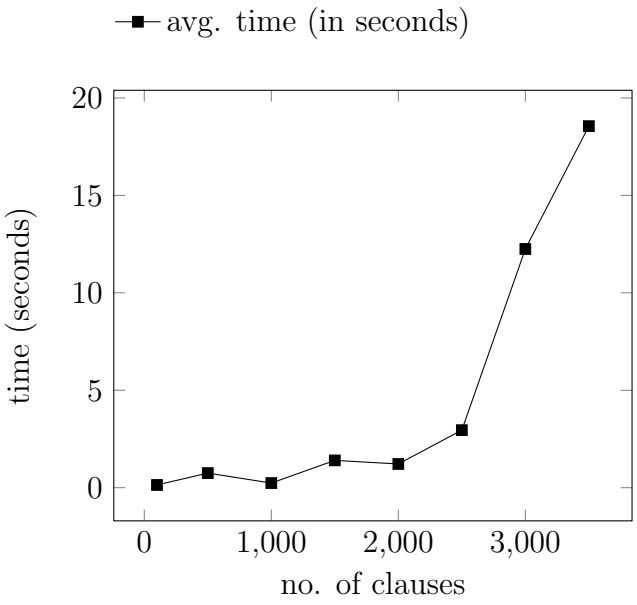

Figure 8.1: Evaluation of step "D" of the pre-processing procedure.

We now present the results of evaluating step "D" of the pre-processing procedure. Figure 8.1 shows a distribution of execution time (in seconds) over the driver $D_{nclauses}$. It can be observed that the execution time of the partial Max-SAT solver increases exponentially as the number of clauses increases. This is not surprising, since partial Max-SAT problems are known to be NP-hard. Nevertheless, problems with up to 2500 clauses can be solved within seconds. Depending on the number of clauses in the $\mathcal{KB}$ as well as the number of clauses per desired goal in the goal library, the results show that we can efficiently handle hundreds of goals. It also worth highlighting that

our implementation of the partial Max-SAT solver proposed in [108] is not optimized. The results presented [108] show that they were able to solve problems with ca. 1500 variables and up to ca. 7000 constraints in under a second (on standard hardware). Also note, step "D" of the pre-processing procedure has to be only executed once, since the results can be reused in applications of the "local not entail assertion" operator on other solution graphs in $\mathbf{G}$.

We now present results for the "local not entail assertion" operator (for situations where the upper path at decision point "d2" in the pre-processing procedure is taken, i.e. where step "D" is not required to be executed).

Figure 8.2 shows a distribution of the average values for the indicator I_{npMSAT} (for both, the *A**- and *uninformed search*) and I_{nSAT} over the driver D_{Levels}. The other two drivers are fixed to $D_{nCS} = 3$ and $D_{ncs} = 3$. We have chosen these particular values, since they are both in the middle of the respective value range. This also allows us to show results for the uninformed search before it times out.

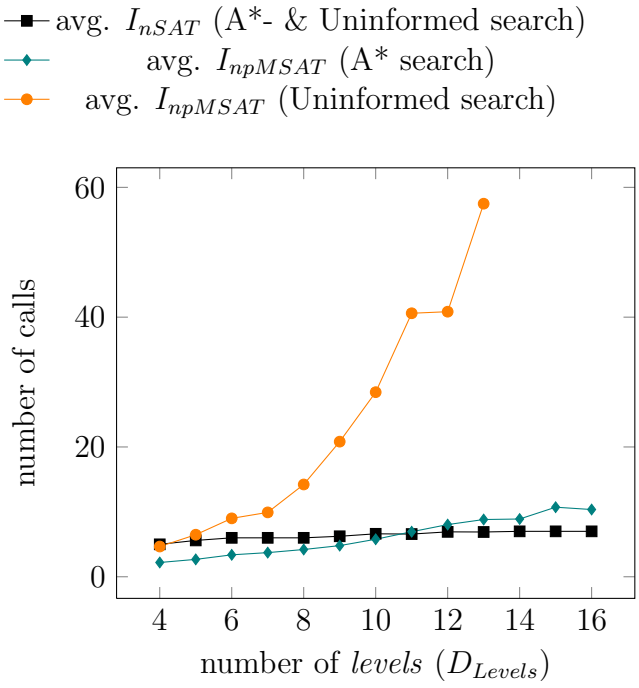

Figure 8.2: Distribution of I_{npMSAT} and I_{nSAT} over D_{Levels}.

The average number of calls to the SAT solver I_{nSAT} (black squared line) is identical for the A* search as well as Uninformed search, since it is only called during the pre-processing procedure (which is used in both the A* as well as uninformed search). The SAT solver is called a constant number of times (i.e. once) in step "A" and step

"B", but a varying number of times during step "C" of the pre-processing procedure. Recall that in step "C" a binary search is used to determine the earliest level in which the assertion α is entailed. The worst case number of steps (i.e. "probes") for a binary search is known to be $log_2(N)+1$, where N is the number of items. Recall, while the best case performance for standard binary search is 1 (a lucky guess), in our case it is $log_2(depth(\mathscr{G}))+1$. This is due to the fact that we cannot tell whether we had a lucky guess, i.e. we may find that α is derivable from level k, but cannot tell whether this is the earliest level before we have gone through the worst case number of probes. In other words, the average number of calls to a SAT solver is $log_2(depth(\mathscr{G}))+1$, which is aligned with the plot shown in Figure 8.3 it shows a slight increase of I_{nSAT} as the number of levels increases.

Figure 8.2 shows that for the A* search the average number of calls to the partial Max-SAT solver I_{npMSAT} has a constant growth rate as the number of levels increases. This is in strong contrast to the exponential increase of I_{npMSAT} for the uninformed search (note that the uninformed search timed out for level $14-16$ and therefore no values are shown). The good performance of the A* search has two main reasons. First, the A* search (as well as uninformed search) has a very moderate branching factor, since any time the "next level can be consistently added" to the solution graph in state s, the successor function returns a *single* successor state for s. Second, the A* search (as opposed to the uninformed search) rarely has to backtrack. Given a state s and its successor states s' and s'', if the *actual weight* of s' is higher than s'', then the actual weight of s' is also higher than the heuristic value of s''[5], since refinements of goals at level k are strictly worth more than all refinements together at levels greater than k. In other words, state s'' is never expanded as it cannot be a solution to the search.

Also observe that I_{npMSAT} not only grows constant but is also relatively low, which must be attributed to the small solution length. Recall, the length of the solution corresponds to the minimum number of states that connect the start state to the goal state, i.e. it corresponds to the difference between the depth of the solution graph in the start state and the depth of the solution graph in the goal state. For example, given a solution graph with a depth of 10 in the start state and a solution graph with a depth of 15 in the goal state, then the length of the solution is 5.

