Modeling and Reasoning on Requirements Evolution 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 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 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 a standard exemplar used in Requirement Engineering.

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 [4]. 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 [10] 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 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) [5]. 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, 8].

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.

The rest of the paper is structured as follows: §2 introduces the notion of CGM through a working example; §3 introduces the notion of evolution requirements and requirements evolution through our working example; §4 formalizes the problem of automatically handling CGM evolutions and evolution requirements for CGMs; §5 provides a brief overview of our tool implementing the presented approach; in §6 we draw some conclusions.

Some of the ideas described here were discussed at conceptual level in a non-technical short paper at Conceptual Modeling conference, ER'2016 [6]. A longer and more detailed version of this paper, which includes also a related work section, is available [7].

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($\mathcal{LRA}$))* [8] extends SMT($\mathcal{LRA}$) by searching solutions which optimize some $\mathcal{LRA}$ objective(s). Efficient OMT($\mathcal{LRA}$) solvers like OPTIMATHSAT [9] allow for handling formulas with thousands of Boolean and rational variables [8, 5].

A Working Example. We recall from [5] the main ideas of Constrained Goal Models (CGM's) and the main functionalities of our CGM-Tool through a meeting scheduling example (Figure 1), a standard exemplar used in Requirements Engineering [11, 3].

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*. 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.

Suppose we want to capture and analyze requirements for a software system that schedules meetings (see [11, 3]). 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 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*. Notice that CGMs can represent both functional requirements (e.g. ScheduleMeeting) and quality requirements (e.g. LowCost).

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. For example, to require that, as a prerequisite for FastSchedule, ScheduleManually and CallParticipants cannot be both satisfied, one can add the constraint "FastSchedule $\rightarrow \neg$(ScheduleManually $\wedge$ CallParticipants)".

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 "(cost $<$ 100€)" as a prerequisite constraint for

