Requirements Evolution and Evolution Requirements with Constrained Goal Models *

Chi Mai Nguyen, Roberto Sebastiani, Paolo Giorgini, and John Mylopoulos

DISI, University of Trento, Italy

Abstract. We are interested in supporting software evolution caused by changing requirements and/or changes in the operational environment of a software system. For example, users of a system may want new functionality or performance enhancements to cope with growing user population (changing requirements). Alternatively, vendors of a system may want to minimize costs in implementing requirements changes (evolution requirements). We propose to use Constrained Goal Models (CGMs) to represent the requirements of a system, and capture requirements changes in terms of incremental operations on a goal model. Evolution requirements are then represented as optimization goals that minimize implementation costs or customer value. We can then exploit reasoning techniques to derive optimal new specifications for an evolving software system. CGMs offer an expressive language for modelling goals that comes with scalable solvers that can solve hybrid constraint and optimization problems using a combination of Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT) techniques. We evaluate our proposal by modeling and reasoning with a goal model for the meeting scheduling examplar.

1 Introduction

We have come to live in a world where the only constant is change. Changes need to be accommodated by any system that lives and operates in that world, biological and/or engineered. For software systems, this is a well-known problem referred to as software evolution. There has been much work and interest on this problem since Lehman's seminal proposal for laws of software evolution [3]. However, the problem of effectively supporting software evolution through suitable concepts, tools and techniques is still largely open. And software evolution still accounts for more than 50% of total costs in a software system's lifecycle.

We are interested in supporting software evolution caused by changing requirements and/or environmental conditions. Specifically, we are interested in models that capture such changes, also in reasoning techniques that derive optimal new specifications for a system whose requirements and/or environment have changed. Moreover, we are interested in discovering new classes of evolution requirements, in the spirit of [8] who proposed such a class for adaptive software systems. We propose to model requirements changes through changes to a goal model, and evolution requirements as optimization

* This research was partially supported by the ERC advanced grant 267856, 'Lucretius: Foundations for Software Evolution' and by SRC GRC Research Project 2012-TJ-2266 WOLF.

goals, such as "Minimize costs while implementing new functionality". Our research baseline consists of an expressive framework for modelling and reasoning with goals called Constrained Goal Models (hereafter CGMs) [4]. The CGM framework is founded on and draws much of its power from Satisfiability Modulo Theories (SMT) and Optimization Modulo Theories (OMT) solving techniques [1, 6].

The contributions of this paper include a proposal for modelling changing requirements in terms of changes to a CGM model, but also the identification of a new class of evolution requirements, expressed as optimization goals in CGM. In addition, we show how to support reasoning with changed goal models and evolution requirements in order to derive optimal solutions. [1]

2 Background: Constrained Goal Models

SMT($\mathcal{LRA}$) and OMT($\mathcal{LRA}$). *Satisfiability Modulo the Theory of Linear Rational Arithmetic (SMT($\mathcal{LRA}$))* [1] is the problem of deciding the satisfiability of arbitrary formulas on atomic propositions and constraints in linear arithmetic over the rationals. *Optimization Modulo the Theory of Linear Rational Arithmetic (OMT($\mathcal{LRA}$))* [6] extends SMT($\mathcal{LRA}$) by searching solutions which optimize some $\mathcal{LRA}$ objective(s). Efficient OMT($\mathcal{LRA}$) solvers like OPTIMATHSAT [7] allow for handling formulas with thousands of Boolean and rational variables [6, 4].

A Working Example. We recall from [4] the main ideas of Constrained Goal Models (CGM's) and the main functionalities of our CGM-Tool through a meeting scheduling example (Figure 1). We call *elements* both goals and domain assumptions. Labeled bullets at the merging point of the edges connecting a group of source elements to a target element are *refinements* (e.g., $(\mathsf{GoodParticipation}, \mathsf{MinimalConflict}) \xrightarrow{R_{20}} \mathsf{GoodQualitySchedule}$), while the R_is denote their labels. The label of a refinement can be omitted when there is no need to refer to it explicitly.

Intuitively, requirements represent desired states of affairs we want the system-to-be to achieve (either mandatorily or possibly); they are progressively refined into intermediate goals, until the process produces actionable goals (tasks) that need no further decomposition and can be executed; domain assumptions are propositions about the domain that need to hold for a goal refinement to work. Refinements are used to represent the alternatives of how to achieve an element; a refinement of an element is a conjunction of the sub-elements that are necessary to achieve it.