It must be highlighted that although the search involves multiple calls to a partial Max-SAT solver, at each call to the solver only the goals at the next immediate level

[5] Recall, that the heuristic value is given by the actual plus the estimated weight

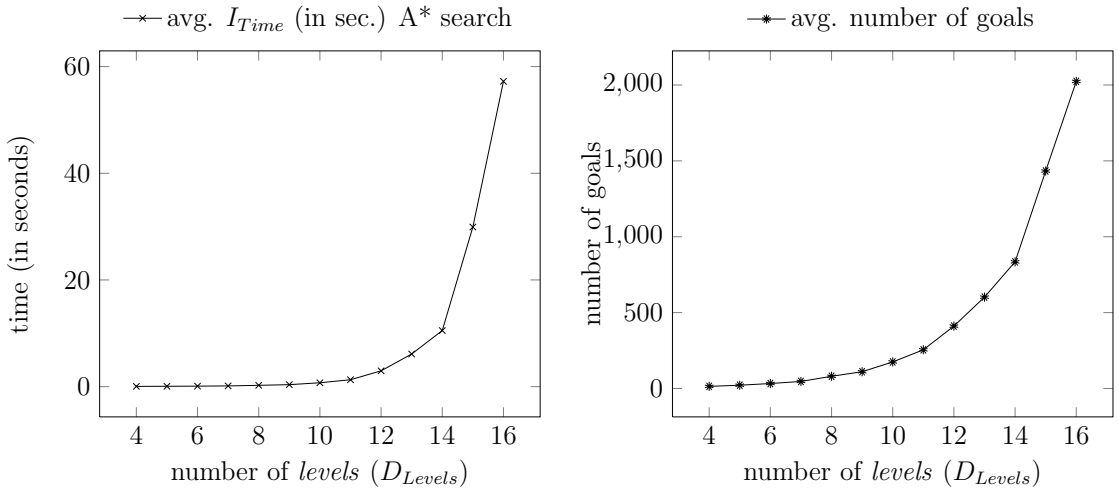

Figure 8.3: Results of experimental evaluation.

are considered. In other words, instead of solving one very large partial Max-SAT problem with all goals of the original solution graph (which would not be feasible for large solution graphs), the search splits the problem into many small problems that can be efficiently solved.

Figure 8.3 (a) shows a distribution of average execution time (I_{Time}) over a number of levels D_{Levels}. As can be observed, the execution time increases exponentially as D_{Levels} increases (the other two drivers were fixed to $D_{nCS} = 3$ and $D_{ncs} = 3$). This is primarily due to the fact that the number of goals per level increases exponentially, which in turn increases the "size" (i.e. number of variables and constraints) of the problem to be solved by the partial Max-SAT solver and consequently the execution time. For comparison, Figure 8.3 (b) shows the increase of the average number of goals per solution graph as the number of levels increases. The number of goals increases exponentially, since the number of goals per level increases by the average number of sub-goals per refinement (recall that we have set the random generation procedure such that it randomly generates $1 - 2$ sub-goals for each parent goal).

Although the execution time increases exponentially, solution graphs with 16 levels and more than 2000 goals can still be solved in under a minute, while solution graphs with 13 levels and ca. 600 goals can be solved in seconds[6]. Nevertheless, it must be taken into account that in our experiments all goals are unit-clauses, and higher exe-

[6] We conducted the experiments on an Intel Core Due 2.66Ghz with 3GB of RAM.

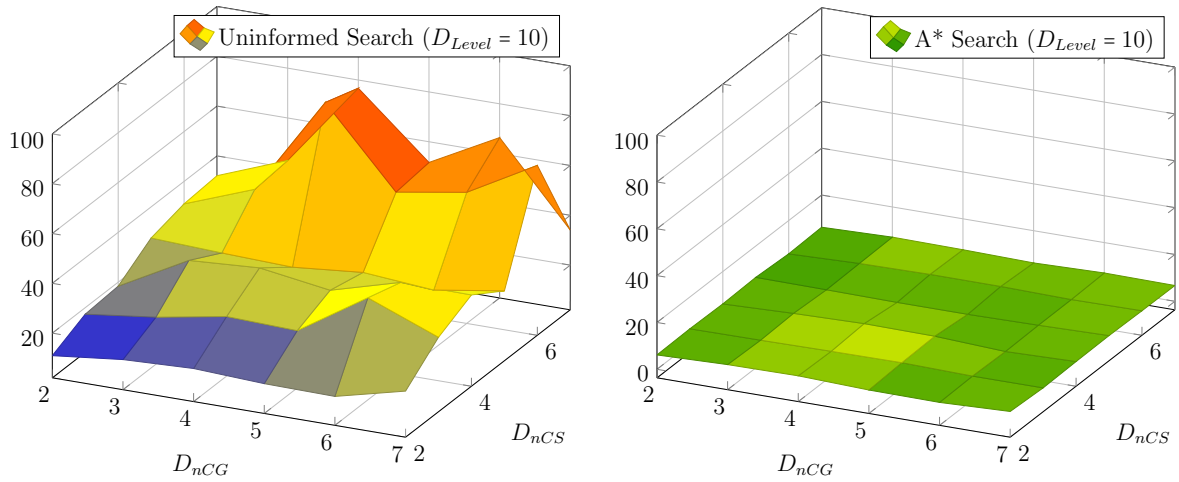

Figure 8.4: Number of calls to the partial Max-SAT solver for the (a) Uninformed search and (b) A* search.

cution times can be expected when goals are expressed by multiple clauses. This could be addressed to some extent by leveraging the techniques described in Argerlich and Manyà [10], which would allow us to more efficiently find solutions to blocks of clauses (where each goal denotes a block). We are currently extending our implementation towards this direction.

Figure 8.4 (a) and (b) show three-dimensional graphs where $M(npMSAT)$ is distributed over D_{nCS} (number of conflicting sets) and D_{nCG} (conflicting goals per set) for the *A* - and Uninformed search*. The other driver is fixed to $D_{Level} = 10$.

As can be observed in Figure 8.4 (a) variations in D_{nCS} and D_{nCG} do notably impact I_{npMSAT} during the Uninformed search. The mindful observer will have also noticed that an increase in D_{nCS} has a stronger impact on I_{npMSAT} than an increase in D_{nCG}. This can be explained as follows. First, observe that removal of a goal from a set of conflicting goals may require the removal of additional goals if they are part of the same set of sub/target-goal, as otherwise the respective refinement would be incorrect. For example, consider a set of conflicting goals $\{g_1, g_2\}$ and let both goal g_1 and g_2 be part of the same set of sub/target-goals of a parent goal g_0. Therefore removing g_1 from the solution graph also requires the removal of g_2 and vice versa. In other words, there are two goals in the set of conflicting goals, but only one option to resolve the conflict. As D_{nCG} increases, the likelihood that two goals are part of the same set of sub/target-goals also increases. Consequently, it is less and less likely that the number of options to resolve the conflict increases.