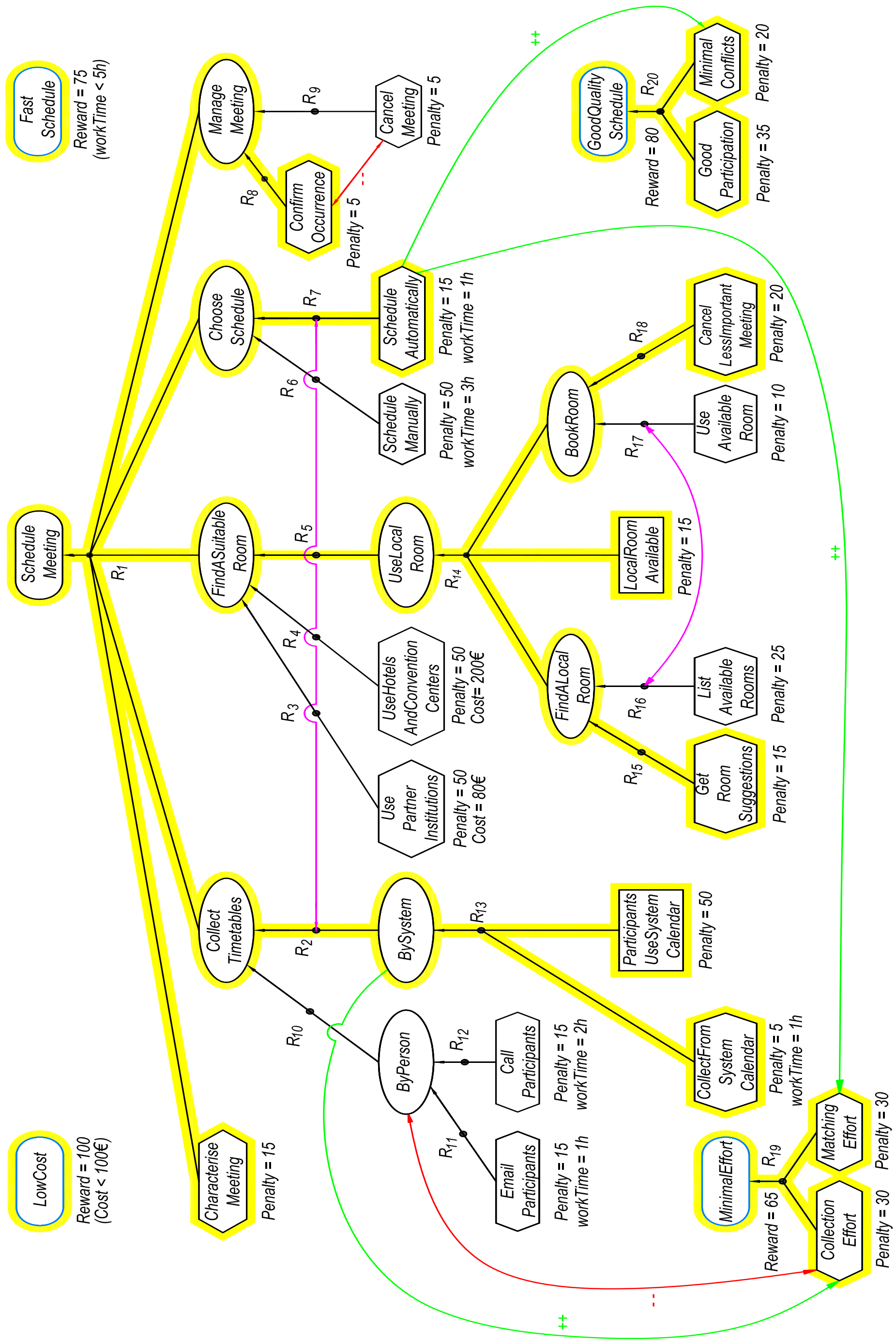

Fig. 1. A CGM $\mathcal{M}_1$, with a realization μ_1 minimizing lexicographically: the difference Penalty-Reward, workTime, and cost.

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.

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:

– 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 "($\mathsf{Cost} < 100€$)" for LowCost, but also can be set as objectives to minimize.)
– introduce a list of *binary preference relations* "$\succeq$"between elements or refinements. (E.g., one can set the preferences BySystem $\succeq$ ByPerson, UseLocalRoom $\succeq$ UsePartnerInstitutions and UseLocalRoom $\succeq$ UseHotelsAndConventionCenters.)

The CGM-Tool provides many automated-reasoning functionalities on CGMs [5].

Search/enumerate realizations. One can automatically check the realizability of a CGM– or to enumerate one or more of its possible realizations– under a group of user assertions and of user-defined constraints; (When a CGM is found un-realizable under a group of user assertions and of user-defined constraints, it highlights the subparts of the CGM and the subset of assertions causing the problem.)

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.

Search/enumerate optimal realizations wrt. binary preferences. Once the list of binary preference is set, the tool finds automatically realizations maximizing the number of fulfiled preferences.

The above functionalities can be combined in various ways. For instance, the realization of Figure 1 is the one returned by CGMtool when asked to minimize lexicographically, in order, the difference Penalty-Reward, workTime, and cost. [1] 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 [9]. We refer the reader to [5] for a much more detailed description of CGMs and their automated reasoning functionalities.

3 Requirements Evolution and Evolution Requirements

Here we show how a CGM can evolve, and how we can handle such evolution.

3.1 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 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 of a Schedule Meeting described 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 of Figure 2 (namely, $\mathcal{M}_2$). $\mathcal{M}_2$ 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}

[1] 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.

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.

3.2 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 of modifying a CGM $\mathcal{M}_1$ into a new version $\mathcal{M}_2$, μ_1 typically is no more a valid realization of $\mathcal{M}_2$. [2] E.g., we notice that μ_1 in Figure 2 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. 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} depicted in Figure 3. 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,

[2] More precisely, rather than "μ_1", here we should say "the restriction of μ_1 to the elements and variables which are still in $\mathcal{M}_2$." We will keep this distinction implicit in the rest of the paper.