The main objective of the CGM in Figure 1 is to achieve the requirement ScheduleMeeting, which is *mandatory*. ScheduleMeeting has only one candidate refinement R_1, consisting in five sub-goals: CharacteriseMeeting, CollectTimetables, FindASuitableRoom, ChooseSchedule, and ManageMeeting. Since R_1 is the only refinement of the requirement, all these sub-goals must be satisfied in order to satisfy

[1] **Note.** This paper was reduced to the current size from its original 14-page length. Accordingly, we have made available an extended version of [5] including *(i)* all figures of the examples which are described only verbally here, *(ii)* the *formalization* of the problem of automatically handling CGM evolutions and evolution requirements for CGMs, *(iii)* an overview of our tool implementing the presented approach, *(iv)* an overview of related work, with a comparison wrt. previous approaches, *(v)* some conclusions and description of future work.

it. There may be more than one way to refine an element; e.g., CollectTimetables is further refined either by R_{10} into the single goal ByPerson or by R_2 into the single goal BySystem. The subgoals are further refined until they reach the level of domain assumptions and tasks.

Some requirements can be "*nice-to-have*", like LowCost, MinimalEffort, FastSchedule, and GoodQualitySchedule (in blue in Figure 1). They are requirements that we would like to fulfill with our solution, provided they do not conflict with other requirements. To this extent, in order to analyze interactively the possible different realizations, one can interactively mark [or unmark] requirements as satisfied, thus making them mandatory (if unmarked, they are nice-to-have ones). Similarly, one can interactively mark/unmark (effortful) tasks as denied, or mark/unmark some domain assumption as satisfied or denied. More generally, one can mark as satisfied or denied every goal or domain assumption. We call these marks *user assertions*.

In a CGM, elements and refinements are enriched by user-defined *constraints*, which can be expressed either graphically as *relation edges* or textually as *Boolean or SMT*($\mathcal{LRA}$) *formulas*. We have three kinds of relation edges. *Contribution edges* "$E_i \xrightarrow{++} E_j$" between elements (in green in Figure 1), like "ScheduleAutomatically $\xrightarrow{++}$ MinimalConflicts", mean that if the source element E_i is satisfied, then also the target element E_j must be satisfied (but not vice versa). *Conflict edges* "$E_i \xleftrightarrow{--} E_j$" between elements (in red), like "ConfirmOccurrence $\xleftrightarrow{--}$ CancelMeeting", mean that E_i and E_j cannot be both satisfied. *Refinement bindings* "$R_i \longleftrightarrow R_j$" between two refinements (in purple), like "$R_2 \longleftrightarrow R_7$", are used to state that, if the target elements E_i and E_j of the two refinements R_i and R_j, respectively, are both satisfied, then E_i is refined by R_i if and only if E_j is refined by R_j. Intuitively, this means that the two refinements are bound, as if they were two different instances of the same choice.

It is possible to enrich CGMs with logic formulas, representing arbitrary logic constraints on elements and refinements. In addition to Boolean constraints, it is also possible to use numerical variables to express different numerical attributes of elements (such as cost, worktime, space, fuel, etc.) and constraints over them. For example, in Figure 1 we associate to UsePartnerInstitutions and UseHotelsAndConventionCenters a cost value of 80€ and 200€ respectively, and we associate "$(\mathsf{cost} < 100€)$" as a prerequisite constraint for the nice-to-have requirement LowCost. Implicitly, this means that no realization involving UseHotelsAndConventionCenters can realize this requirement.

We suppose now that ScheduleMeeting is asserted as satisfied (i.e. it is mandatory) and that no other element is asserted. Then the CGM in Figure 1 has more than 20 possible *realizations*. The sub-graph which is highlighted in yellow describes one of them. Intuitively, a realization of a CGM under given user assertions (if any) represents one of the alternative ways of refining the mandatory requirements (plus possibly some of the nice-to-have ones) in compliance with the user assertions and user-defined constraints. It is a sub-graph of the CGM including a set of satisfied elements and refinements: it includes all mandatory requirements, and [resp. does not include] all elements satisfied [resp. denied] in the user assertions; for each non-leaf element included, at least one of its refinement is included; for each refinement included, all its target elements are included; finally, a realization complies with all relation edges and with all constraints.