Figure 8.4 (b) shows that variations in D_{nCS} and D_{nCG} only slightly impact the number of calls to the partial Max-SAT solver during the A* search. These are good outcomes, as they suggest a relatively stable execution time for solution graphs of a particular size, regardless of the number of conflicts.[7]

8.4 Summary

This chapter discussed and presented experimental results of a prototype implementation of the "not entail assertion" operator.

In Chapter 8.1, we first motivated a set of indicators and drivers. We proposed indicators to observe execution time, as well as the required number of calls to a SAT as well as partial Max-SAT solver. The indicators were observed with respect to changes in the size of a solution graph (i.e. the number of levels in a solution graph) as well as the approximated number of options to make α non-derivable.

In Chapter 8.2, we discussed experimental preliminaries. In particular, we described a procedure for randomly generating solution graphs for different instantiations of the identified drivers. This was followed by a description of a modified version of the A* search as uninformed search, which provided us with a baseline against which to compare.

The results of the experiments conducted were presented and discussed in Chapter 8.3. The experiments showed that our implementation of the "local not entail assertion" operator can handle solution graphs with 13 levels and ca. 600 goals (which is well within the size of solution graphs that can be expected in industry) in seconds[8]. Furthermore, the experiments showed that an increase in the size of a solution graph only slightly impacts the number of calls to the SAT- and partial Max-SAT solver. This suggests (as expected) that the calls to the SAT- and partial Max-SAT solver are the main bottlenecks. This is good news, as it allows our implementation to significantly benefit from the active research in more efficient SAT and partial Max-SAT solvers. Furthermore, the experiments showed that the approximated number of options to make α non-derivable does not have an impact on the performance of the A* search, but significantly impacts the uninformed search.

[7] However, execution time is not guaranteed to be constant for solution graphs of a particular size, since some SAT instances may be harder to solve than others.

[8] We conducted the experiments on an Intel Core Due 2.66Ghz with 3GB of RAM.

Chapter 9

Conclusion

A goal model is an important artifact in an organisational context. It encodes organisational intent and serves as a motivation for- and justification of system requirements.. However, the fast changing business environment requires organisations to constantly reposition and reinvent themselves. This involves adopting new goals or giving up goals that have become undesirable or infeasible to realize. In short, it requires the goal model to be maintained.

Over the course of the previous chapters we have presented a framework that can be deployed to algorithmically support the maintenance of formal and hierarchically structured goal models. This chapter concludes our previous exposition. In particular, we first recapitulate the challenges and research questions that we set out to address. This is followed by a summary of how these challenges and research questions were addressed. The chapter ends with an outlook on different directions in which our work could be extended in the future.

9.1 Summary of Research Questions

Typically, goals are structured in AND/OR graphs. However, we have shown that there are issues when goals models, formalized as AND/OR graphs, need to be maintained. For example, goals that are removed are permanently lost, and reasoning with "derived goals" is not naturally supported, nor is the toleration of inconsistencies. While the permanent loss of goals and the need to tolerate inconsistency has been addressed in existing goal/requirements maintenance frameworks, little has been done to take into account the goal model's structure to avoid situations in which goals are unnecessarily removed or retained, as well as to leverage the structure to devise

efficient maintenance algorithms.

An important challenge of the goal model maintenance exercise is to avoid erroneous and non-minimal (i.e. sub-optimal) modifications. Algorithmic maintenance support can overcome this by systematically exploring the space of goal model configurations to identify the one(s) that optimally incorporate a given change request. There can be different kinds of change requests and consequently a family of maintenance operators is required. At the same time, minimal change can be interpreted in various ways and the formal language in which goals are specified varies, such that the maintenance operators must be instantiatable, respectively.

As the goal model is maintained, the answer(s) to the requirements problem it justifies need to be adopted, as well. This includes the identification of goals which are not fulfilled, and system functionality which does not contribute to the fulfilment of a goal. Manually performed, this exercise is often erroneous and time consuming, and therefore a (partial-) automating of this process is desirable. Maintaining an answer to the requirements problem also involves reasoning about the fulfilment of (non-functional) goals, which do not have a clear-cut satisfaction criteria. These goals typically reflect the intent to maximize or minimize a non-functional system property. We argued that such (soft-)goals can be considered as fulfilled if we can show that no better system behaviour can be found. This requires the ability to approximate, during design-time, the performance of a description of system behaviour, as well as optimization machinery to guarantee that no better behaviour (within a certain boundary) can be found.

The above challenges were consolidated in the following research question.

- **Q1:** *How can a maintenance framework that supports AND/OR graphs, but does not face their deficiencies during the maintenance process, be devised? In particular: (a) How can we represent and reason with inconsistent goals? (b) How can the justification relation between goals be taken into account when the model is maintained? (c) How can the reuse of goals be supported? (d) How can the reasoning with derived goals be supported?*

- **Q2:** *How can we algorithmically support the maintenance of formal goal models? In particular: (a) What maintenance operators are required and how can they be defined in a modular manner that is agnostic to the language in which goals are formalized or in which minimal change is defined? (b) How can we use the structure of a goal model to efficiently compute optimal change options?*

- **Q3:** *How can the exploration of answers to the functional as well as non-functional aspects of the requirements problem be supported to ensure that the system requirements accurately reflect changes in the goal model? In particular: (a) How can we assess the impact of changes to the goal model on a pool of service descriptions? (b) How can the non-functional performance of a design artifact (like a business process design) be estimated? (c) How can we algorithmically optimize the estimated non-functional performance of a business process design?*

9.2 Summary of Contributions

In Chapter 4, we have shown how AND/OR graph based goal models can be represented, without loss of generality, by a set of solution graphs. We have shown that this allows us to represent different and potentially inconsistent perspectives of intentionality, as is often required in a multiple stakeholder perspective (addressing question **Q1** (a)). Solution graphs are hierarchically structured via refinement/justification relations between goals. The distinction between desired- and dependent goals further allows us to ensure that any valid solution graph does not contain any superfluous goals, i.e. either a goal is desired or it is dependent, but supports the fulfilment of a desired goal (addressing question **Q1** (b)). As each solution graphs is derivable from a goal library, it is guaranteed that the removal of any goal from one of the solution graphs is not completely lost, but retained in the goal library (addressing question **Q1** (c)). We further showed how a hierarchical aware entailment relation (i.e. k-level entailment) can be associated with a single solution graph as well as a set of solution graphs and how this allows us to reason with derived goals (addressing question **Q1** (c)).