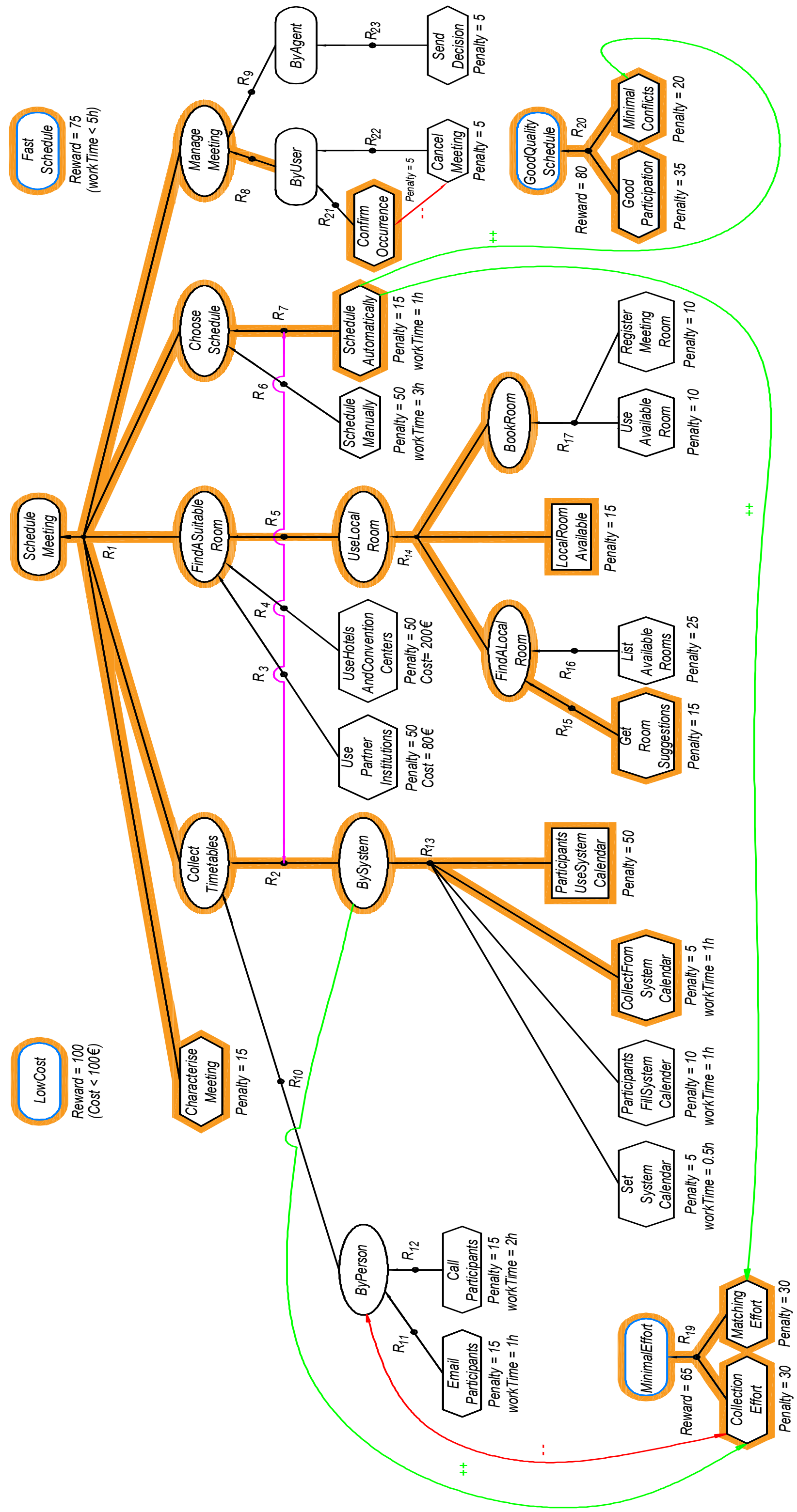

Fig. 2. The novel CGM $\mathcal{M}_2$, with the previous realization μ_1 highlighted for comparison. (Notice that μ_1 is no more a valid realization for $\mathcal{M}_2$.)