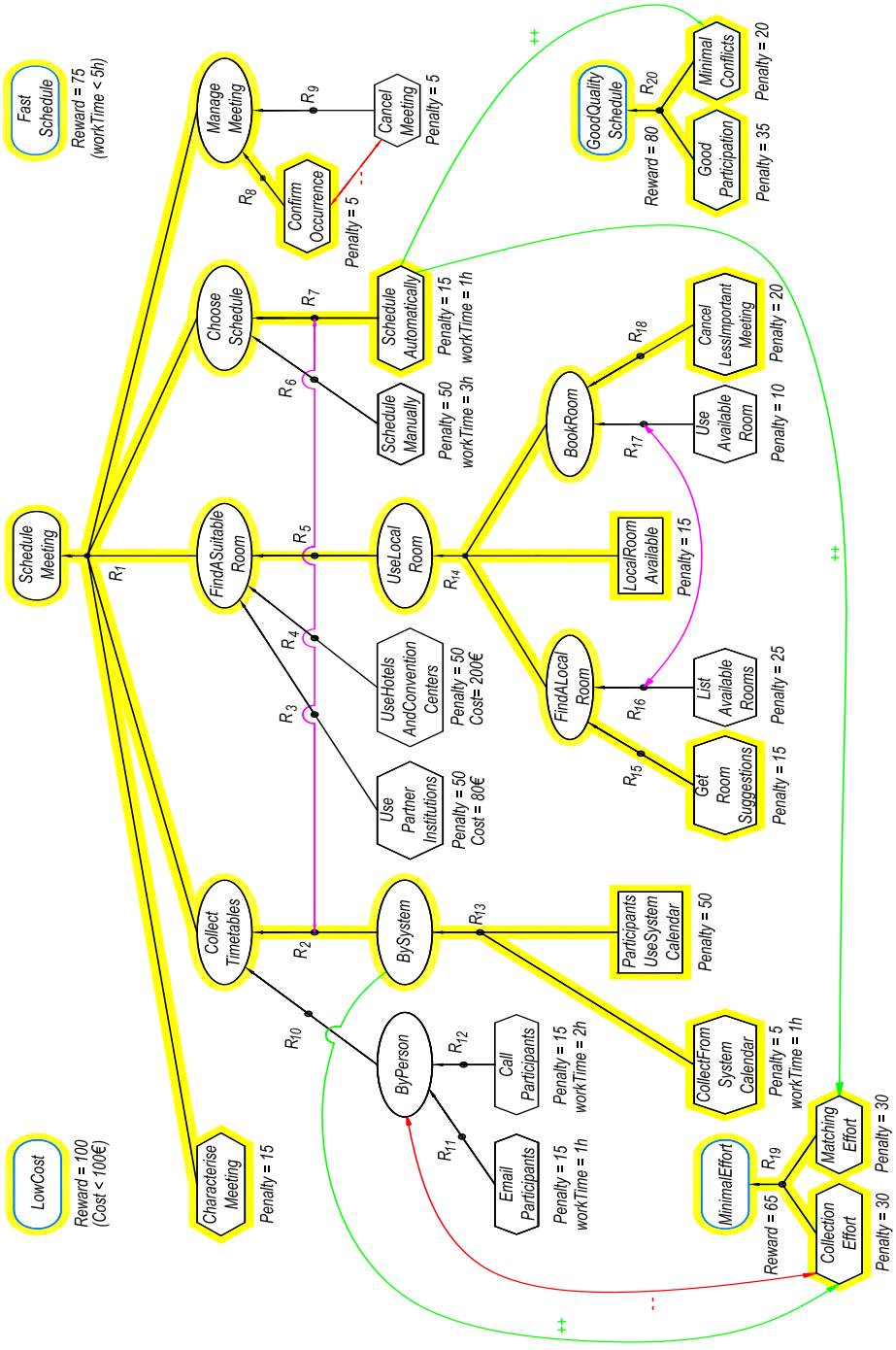

Fig. 1. A CGM $\mathcal{M}_1$, with a realization μ_1 minimizing lexicographically: the difference Penalty-Reward, workTime, and cost. Notationally, round-corner rectangles (e.g., ScheduleMeeting) are root goals, representing stakeholder *requirements*; ovals (e.g. CollectTimetables) are *intermediate goals*; *hexagons* (e.g. CharacteriseMeeting) are *tasks*, i.e. non-root leaf goals; rectangles (e.g., ParticipantsUseSystemCalendar) are *domain assumptions*.