Over the course of Chapter 5, we have motivated a family of maintenance operators and demonstrated how these can be used to address various change requests. The maintenance operators follow the principle of minimal change and make use of an abstract proximity relation to abstract from a particular definition of minimal change. Furthermore, the operators are defined in a manner that is agnostic to the underlying formal language in which goals are formalized. This addresses question **Q2** (a). In Chapter 7, we then proceeded to show how the "not entail assertion" operator (which can also be used to implement other maintenance operators) can be efficiently implemented using A* search, which makes particular use of the solution graph structure. Our experiments with a prototype implementation were presented in Chapter 8. The

results demonstrated that the operator can scale up to goal models of a size that can be expected in a practical setting and hence showed that algorithmic support in the maintenance of formal goal model can be provided (addressing question **Q2** (b)).

In Chapter 6, we first showed how our previous work on maintaining goal models in the context of a service landscape allows us to address functional aspects of the requirements problem in the context of our maintenance framework (addressing question **Q3** (a). We then showed how our previous work on Green Business Process Management is the basis for providing answers to the non-functional aspects of the requirements problem. In particular, we showed how modelling and correlating resource models with business process designs can be used to estimate the non-functional performance (we focused on CO2 emissions) of activities at design time. We then showed how these (quantitative or qualitative) values can be accumulated through the process design to receive cumulative performance values for the process design. This allows us to estimate the non-functional performance characteristic of a design artifact which has not yet been implemented. This is essential to reason over the fulfilment of a non-functional goal (addressing question **Q3** (b). We proceeded to show how the expected non-functional performance of a (business process) design artifact can be improved in a semi-automated manner, such that the artifact remains to realize the correlated goal. This was then shown to be the basis for making statements about the fulfilment of (non-functional) soft-goals (addressing question **Q3** (c)).

In answering the above research question, we have addressed various gaps in the existing literature. Table 9.1 summarizes the gaps that are addressed in this thesis in juxtaposition with existing work.

	GSA	QC	ARC	THEORIST	REFORM	REKB	Our Work
just. struc.	∘[1]	-	-	-	∘[1]	x	x
hierarch. eintail.	-[2]	-	-	-	-	-	x
goal reuse	x	x	-	x	x	x	x
lang. agnostic	x	-	-	x	x	-	x
minimal change	x	-	-	x	x	x	x
tol. incon.	-	x	-	x	x	x	x
impl.+eval.	-	-	-	-	-	x	x

Table 9.1: Comparison of supported features ("x" = support, "∘" = partial support, "-" = no support).

In Table 9.1, GSA refers to the work by MacNish and her co-authors [113, 38], QC refer to the work by Nuseibeh and his co-authors [126, 82], ARC refers to the work by Garces and his co-authors [52, 31], THEORIST refers to the work by Zwhogi et al. [170], REFORM refers to the work by Ghose [64, 63] and REKB refers to the work by Ernst and his co-authors [42, 44, 45].

9.3 Future Work

The body of work presented in this thesis can be extended in various interesting ways.

- We used semantic effect annotated business process designs or services to automatically establish realization links to goals in the goal model and showed how this enables us to partially automate the identification of answers to the requirements problem. However, it must be emphasized that the analysis is restricted to these types of artifacts. It would therefore be interesting to explore, in the context of our framework, how realization links between goals and other requirements artifacts can be established in an automated or semi-automated manner.

- Furthermore, we did not take into account that business process designs or services (or other artifacts) may be interrelated, such that each individual process design might realize goals in the goal model but not be conjointly executable with others. Amongst others, this begs the following questions. What interrelations do exist? How can they be (in the best case automatically) identified? How can these interrelations be represented and reasoned with in the context of the proposed maintenance framework?

- The business process improvement procedure, presented in Chapter 6.2.3, requires the existence of an initial process design that already realizes a given goal. Unfortunately, this procedure is not applicable in situations where new goals have been added to a solution graph and we are interested in identifying processes that realize them. It would therefore be interesting to explore how the process improvement procedure could be turned into a process construction pro-

[1] Allows to capture a requirement's justification, but does not avoid situations in which unjustified requirements remain after a change.

[2] An entailment relation is discussed to relate elements at different levels of refinement.

cedure. Such algorithmic support would further ease the exploration of answers to the requirements problem.

- Chapter 6.2.4 briefly discusses the fulfilment of soft-goals as part of the search for a solution to the requirements problem. However, soft-goals are not part of the maintenance story. Nevertheless, soft-goals are deemed to change as the strategic focus of an organisation shifts. This poses the following questions. How can soft-goals be represented and maintained as part of our goal model formalization? In particular, how can we ensure that new soft-goals (i.e. objective functions) are adopted in a manner that does not conflict with other soft-goals and what are sensible intuitions of minimal change?

- In Chapter 5.2.5, we sketched how the "completion" operator can be implemented via an mixed initiative approach. In this approach, the software tool and the human requirements engineer work hand in hand to compute and select correct goal refinements. However, some of these refinements may have already been computed by previous applications of the operator, resulting in redundant and superfluous computation and human effort. This begs the question of how to avoid this redundant work? One possible option (the one we are currently exploring) is to store and maintain all identified refinements in (what could be referred to as) a "refinement library". While the software tool would still have to ensure correctness of a refinement taken from the refinement library (a refinement may have become incorrect due to a change in the domain knowledge base), we believe that this involves less computational effort than computing the refinement from scratch. (We are currently exploring the application of adductive reasoning techniques for this exercise.)

References

[1] C.E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2), 1985.

[2] V. Andrikopoulos, S. Benbernou, and M.P. Papazoglou. On the evolution of services. *IEEE Transactions on Software Engineering*, 38(3):609–628, 2012.

[3] Annie I. Antón, W. Michael McCracken, and Colin Potts. Goal decomposition and scenario analysis in business process reengineering. In *Proceedings of Advanced Information Systems Engineering*, CAiSE '94, pages 94–104, London, UK, UK, 1995. Springer-Verlag.

[4] G. Aldo Antonelli. Non-monotonic logic, 2012. `http://plato.stanford.edu/archives/win2012/entries/logic-nonmonotonic/`; last accessed 31.07.2013.

[5] G. Antoniou and A. Ghose. Formal requirements engineering: Tracing and resolving conflicts using nonmonotonic representations. In *In Intelligent Software Engineering: Papers from the AAAI-99 Workshop*. AAAI Press, 1999.

[6] G. Antoniou, Cara MacNish, and N. Y. Foo. A note on the refinement of nonmonotonic knowledge bases. *Knowledge and Information Systems*, 2:479–486, 2000.

[7] G. Antoniou and M. A Williams. *Nonmonotonic reasoning*. Cambridge, Mass. MIT Press, 1997.

[8] G. Antoniou and M.-A. Williams. Revising default theories. In *Tools with Artificial Intelligence, 1998. Proceedings. Tenth IEEE International Conference on*, pages 423 –430, 1998.