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,...). This is formalized in §4, and a functionality for maximizing familiarity is implemented in CGM-Tool.

For example, if we ask CGM-Tool to find a realization which maximizes our notion of familiarity (see §4), we obtain the novel realization μ_2^{fam} depicted in Figure 4. 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.) This is formalized in §4, and a functionality for minimizing change effort is implemented in CGM-Tool.

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} depicted in Figure 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 $\mathsf{Penalty} - \mathsf{Reward} = -50$, $\mathsf{workTime} = 3.5h$ and $\mathsf{cost} = 80$€. 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.

4 Automated Reasoning with Evolution Requirements

CGMs and realizations. We first recall some formal definitions from [5].

A *Constrained Goal Model (CGM)* is a tuple $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{B}, \mathcal{N}, \mathcal{D}, \Psi \rangle$, s.t.

- $\mathcal{B} \stackrel{\text{def}}{=} \mathcal{G} \cup \mathcal{R} \cup \mathcal{A}$ is a set of atomic propositions, where $\mathcal{G} \stackrel{\text{def}}{=} \{G_1, ..., G_N\}$, $\mathcal{R} \stackrel{\text{def}}{=} \{R_1, ..., R_K\}$, $\mathcal{A} \stackrel{\text{def}}{=} \{A_1, ..., A_M\}$ are respectively sets of goal, refinement and domain-assumption labels. We denote with $\mathcal{E}$ the set of element labels: $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{G} \cup \mathcal{A}$;
- $\mathcal{N}$ is a set of numerical variables in the rationals;
- $\mathcal{D}$ is an and-or directed acyclic graph of *elements* in $\mathcal{E}$ (or nodes) and *refinements* in $\mathcal{R}$ (and nodes);
- Ψ is a SMT($\mathcal{LRA}$) formula on $\mathcal{B}$ and $\mathcal{N}$, representing the conjunction of all relation edges, user-defined constraints and assertions.

The structure of a CGM is an and-or directed acyclic graph (DAG) of *elements*, as nodes, and *refinements*, as (grouped) edges, which are labeled by atomic propositions and can be augmented with arbitrary constraints in form of graphical relations and Boolean or SMT($\mathcal{LRA}$) formulas –typically conjunctions of smaller global and local constraints– on the element and refinement labels and on the numerical variables. Notice that each non-leaf element E is implicitly or-decomposed into the set of its incoming refinements $\{R_i\} \stackrel{\text{def}}{=} \text{RefinementsOf(E)}$ (i.e., $E \leftrightarrow (\bigvee_i R_i)$) and that each refinement R is and-decomposed into the set of its source elements $\{E_j\}$ (i.e., $R \leftrightarrow (\bigwedge_j E_j)$). Intuitively, a CGM describes a (possibly complex) combination of alternative ways of realizing a set of requirements in terms of a set of tasks, under certain domain assumptions.

Let $\mathcal{M} \stackrel{\text{def}}{=} \langle \mathcal{B}, \mathcal{N}, \mathcal{D}, \Psi \rangle$ be a CGM. A *realization* μ of $\mathcal{M}$ is an assignment of truth values to $\mathcal{B}$ and of rational values to $\mathcal{N}$ (aka, a $\mathcal{LRA}$-interpretation) which:

(*a*) for each non-leaf element E, μ satisfies $\left(E \leftrightarrow (\bigvee_{R_i \in \text{RefinementsOf(E)}} R_i)\right)$ –i.e., E is part of a realization μ if and only if one of its refinements is in μ;