In general, a CGM under given user assertions has many possible realizations. To distinguish among them, stakeholders may want to express *preferences* on the requirements to achieve, on the tasks to accomplish, and on elements and refinements to choose. The CGM-Tool provides various methods to express preferences, including:

- attribute *rewards and penalties* to nice-to-have requirements and tasks respectively, so that to maximize the former and minimize the latter; (E.g., satisfying LowCost gives a reward = 100, whilst satisfying CharacteriseMeeting gives a penalty = 15.)
- introduce *numerical attributes*, *constraints* and *objectives*; (E.g., the numerical attribute Cost not only can be used to set prerequisite constraints for requirements, like "(Cost < 100€)" for LowCost, but also can be set as objectives to minimize.)

The CGM-Tool provides many automated-reasoning functionalities on CGMs [4].

Search/enumerate minimum-penalty/maximum reward realizations. One can assert rewards to the desired requirements and set penalties of tasks, then the tool finds automatically the optimal realization(s).

Search/enumerate optimal realizations wrt. pre-defined/user-defined objectives. One can define objective functions $obj_1, ..., obj_k$ over goals, refinements and their numerical attributes; then the tool finds automatically realizations optimizing them.

The above functionalities can be combined in various ways. For instance, the realization of Figure 1 is the one returned by CGM-tool when asked to minimize lexicographically, in order, the difference Penalty-Reward, workTime, and cost. [2] They have been implemented by encoding the CGM and the objectives into an SMT($\mathcal{LRA}$) formula and a set of $\mathcal{LRA}$ objectives, which is fed to the OMT tool OPTIMATHSAT [7]. We refer the reader to [4] for a much more detailed description of CGMs and their automated reasoning functionalities.

3 Requirements Evolution and Evolution Requirements

Requirements Evolution. Constrained goal models may evolve in time: goals, requirements and assumptions can be added, removed, or simply modified; Boolean and SMT constraints may be added, removed, or modified as well; assumptions which were assumed true can be assumed false, or vice versa.

Some modifications *strengthen* the CGMs, in the sense that they reduce the set of candidate realizations. For instance, dropping one of the refinements of an element (if at least one is left) reduces the alternatives in realizations; adding source elements to a refinement makes it harder to satisfy; adding Boolean or SMT constraints, or making some such constraint strictly stronger, restricts the set of candidate solutions; changing the value of an assumption from true to false may drop some alternative solutions. Vice versa, some modifications *weaken* the CGMs, augmenting the set of candidate realizations: for instance, adding one of refinement to an element, dropping source elements to a refinement, dropping Boolean or SMT constraints, or making some such constraint

[2] A solution *optimizes lexicographically* an ordered list of objectives $\langle obj_1, obj_2, ...\rangle$ if it makes obj_1 optimum and, if more than one such solution exists, it makes also obj_2 optimum, ..., etc.

strictly weaker, changing the value of an assumption from false to true. In general, however, since in a CGM the goal and/or decomposition graph is a DAG and not a tree, and the and/or decomposition is augmented with relational edges and constraints, modifications may produce combinations of the above effects, possibly propagating unexpected side effects which are sometimes hard to predict.

We consider the CGM in Figure 1 (namely, $\mathcal{M}_1$) as our starting model, and we assume that for some reasons it has been modified into the CGM $\mathcal{M}_2$ of Figure 2 in [5] (see §1). $\mathcal{M}_2$ differs from $\mathcal{M}_1$ for the following modifications:

(a) two new tasks, SetSystemCalendar and ParticipantsFillSystemCalendar, are added to the sub-goal sources of the refinement R_{13};

(b) a new source task RegisterMeetingRoom is added to R_{17}, and the binding between R_{16} and R_{17} is removed; the refinement R_{18} of the goal BookRoom and its source task CancelLessImportantMeeting are removed;

(c) the alternative refinements R_8 and R_9 of ManageMeeting are also modified: two new internal goals ByUser and ByAgent are added and become the single source of the two refinements R_8 and R_9 respectively, and the two tasks ConfirmOccurrence and CancelMeeting become respectively the sources of two new refinements R_{21} and R_{22}, which are the alternative refinements of the goal ByUser; the new goal ByAgent is refined by the new refinement R_{23} with source task SendDecision.