[9] Grigoris Antoniou. A tutorial on default logics. *ACM Comput. Surv.*, 31(4):337–359, 1999.

[10] J. Argerlich and Felip Manyà. Exact max-sat solvers for over-constrained problems. *Journal of Heuristics*, 12(12):375–392, 2006.

[11] Australian Government Department of Climate Change. About the national greenhouse and energy reporting act. `http://www.climatechange.gov.au/reporting/`, 2009. last accessed 01.03.2010.

[12] Australian Government Department of Climate Change. National green house accounts (nga) factors. `http://www.climatechange.gov.au/workbook/pubs/workbook-jun09.pdf`, 2009. last accessed 01.03.2010.

[13] R. Balzer. Tolerating inconsistency. In *Proceedings of the 13th international conference on Software engineering*, pages 158–165. IEEE Computer Society Press, 1991.

[14] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy goals for requirements-driven adaptation. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 125–134. IEEE, 2010.

[15] J. Bell and Z. Huang. Dynamic goal hierarchies. *Intelligent Agent Systems Theoretical and Practical Issues*, pages 88–103, 1997.

[16] Philippe Besnard and Anthony Hunter. Quasi-classical logic: Non-trivializable classical reasoning from inconsistent information. In Christine Froidevaux and Jrg Kohlas, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, volume 946 of *Lecture Notes in Computer Science*, pages 44–51. Springer Berlin Heidelberg, 1995.

[17] Armin Biere. *Handbook of satisfiability*, volume 185. IOS Press, 2009.

[18] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM (JACM)*, 44(2):236, 1997.

[19] G. Brewka. Belief revision in a framework for default reasoning. *The logic of theory change*, pages 206–222, 1991.

[20] Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.

[21] Amit K. Chopra, Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Modeling and reasoning about service-oriented applications via goals and commitments. In *CAiSE*, 2010.

[22] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, et al. Web services description language (wsdl) 1.1. `http://www.w3.org/TR/wsdl`, 2001. last accessed 02.08.2013.

[23] L. Chung, P. Leite, and J. Cesar. On non-functional requirements in software engineering. In *Conceptual Modeling: Foundations and Applications*, page 379. Springer-Verlag, 2009.

[24] Enzo Colombo, John Mylopoulos, and Paola Spoletini. Modeling and analyzing context-aware composition of services. In Boualem Benatallah, Fabio Casati, and Paolo Traverso, editors, *Service-Oriented Computing - ICSOC 2005*, volume 3826 of *Lecture Notes in Computer Science*, pages 198–213. Springer Berlin / Heidelberg, 2005.

[25] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[26] C. da Costa Pereira and A.G.B. Tettamanzi. A belief-desire framework for goal revision. In *Proceedings of the 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems: Part I*, pages 164–171. Springer-Verlag, 2007.

[27] C. da Costa Pereira, A.G.B. Tettamanzi, and L. Amgoud. Goal revision for a rational agent. In *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*, pages 747–748, 2006.

[28] A. Dardenne, A. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1-2):3–50, 1993.

[29] Anne Dardenne, Stephen Fickas, and Axel van Lamsweerde. Goal-directed concept acquisition in requirements elicitation. In *Proceedings of the 6th international workshop on Software specification and design*, IWSSD '91, pages 14–21, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press.

[30] R. Darimont and A. Van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. *ACM SIGSOFT Software Engineering Notes*, 21(6):179–190, 1996.

[31] A.S. d'Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer. Combining abductive reasoning and inductive learning to evolve requirements specifications. *Software, IEE Proceedings* -, 150(1):25 – 38, feb. 2003.

[32] Johan de Kleer. An Assumption-Based TMS. *Artificial Intelligence*, 28(2):127–162, 1986.

[33] Johan de Kleer. Extending the ATMS. *Artificial Intelligence*, 28(2):163 – 196, 1986.

[34] James P. Delgrande and W. Ken Jackson. Default Logic Revisited. In *Principles of Knowledge Representation and Reasoning*, pages 118–127, 1991.

[35] Jon Doyle. A truth maintenance system. *Artif. Intell.*, 12(3):231–272, 1979.

[36] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, and A. Rifaut. A knowledge representation language for requirements engineering. *Proceedings of the IEEE*, 74(10):1431 – 1444, oct. 1986.

[37] Eric Dubois, Philippe Bois, and Andr Rifaut. Elaborating, structuring and expressing formal requirements of composite systems. In Pericles Loucopoulos, editor, *Advanced Information Systems Engineering*, volume 593 of *Lecture Notes in Computer Science*, pages 327–347. Springer Berlin Heidelberg, 1992.

[38] D. Duffy, C. MacNish, J. McDermid, and P. Morris. A framework for requirements analysis using automated reasoning. In *Advanced Information Systems Engineering*, pages 68–81. Springer, 1995.

[39] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on*, pages 411 – 420, 2001.

[40] S. Easterbrook and B. Nuseibeh. Managing inconsistencies in an evolving specification. In *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, pages 48 – 55, 1995.

[41] R. Eramo, A. Pierantonio, J.R. Romero, and A. Vallecillo. Change Management in Multi-Viewpoint System Using ASP. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th*, pages 433–440. IEEE, 2009.

[42] N.A. Ernst, A. Borgida, and I. Jureta. Finding incremental solutions for evolving requirements. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 15 –24, 29 2011-sept. 2 2011.

[43] Neil Ernst, John Mylopoulos, Alex Borgida, and Ivan Jureta. Reasoning with optional and preferred requirements. In Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand, editors, *Conceptual Modeling ER 2010*, volume 6412 of *Lecture Notes in Computer Science*, pages 118–131. Springer Berlin / Heidelberg, 2010.

[44] Neil Alexander Ernst. *Software Evolution: a Requirements Engineering Approach*. PhD thesis, University of Toronto, 2012.

[45] NeilA. Ernst, Alexander Borgida, John Mylopoulos, and Ivan. Jureta. Agile requirements evolution via paraconsistent reasoning. In Jolita Ralyt, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 382–397. Springer Berlin Heidelberg, 2012.

[46] NeilA. Ernst, John Mylopoulos, and Yiqiao Wang. Requirements evolution and what (research) to do about it. In Kalle Lyytinen, Pericles Loucopoulos, John Mylopoulos, and Bill Robinson, editors, *Design Requirements Engineering: A Ten-Year Perspective*, volume 14 of *Lecture Notes in Business Information Processing*, pages 186–214. Springer Berlin Heidelberg, 2009.

[47] European Commission. Monitoring, reporting and verification. `http://ec.europa.eu/environment/climat/emission/mrg_en.htm`, 2009. last accessed 01.03.2010.