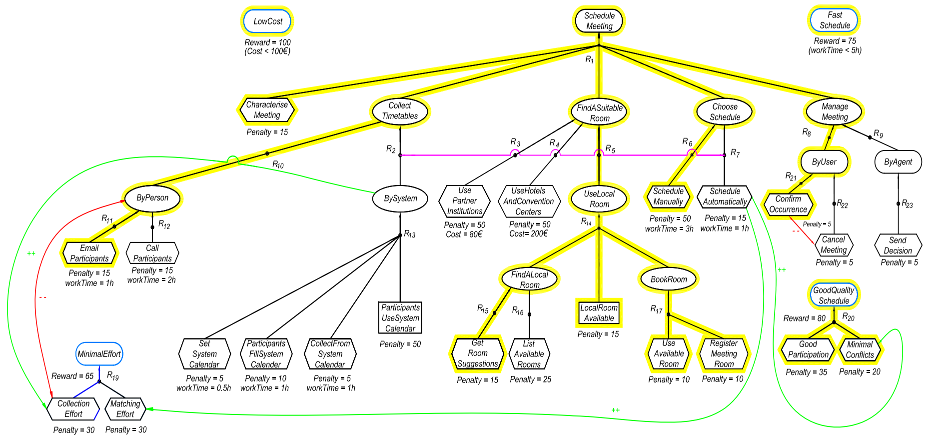

Fig. 3. New CGM $\mathcal{M}_2$, with realization μ_2^{lex} which minimizes lexicographically: the difference Penalty-Reward, workTime, and cost.

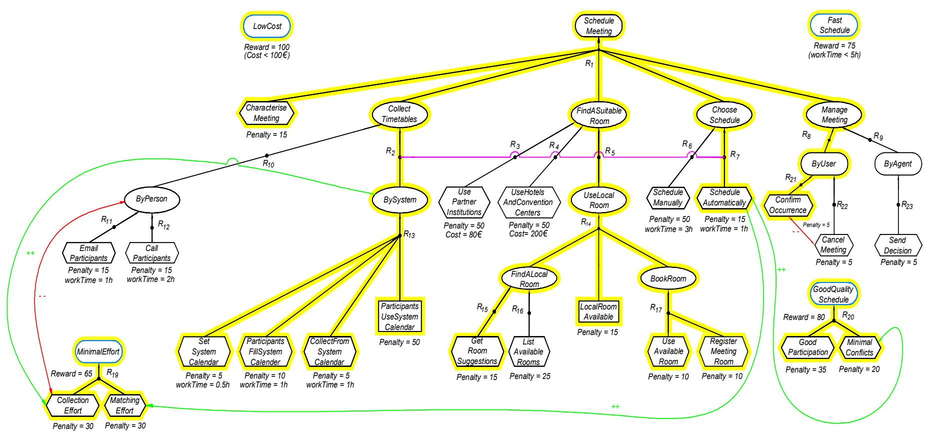

Fig. 4. New CGM $\mathcal{M}_2$, with realization μ_2^{fam} with maximizes the familiarity wrt. μ_1.

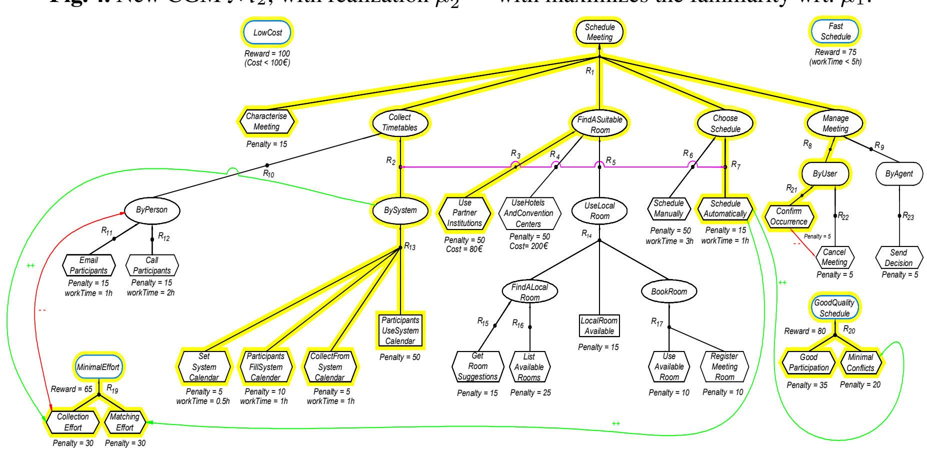

Fig. 5. New CGM $\mathcal{M}_2$, with realization μ_2^{eff} with minimimizes the change effort wrt. μ_1.