Evolution Requirements. We consider the generic scenario in which a previous version of a CGM $\mathcal{M}_1$ with an available realization μ_1 is modified into a new CGM $\mathcal{M}_2$. As a consequence, μ_1 typically is no more a valid realization of $\mathcal{M}_2$. E.g., we notice that μ_1 in Figure 2 in [5] does not represent a valid realization of $\mathcal{M}_2$: not all source tasks of R_{13} are satisfied, BookRoom has no satisfied refinement, and the new goal ByUser and refinement R_{21} are not satisfied. It is thus necessary to produce a new realization μ_2 for $\mathcal{M}_2$.

In general, when one has a sequence $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_i, ...$ of CGMs and must produce a corresponding sequence $\mu_1, \mu_2, ..., \mu_i, ...$ of realizations, it is necessary to decide some criteria by which the realizations μ_i evolve in terms of the evolution of the CGMs $\mathcal{M}_i$. We call these criteria, *evolution requirements*. We describe some possible criteria.

Recomputing realizations. One possible evolution requirement is that of always having the "best" realization μ_i for each $\mathcal{M}_i$, according to some objective (or lexicographic combination of objectives). Let $\mathcal{M}_1$, $\mathcal{M}_2$, and μ_1 be as above. One possible choice for the user is to compute a new optimal realization μ_2 from scratch, using the same criteria used in computing μ_1 from $\mathcal{M}_1$. In general, however, it may be the case that the new realization μ_2 is very different from μ_1, which may displease the stakeholders.

We consider now the realization μ_1 of the CGM $\mathcal{M}_1$ highlighted in Figure 1 and the modified model $\mathcal{M}_2$ of Figure 2 in [5]. If we run CGM-Tool over $\mathcal{M}_2$ with the same optimization criteria as for μ_1 –i.e., minimize lexicographically, in order, the difference Penalty-Reward, workTime, and cost– we obtain a novel realization μ_2^{lex} (Figure 3 in [5]. The new realization μ_2^{lex} satisfies all the requirements (both "nice to have" and mandatory) except MinimalEffort. It includes the following tasks: CharateriseMeeting, EmailParticipants, GetRoomSuggestions, UseAvailableRoom, RegisterMeetingRoom, ScheduleManually, ConfirmOccurrence, GoodParticipation,

and MinimalConflicts, and it requires one domain assumption: LocalRoomAvailable. This realization was found automatically by our CGM-Tool in 0.059 seconds on an Apple MacBook Air laptop.

Unfortunately, μ_2^{lex} turns out to be extremely different from μ_1. This is due to the fact that the novel tasks SetSystemCalendar and ParticipantsFillSystemCalendar raise significantly the penalty for R_{13} and thus for R_2; hence, in terms of the Penalty-Reward objective, it is now better to choose R_{10} and R_6 instead of R_2 and R_7, even though this forces ByPerson to be satisfied, which is incompatible with CollectionEffort, so that MinimalEffort is no more achieved. Overall, for μ_2 we have $\mathsf{Penalty} - \mathsf{Reward} = -65$, $\mathsf{workTime} = 4h$ and $\mathsf{cost} = 0€$.

In many contexts, in particular if μ_1 is well-established or is already implemented, one may want to find a realization μ_2 of the modified CGM $\mathcal{M}_2$ which is as similar as possible to the previous realization $\mathcal{M}_1$. The suitable notion of "similarity", however, may depend on stakeholder's needs. In what follows, we discuss two notions of "similarity" from [2], *familiarity* and *change effort*, adapting and extending them to CGMs.

Maximizing familiarity. In our approach, in its simplest form, the *familiarity* of μ_2 wrt. μ_1 is given by the number of elements of interest which are common to $\mathcal{M}_1$ and $\mathcal{M}_2$ and which either are in both μ_1 and μ_2 or are out of both of them; this can be augmented also by the number of new elements in $\mathcal{M}_2$ of interest (e.g., tasks) which are denied. In a more sophisticate form, the contribution of each element of interest can be weighted by some numerical value (e.g., Penalty, cost, WorkTime,...).

For example, if we ask CGM-Tool to find a realization which maximizes our notion of familiarity, we obtain the novel realization μ_2^{fam} (Figure 4 in [5]). μ_2^{fam} satisfies all the requirements (both "nice to have" and mandatory ones), and includes the following tasks: CharacteriseMeeitng, SetSystemCalendar, ParticipantsFillSystemCalendar, CollectFromSystemCalendar, GetRoomSuggestions, UseAvailableRoom, RegisterMeetingRoom, ScheduleAutomatically, ConfirmOccurrence, GoodParticipation, MinimalConflicts, CollectionEffort, and MatchingEffort; μ_2^{fam} also requires two domain assumptions: ParticipantsUseSystemCalendar and LocalRoomAvailable.