[48] A.C.W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multiperspective specifications. *Software Engineering, IEEE Transactions on*, 20(8):569 –578, aug 1994.

[49] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.

[50] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177–201, 1993.

[51] M. Galster and E. Bucherer. A business-goal-service-capability graph for the alignment of requirements and services. In *IEEE Congress on Services*, 2008.

[52] A. S. d'Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer. An analysis-revision cycle to evolve requirements specifications. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 354–, Washington, DC, USA, 2001. IEEE Computer Society.

[53] P. Gardenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, 1988.

[54] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[55] S. Ghanavati, D. Amyot, and L. Peyton. Towards a framework for tracking legal compliance in healthcare. In *Caise*, 2007.

[56] A. Ghose, K. Hoesch-Klohe, L. Hinsche, and L.-S. Le. Green business process management: A research agenda. *Australian Journal of Information Systems*, 16(2), 2009.

[57] A. Ghose and G. Koliadis. Pctk: A toolkit for managing business process compliance. In *Proceedings of the Second International Workshop on Juris-informatics*, 2008.

[58] A. Ghose, G. Koliadis, and A. Chueng. Process Discovery from Model and Text Artefacts. In *IEEE Congress on Services*, pages 167–174, 2007.

[59] Aditya Ghose and G.Koliadis. Pctk: A toolkit for managing business process compliance. In *Proc. of the 2008 International Workshop on Juris-Informatics (JURISIN-2008)*, 2008.

[60] Aditya Ghose and Randy Goebel. Belief states as default theories: Studies in non-prioritized belief change. In *ECAI*, pages 8–12, 1998.

[61] Aditya Ghose and George Koliadis. Auditing business process compliance. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC-2007)*, 2007.

[62] Aditya K. Ghose, Nanjangud C. Narendra, Karthikeyan Ponnalagu, Anurag Panda, and Atul Gohad. Goal-driven business process derivation. In *Proceedings of the 9th international conference on Service-Oriented Computing*, pages 467–476, Berlin, Heidelberg, 2011. Springer-Verlag.

[63] A.K. Ghose. Managing requirements evolution: Formal support for functional and non-functional requirements. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 77–84. Citeseer, 1999.

[64] A.K. Ghose. Formal Tools for Managing Inconsistency and Change in RE. In *Proceedings of the 10th International Workshop on Software Specification and Design*, page 171. IEEE Computer Society, 2000.

[65] A.K. Ghose and University of Alberta. Dept. of Computing Science. *Practical belief change*. PhD thesis, University of Alberta, 1995.

[66] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal Reasoning Techniques for Goal Models. *Journal on Data Semantics*, pages 1–20, 2003.

[67] Fred Glover and Claude McMillan. The general employee scheduling problem: an integration of ms and ai. *Comput. Oper. Res.*, 13(5):563–573, may 1986.

[68] Sol Greenspan, John Mylopoulos, and Alex Borgida. On Formal Requirements Modeling Languages: RML revisited. In *Proceedings of the 16th international conference on Software engineering*, ICSE '94, pages 135–147, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[69] Sol J. Greenspan, John Mylopoulos, and Alex Borgida. Capturing more world knowledge in the requirements specification. In *Proceedings of the 6th international conference on Software engineering*, ICSE '82, pages 225–234, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.

[70] Object Management Group. Business Motivation Model (BMM) Version 1.1. `http://www.omg.org/spec/BMM/1.1/PDF`, 2010. last accessed 31.07.2013.

[71] Sven Ove Hansson. Logic of belief revision. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2011 edition, 2011. `http://plato.stanford.edu/archives/fall2011/entries/logic-belief-revision/` last accessed 31.07.2013.

[72] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[73] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. Softw. Eng. Methodol.*, 5, July 1996.

[74] A. Hibner and K. Zielinski. Semantic-based dynamic service composition and adaptation. In *Services, 2007 IEEE Congress on*, pages 213 –220, july 2007.

[75] Kerry Hinge, Aditya Ghose, and George Koliadis. Process seer: A tool for semantic effect annotation of business process models. In IEEE Computer Society Press, editor, *Proc. of the 13th IEEE International EDOC Conference (EDOC-2009)*, 2009.

[76] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[77] K. Hoesch-Klohe and A. Ghose. Business Process Improvement in Abnoba. In *ICSOC 2010 International Workshops PAASC, WESOA, SEE, and SC-LOG*. Springer LNCS, 2010.

[78] K. Hoesch-Klohe and A. Ghose. Carbon-Aware Business Process Design in Abnoba. In *Proceedings of the 8th International Conference on Service Oriented Computing*, 2010.

[79] K. Hoesch-Klohe and A. Ghose. Towards Green Business Process Management. In *Proceedings of the 7th International Conference on Services Computing (Industry and Application Track)*, 2010.

[80] K. Hoesch-Klohe, A. Ghose, and Hoa Khanh Dam. Maintaining Motivation Models (in BMM) in the Context of a (WSDL-S) Service Landscape. In *Proceedings of the 10th International Conference on Service Oriented Computing*, 2012.

[81] A. Hunter and B. Nuseibeh. Inconsistency Handling in Multi-Perspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[82] A. Hunter and B. Nuseibeh. Analysing inconsistent specifications. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 78 –86, jan 1997.

[83] S. Ingolfo, A. Siena, and J. Mylopoulos. Establishing regulatory compliance for software requirements. *ER*, 2011.

[84] Paola Inverardi and Marco Mori. Feature oriented evolutions for context-aware adaptive systems. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 93–97, New York, NY, USA, 2010. ACM.

[85] Daniel Jackson. Automating first-order relational logic. *SIGSOFT Softw. Eng. Notes*, 25(6):130–139, nov 2000.

[86] M. Shaban Jokhio. Goal-based testing of semantic web services. In *ASE*, pages 707–711, 2009.

[87] I. Jureta, A. Borgida, N. Ernst, and J. Mylopoulos. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 115 –124, 2010.

[88] Ivan Jureta, Stephane Faulkner, and Pierre-Yves Schobbens. A more expressive softgoal conceptualization for quality requirements analysis. In DavidW. Embley, Antoni Oliv, and Sudha Ram, editors, *Conceptual Modeling - ER 2006*, volume 4215 of *Lecture Notes in Computer Science*, pages 281–295. Springer Berlin Heidelberg, 2006.

[89] Ivan Jureta, John Mylopoulos, and Stéphane Faulkner. Revisiting the core ontology and problem in requirements engineering. *CoRR*, abs/0811.4364, 2008.

[90] Rim Samia Kaabi, Carine Souveyet, and Colette Rolland. Eliciting service composition in a goal driven manner. In *ICSOC*, pages 308–315, 2004.

[91] R.S. Kaplan and S.R. Anderson. Time-Driven Activity-Based Costing. *harvard business review*, page 1, 2004.