(b) for each refinement $\big(E_1, \ldots, E_n\big) \xrightarrow{R} E$, μ satisfies $((\bigwedge_{i=1}^{n} E_i) \leftrightarrow R)$ –i.e., R is part of μ iff and only if all of its sub-elements E_i are in μ;
(c) μ satisfies Ψ –i.e., the elements and refinements occurring in μ, and the values assigned by μ to the numerical attributes, comply with all the relation edges, the user-defined constraints and user assertions in Ψ.

We say that an element E or refinement R is *satisfied* [resp. *denied*] in μ if it is assigned to $\top$ [resp. $\bot$] by μ. μ is represented graphically as the sub-graph of $\mathcal{M}$ where all the denied element and refinement nodes are eliminated. We say that $\mathcal{M}$, including user assertions, is *realizable* if it has at least one realization, *unrealizable* otherwise.

As described in [5], a CGM $\mathcal{M}$ is encoded into a SMT$(\mathcal{LRA})$ formula $\Psi_{\mathcal{M}}$, and the user preferences into *numerical objective functions* $\{obj_1, ..., obj_k\}$, which are fed to the OMT solver OptiMathSAT, which returns optimal solutions wrt. $\{obj_1, ..., obj_k\}$, which are then converted back by CGM-tool into optimal realizations.

Evolution Requirements. Here we formalize the notions described in §3.2. Let $\mathcal{M}_1 \stackrel{\text{def}}{=} \langle \mathcal{B}_1, \mathcal{N}_1, \mathcal{D}_1, \Psi_1 \rangle$ be the original model, μ_1 be some realization of $\mathcal{M}_1$ and $\mathcal{M}_2 \stackrel{\text{def}}{=} \langle \mathcal{B}_2, \mathcal{N}_2, \mathcal{D}_2, \Psi_2 \rangle$ be a new version of $\mathcal{M}_1$. We look for a novel realization μ_2 for $\mathcal{M}_2$.

Stakeholders can select a subset of the elements, called *elements of interest*, on which to focus, which can be requirements, tasks, domain assumptions, and intermediate goals. (When not specified otherwise, we will assume by default that all elements are of interest.) Let $\mathcal{E}^* \subseteq \mathcal{E}_1 \cup \mathcal{E}_2$ be the subset of the elements of interest, and let $\mathcal{E}_1^* \stackrel{\text{def}}{=} \mathcal{E}^* \cap \mathcal{E}_1$ and $\mathcal{E}_2^* \stackrel{\text{def}}{=} \mathcal{E}^* \cap \mathcal{E}_2$ be the respective subsets of $\mathcal{M}_1$ and $\mathcal{M}_2$. We define $\mathcal{E}^*_{common} \stackrel{\text{def}}{=} \{E_i \in \mathcal{E}_2^* \cap \mathcal{E}_1^*\}$ as the set of elements of interest occurring in both $\mathcal{M}_1$ and $\mathcal{M}_2$, and $\mathcal{E}^*_{new} \stackrel{\text{def}}{=} \{E_i \in \mathcal{E}_2^* \setminus \mathcal{E}_1^*\}$ as the set of new elements of interest in $\mathcal{M}_2$.