Notice that all the tasks which are satisfied in μ_1 are satisfied also in μ_2^{fam}, and only the intermediate goal ByUser, the refinement R_{21} and the four tasks SetSystemCalendar, ParticipantsFillSystemCalendar, UseAvailableRoom, and RegisterMeetingRoom are added to μ_2^{fam}, three of which are newly-added tasks. Thus, on common elements, μ_2^{fam} and μ_1 differ only on the task UseAvailableRoom, which must be mandatorily be satisfied to complete the realization. Overall, wrt. μ_2^{lex}, we pay familiarity with some loss in the "quality" of the realization, since for μ_2^{fam} we have $\mathsf{Penalty} - \mathsf{Reward} = -50$, $\mathsf{workTime} = 3.5h$ and $\mathsf{cost} = 0€$. This realization was found automatically by our CGM-Tool in 0.067 seconds on an Apple MacBook Air laptop.

Minimizing change effort. In our approach, in its simplest form, the *change effort* of μ_2 wrt. μ_1 is given by the number of newly-satisfied tasks, i.e., the amount of the new tasks which are satisfied in μ_2 plus that of common tasks which were not satisfied in μ_1 but are satisfied in μ_2. In a more sophisticate form, the contribution of each task of interest can be weighted by some numerical value (e.g., Penalty, cost, WorkTime,...). Intuitively, since satisfying a task requires effort, this value considers the extra effort

required to implement μ_2. (Notice that tasks which pass from satisfied to denied do not reduce the effort, because we assume they have been implemented anyway.)

For example, if we ask CGM-Tool to find a realization which minimizes the number of newly-satisfied tasks, we obtain the realization μ_2^{eff} (Figure 5 in [5]). The realization satisfies all the requirements (both "nice to have" and mandatory), and includes the following tasks: CharacteriseMeeitng, SetSystemCalendar, ParticipantsFillSystemCalendar, CollectFromSystemCalendar, UsePartnerInstitutions, ScheduleAutomatically, ConfirmOccurrence, GoodParticipation, MinimalConflicts, CollectionEffort, and MatchingEffort; μ_2^{eff} also requires one domain assumption ParticipantsUseSystemCalendar.

Notice that, in order to minimize the number of new tasks needed to be achieved, in μ_2^{eff}, FindASuitableRoom is refined by R_3 instead of R_5. In fact, in order to achieve R_5, we would need to satisfy two extra tasks (UseAvailableRoom and RegisterMeetingRoom) wrt. μ_1, whilst for satisfying R_3 we only need to satisfy one task (UsePartnerInstitutions). Besides, two newly added tasks SetSystemCalendar and ParticipantsFillSystemCalendar are also included in μ_2^{eff}. Thus the total effort of evolving from μ_1 to μ_2^{eff} is to implement three new tasks. Overall, for μ_2^{eff} we have Penalty$-$Reward $= -50$, workTime $= 3.5h$ and cost $= 80$€. This realization was found automatically by our CGM-Tool in 0.085 seconds on an Apple MacBook Air laptop.

Combining familiarity or change effort with other objectives. In our approach, familiarity and change effort are numerical objectives like others, and as such they can be combined lexicographically with other objectives, so that stakeholders can decide which objectives to prioritize.

References

1. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.
2. N. A. Ernst, A. Borgida, J. Mylopoulos, and I. Jureta. Agile Requirements Evolution via Paraconsistent Reasoning. In J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza, editors, *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 382–397. Springer, 2012.
3. M. M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. In *Proceedings of the IEEE*, pages 1060–1076, Sept. 1980.
4. C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos. Multi object reasoning with constrained goal model. *CoRR*, abs/1601.07409, 2016. Under journal submission. Available as `http://arxiv.org/abs/1601.07409`.
5. C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos. Requirements evolution and evolution requirements with constrained goal models. *CoRR*, abs/1604.04716, 2016. Available as `http://arxiv.org/abs/1604.04716`.
6. R. Sebastiani and S. Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), March 2015.
7. R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Computer-Aided Verification, CAV*, volume 9206 of *LNCS*. Springer, 2015.
8. V. E. S. Souza. *Requirements-based Software System Adaptation*. Phd thesis, University of Trento, 2012.