[92] Hirofumi Katsuno and Alberto O Mendelzon. On the difference between updating a knowledge base and revising it. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, volume 387, page 394, 1991.

[93] G. Koliadis and A. Ghose. Relating Business Process Models to Goal-Oriented Requirements Models in KAOS. *Lecture Notes in Computer Science*, 4303:25, 2006.

[94] G. Koliadis, A. Ghose, and M. Bhuiyan. Correlating business process and organizational models to manage change. In *Proc. of the 2006 Australasian Conference on Information Systems*, 2006.

[95] Tri. Kurniawan, AdityaK. Ghose, and Lam-Son L. A framework for optimizing inter-operating business process portfolio. In Jaroslav Pokorny, Vaclav Repa, Karel Richta, Wita Wojtkowski, Henry Linger, Chris Barry, and Michael Lang, editors, *Information Systems Development*, pages 383–396. Springer New York, 2011.

[96] Tri. Kurniawan, AdityaK. Ghose, Lam-Son L, and HoaKhanh Dam. On formalizing inter-process relationships. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 75–86. Springer Berlin Heidelberg, 2012.

[97] M.M. Kwan and PR Balasubramanian. Adding workflow analysis techniques to the IS development toolkit. In *Proceedings of the Hawii International Conference on System Science*, volume 31, pages 312–321. IEEE Institute of Electrical and Electronics, 1998.

[98] Axel Lamsweerde. Reasoning about alternative requirements options. In Alexander T. Borgida, Vinay K. Chaudhri, Paolo Giorgini, and Eric S. Yu, editors, *Conceptual Modeling: Foundations and Applications*, pages 380–397. Springer-Verlag, Berlin, Heidelberg, 2009.

[99] Axel Van Lamsweerde, Robert Darimont, and Emmanuel Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24:908–926, 1998.

[100] Alexei Lapouchnian, Yijun Yu, and John Mylopoulos. Requirements-driven design and configuration management of business processes. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 246–261. Springer Berlin Heidelberg, 2007.

[101] Lam-Son Lê, Aditya K. Ghose, Muralee Krishnan, Krishnajith M. Krishnankunju, and Konstantin Hoesch-Klohe. Correlating business objectives with services: An ontology-driven approach. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 306–313, Washington, DC, USA, 2011. IEEE Computer Society.

[102] Emmanuel Letier and Axel van Lamsweerde. Deriving operational software specifications from system goals. In *SIGSOFT FSE*, pages 119–128, 2002.

[103] Emmanuel Letier and Axel van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 53–62. ACM, 2004.

[104] Isaac Levi. Subjunctives, dispositions and chances. *Synthese*, 34:423–455, 1977.

[105] Isaac Levi. *Mild contraction: Evaluating loss of information due to loss of belief*. Oxford University Press, 2004.

[106] Keith Levi and Ali Arsanjani. A goal-driven approach to enterprise component identification and specification. *Communication of the ACM*, 45(10):45–52, 2002.

[107] S. Liaskos, M. Litoiu, M. Jungblut, and J. Mylopoulos. Goal-based behavioral customization of information systems. In *Advanced Information Systems Engineering*, pages 77–92. Springer, 2011.

[108] Mark H. Liffiton and Karem A. Sakallah. On finding all minimally unsatisfiable subformulas. In *Proceedings of the 8th international conference on Theory and Applications of Satisfiability Testing*, SAT'05, pages 173–186, Berlin, Heidelberg, 2005. Springer-Verlag.

[109] Thomas Linke and Torsten Schaub. An approach to query-answering in reiter's default logic and the underlying existence of extensions problem. In *Proceedings of the Sixth European Workshop on Logics in Artificial Intelligence*, pages 233–247. Springer, 1998.

[110] Lin Liu, Qiang Liu, Chi-Hung Chi, and Zhi Jin. Towards a service requirements modelling ontology based on agent knowledge and intentions. *International Journal of Agent-Oriented Software Engineering*, 2(3):324–349, 2008.

[111] Lin Liu and Eric Yu. Designing information systems in social context: A goal and scenario modelling approach. *Info. Syst*, 29:187–203, 2003.

[112] Q. Lu, V. Tosic, and P. Bannerman. Support for the Business Motivation Model in the WS-Policy4MASC Language and MiniZnMASC Middleware. *Service-Oriented Computing*, pages 265–279, 2011.

[113] Cara MacNish and Mary-Anne Williams. From belief revision to design revision: Applying theory change to changing requirements. In *PRICAI Workshops'96*, pages 206–220, 1996.

[114] A. Martelli and U. Montanari. Additive and/or graphs. In *Proceedings of the 3. International Joint Conferences on Artificial Intelligence*, IJCAI 73, pages 1–11, 1973.

[115] Marvin Minsky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[116] M.Muehlen. Organizational management in workflow applications–issues and perspectives. *Information Technology and Management*, 5(3):271–291, 2004.

[117] E. D. Morrison, A. Menzies, G. Koliadis, and A. K. Ghose. Business process integration: Method and analysis. In *Proc. Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009)*, 2009.

[118] M.Z. Muehlen. Resource modeling in workflow applications. In *Proceedings of the 1999 Workflow Management Conference*, pages 137–153, 1999.

[119] Arun Mukhija and Martin Glinz. Runtime adaptation of applications through dynamic recomposition of components. In *Proc. of 18th International Conference on Architecture of Computing Systems*, 2005.

[120] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems (TOIS)*, 8(4):325–362, 1990.

[121] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: a process-oriented approach. *Software Engineering, IEEE Transactions on*, 18(6):483–497, 1992.

[122] Bernhard Nebel. A knowledge level analysis of belief revision. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 301–311. Morgan Kaufmann, 1989.

[123] M. Netjes, H.A. Reijers, and W.M.P. Aalst. On the Formal Generation of Process Redesigns. In *Business Process Management Workshops*, pages 224–235. Springer, 2009.

[124] N.J. Nilsson. *Problem-solving methods in artificial intelligence*. McGraw-Hill computer science series. McGraw-Hill, 1971.

[125] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Trans. Softw. Eng.*, 20(10):760–773, oct 1994.

[126] Bashar Nuseibeh and Alessandra Russo. Using abduction to evolve inconsistent requirements specifications. In *The Use of Logical Abduction in Software Engineering 25*, 1999.

[127] Object Management Group. Business process modeling notation (bpmn) ftf beta 1 for version 2.0. `http://www.omg.org/cgi-bin/doc?dtc/09-08-14.pdf`, 2009. last accessed 21.02.2010.

[128] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11), 2007.

[129] Pavlos Peppas, Abhaya Nayak, Maurice Pagnucco, Norman Y Foo, Rex Kwok, and Mikhail Prokopenko. Revision vs. update: Taking a closer look'. In *ECAI*, pages 95–99. Citeseer, 1996.