Familiarity. In its simplest form, the cost of familiarity can be defined as follows:

$$\begin{aligned} \mathsf{FamiliarityCost}(\mu_2|\mu_1) \stackrel{\text{def}}{=} & \mid \{E_i \in \mathcal{E}^*_{common} \mid \mu_2(E_i) \neq \mu_1(E_i)\} \mid && (1) \\ & + \mid \{E_i \in \mathcal{E}^*_{new} \mid \mu_2(E_i) = \top\} \mid, && (2) \end{aligned}$$

where $\mid S \mid$ denotes the number of elements of a set S. $\mathsf{FamiliarityCost}(\mu_2|\mu_1)$ is the sum of two components:

(1) the number of common elements of interest (e.g., tasks) which were in μ_1 and are no more in μ_2, plus the number of these which were not in μ_1 and now are in μ_2,
(2) the number of new elements of interest which are in μ_2.

In a more sophisticate form, each element of interest E_i can be given some rational weight value w_i [3], so that the cost of familiarity can be defined as follows:

$$\begin{aligned} \mathsf{WeightFamiliarityCost}(\mu_2|\mu_1) \stackrel{\text{def}}{=} & \sum_{E_i \in \mathcal{E}^*_{common}} w_i \cdot \mathsf{Int}(\mu_2(E_i) \neq \mu_1(E_i)) && (3) \\ & + \sum_{E_i \in \mathcal{E}^*_{new}} w_i \cdot \mathsf{Int}(\mu_2(E_i) = \top), && (4) \end{aligned}$$

[3] Like Penalty, Cost and WorkTime in Figure 1.

where $\mathsf{Int}()$ converts true and false into the values 1 and 0 respectively.

Both forms are implemented in CGM-Tool. (Notice that (1) and (2), or even (3) and (4), can also be set as distinct objectives in CGM-Tool.) Consequently, a realization μ_2 maximizing familiarity is produced by invoking the OMT solver on the formula $\Psi_{\mathcal{M}_2}$ and the objective $\mathsf{FamiliarityCost}(\mu_2|\mu_1)$ or $\mathsf{WeightFamiliarityCost}(\mu_2|\mu_1)$ to minimize.

Change effort. We restrict the elements of interest to tasks only. In its simplest form, the change effort can be defined as follows:

$$\mathsf{ChangeEffort}(\mu_2|\mu_1) \stackrel{\text{def}}{=} |\ \{T_i \in \mathcal{E}^*_{common}\ |\ \mu_2(T_i) = \top, \text{ and } \mu_1(T_i) = \bot\}\ | \quad (5)$$
$$+ |\ \{T_i \in \mathcal{E}^*_{new}\ |\ \mu_2(T_i) = \top\}\ |\ . \quad (6)$$

$\mathsf{ChangeEffort}(\mu_2|\mu_1)$ is the sum of two components:

(5) is the number of common tasks which were not in μ_1 and which are now in μ_2,
(6) is the number of new tasks which are in μ_2.

As above, in a more sophisticate form, each task of interest T_i can be given some rational weight value w_i, so that the change effort can be defined as follows:

$$\mathsf{WeightChangeEffort}(\mu_2|\mu_1) \stackrel{\text{def}}{=} \sum_{T_i \in \mathcal{E}^*_{common}} w_i \cdot \mathsf{Int}(\mu_2(T_i) = \top) \cdot \mathsf{Int}(\mu_1(T_i) = \bot)$$
$$+ \sum_{T_i \in \mathcal{E}^*_{new}} w_i \cdot \mathsf{Int}(\mu_2(T_i) = \top).$$

Both forms are implemented in CGM-Tool. Consequently, a novel realization μ_2 minimizing change effort is produced by invoking the OMT solver on the formula $\Psi_{\mathcal{M}_2}$ and the objective $\mathsf{ChangeEffort}(\mu_2|\mu_1)$ or $\mathsf{WeightChangeEffort}(\mu_2|\mu_1)$.

Notice an important difference between (1) and (5), even if the former is restricted to tasks only: a task which is satisfied in μ_1 and is no more in μ_2 worsens the familiarity of μ_2 wrt. μ_1 (1), but it does not affect its change effort (5), because it does not require implementing one more task.

Comparison wrt. previous approaches. Importantly, Ernst et al. [2] proposed two similar notion of familiarity and change effort for (un-)constrained goal graphs:

familiarity: maximize (the cardinality of) the set of tasks used in the previous solution;
change effort: (i) minimize (the cardinality of) the set of new tasks in the novel realization –or, alternatively, (ii) minimize also the number of tasks.