[130] R.M. Podorozhny, B.S. Lerner, L.J. Osterweil, R.M. Podorozhny, B.S. Lerner, and L.J. Osterweil. Modeling resources for activity coordination and scheduling. In *Proceedings of Coordination 1999*, pages 307–322, 1999.

[131] David Poole, Randy G Goebel, and Romas Aleliunas. *Theorist: A logical reasoning system for defaults and diagnosis*. University of Waterloo Canada, 1986.

[132] Nauman A. Qureshi, Ivan Jureta, and Anna Perini. Requirements engineering for self-adaptive systems: core ontology and problem statement. In *Proceedings of the 23rd international conference on Advanced information systems engineering*, CAiSE'11, pages 33–47, Berlin, Heidelberg, 2011. Springer-Verlag.

[133] HA Reijers and S. Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283–306, 2005.

[134] R. Reiter. A logic for default reasoning. In Matthew L. Ginsberg, editor, *Readings in nonmonotonic reasoning*, pages 68–93. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[135] Charles Rich and Yishai A. Feldman. Seven layers of knowledge representation and reasoning in support of software development. *IEEE Trans. Softw. Eng.*, 18(6):451–469, jun 1992.

[136] W.N. Robinson. Negotiation behavior during requirement specification. In *Software Engineering, 1990. Proceedings., 12th International Conference on*, pages 268 –276, mar 1990.

[137] W.N. Robinson and S. Volkov. A meta-model for restructuring stakeholder requirements. In *Software Engineering, 1997., Proceedings of the 1997 (19th) International Conference on*, pages 140 –149, may 1997.

[138] N. Russell, A.H.M. Ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow resource patterns. In *in the 17th Conference on Advanced Information Systems Engineering (CAISE05). Porto, Portugal*, pages 13–17, 2005.

[139] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[140] M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Requirements Engineering*, 11(3):174–193, 2006.

[141] M. Sabetzadeh, S. Nejati, S. Liaskos, S. Easterbrook, and M. Chechik. Consistency checking of conceptual models via model merging. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 221 –230, oct. 2007.

[142] Pete Sawyer, Nelly Bencomo, Jon Whittle, Emmanuel Letier, and Anthony Finkelstein. Requirements-aware systems: A research agenda for re for self-adaptive systems. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 95–103, Washington, DC, USA, 2010. IEEE Computer Society.

[143] Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos. Simple and minimum-cost satisfiability for goal models. In *16th Conference On Advanced Information Systems Engineering (CAiSE*04)*, pages 20–35. Springer, 2004.

[144] D. Sheridan. The optimality of a fast cnf conversion and its use with sat. In *Proceedings of SAT, International Conference on Theory and Applications of Satisfiability Testing, Vancouver (Canada)*. Citeseer, 2004.

[145] Alberto Siena, Giampaolo Armellin, Gianluca Mameli, John Mylopoulos, Anna Perini, and Angelo Susi. Establishing regulatory compliance for information system requirements: An experience report from the health care domain. In *ER*, 2010.

[146] Ian Sommerville and Gerald Kotonya. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[147] V.E.S. Souza, A. Lapouchnian, and J. Mylopoulos. (requirement) evolution requirements for adaptive systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 155 –164, june 2012.

[148] George Spanoudakis and Andrea Zisman. Software traceability: A roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing, 2004.

[149] E. Burton Swanson. The dimensions of maintenance. In *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, pages 492–497, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[150] Hamdy A Taha. *Operations research: an introduction*, volume 8. Prentice hall Englewood Cliffs, 1997.

[151] A.G.B. TETTAMANZI and C.C. PEREIRA. Towards a framework for goal revision. In *Proceedings of the 18. Belgium-Netherlands conference on artificial intelligence (BNAIC), 5-6 Oct 2006, Namur, Belgium.*, 2006.

[152] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, volume 249, page 263. Toronto, Canada, 2001.

[153] A. van Lamsweerde. Goal-oriented requirements enginering: a roundtrip from research to practice. In *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pages 4 – 7, sept. 2004.

[154] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Trans. Softw. Eng.*, 26(10):978–1005, oct 2000.

[155] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 100–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[156] Yiqiao Wang and John Mylopoulos. Self-repair through reconfiguration: A requirements engineering approach. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 257–268, Washington, DC, USA, 2009. IEEE Computer Society.

[157] B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. *Lecture Notes in Computer Science*, 4495:574, 2007.

[158] I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: On the semantic consistency of executable process models. In *Sixth European Conference on Web Services*, pages 102–111. IEEE, 2008.

[159] J. Whittle, P. Sawyer, N. Bencomo, B.H.C. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 79 –88, 31 2009-sept. 4 2009.

[160] Inc. Wikipedia: The Free Encyclopedia. Wikimedia Foundation. Cloud computing. `http://en.wikipedia.org/wiki/Cloud_computing`, 2013. last accessed 15.07.2013.

[161] Inc. Wikipedia: The Free Encyclopedia. Wikimedia Foundation. Graph (mathematics). `http://en.wikipedia.org/wiki/Graph_(mathematics)`, 2013. last accessed 15.07.2013.

[162] Inc. Wikipedia: The Free Encyclopedia. Wikimedia Foundation. Sarbanesoxley act. `http://en.wikipedia.org/wiki/Sarbanes%E2%80%93Oxley_Act`, 2013. last accessed 15.07.2013.

[163] Inc. Wikipedia: The Free Encyclopedia. Wikimedia Foundation. Service-oriented architecture. `http://en.wikipedia.org/wiki/Service-oriented_architecture`, 2013. last accessed 15.07.2013.

[164] M-A. Williams. Towards a practical approach to belief revision: Reason-based change. In L.C. Aiello and C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 412–421. Morgan Kaufmann Publishers, 1996.

[165] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, page 226, 1997.

[166] Eric Siu-Kwong Yu. *Modelling strategic relationships for process reengineering.* PhD thesis, University of Toronto, Toronto, Ont., Canada, Canada, 1996.

[167] E.S.K. Yu and J. Mylopoulos. From ER to 'AR'-modelling strategic actor relationships for business process reengineering. *International Journal of Cooperative Information Systems*, 4:125–144, 1995.

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

[169] Yi Zhou, Leendert van der Torre, and Yan Zhang. Partial goal satisfaction and goal change: weak and strong partial implication, logical properties, complexity. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, pages 413–420, 2008.

[170] D. Zowghi, A. K. Ghose, and P. Peppas. A framework for reasoning about requirements evolution. In *In Proceedings of PRICAI 96, number 1114 in LNAI*, pages 157–168. Springer-Verlag, 1996.

[171] Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993 – 1009, 2003.