We notice remarkable differences of our approach wrt. the one in [2].

First, our notion of familiarity presents the following novelties:

(i) it uses all kinds of elements, on stakeholders' demand, rather than only tasks;
(ii) it is (optionally) enriched also with (2);
(iii) (1) is sensitive also to tasks which were in the previous realization and which are not in the novel one, since we believe that also these elements affect familiarity.

Also, in our approach both familiarity and change effort allow for adding *weights* to tasks/elements, and to combine familiarity and change-effort objectives lexicographically with other user-defined objectives.

Second, unlike with [2], in which the optimization procedure is hardwired, we rely on logical encodings of novel objectives into OMT($\mathcal{LRA}$) objectives, using OPTIMATHSAT as workhorse reasoning engine. Therefore, new objectives require implementing no new reasoning procedure, only new OMT($\mathcal{LRA}$) encodings. For instance, we could easily implement also the notion of familiarity of [2] by asking OPTIMATHSAT to minimize the objective: $| \{T_i \in \mathcal{E}^*_{common} \mid \mu_2(T_i) = \bot,$ and $\mu_1(T_i) = \top\} |$.

Third, our approach deals with CGMs, which are very expressive formalisms, are enriched by Boolean and numerical constraints, and are supported by a tool (CGM-Tool) with efficient search functionalities for optimum realizations. These functionalities, which are enabled by state-of-the-art SMT and OMT technologies [8, 9], scale very well, up to thousands of elements, as shown in the empirical evaluation of [5]. In this paper we further enrich these functionalities so that to deal also with evolving CGMs and evolution requirements.

Fourth, unlike with [2], where realizations are intrinsically supposed to be *minimal*, in our approach minimality is an objective stakeholders can set and obtain as a byproduct of *minimum* solutions, but it is not mandatory. This fact is relevant when dealing with familiarity evolution requirements, because objective (1) can conflict with minimality, because it may force the presence of tasks from the previous solution which have become redundant in the new model. Thus, sometimes CGM-tool may return a non-minimal model if the stakeholder prioritizes familiarity above all other objectives.

5 Implementation

CGM-Tool provides support for modeling and reasoning on CGMs [5]. Technically, CGM-Tool is a standalone application written in Java and its core is based on Eclipse RCP engine. Under the hood, it encodes CGMs and invokes the OptiMathSAT [4] OMT solver [9] to support reasoning on CGMs. It is freely distributed for multiple platforms [5]. Currently CGM-Tool supports the functionalities in [5]:

Specification of projects: CGMs are created within the scope of project containers. A project contains a set of CGMs that can be used to generate reasoning sessions with OptiMathSAT (i.e., scenarios);

Diagrammatic modeling: the tool enables the creation of CGMs as diagrams; it provides real-time check for refinement cycles and reports invalid links;

Consistency/well-formedness check: CGM-Tool provides the ability to run consistency analysis and well-formedness checks on the CGMs;

Automated Reasoning: CGM-Tool provides the automated reasoning functionalities mentioned in section 2, and described in detail in [5].

With this work, we have enhanced CGM-Tool with the following functionalities:

[4] `http://optimathsat.disi.unitn.it`
[5] `http://www.cgm-tool.eu/`

Evolution Requirements Modelling and Automated Reasoning: by means of scenarios, stakeholders can generate *evolution sessions*, which allows for (i) defining the first model and finding the first optimal realization, (ii) modifying the model to obtain the new models, and (iii) generating automatically the "similar" realization (as discussed in section 3.2).

As a proof of concept, we have performed various attempts on variants of the CGM of §3. The automated generation of the realizations always required negligible amounts of CPU time, like those reported in §3.2.

6 Conclusions

We have proposed to model changing requirements in terms of changes to CGMs. Moreover, we have introduced a new class of requirements (evolution requirements) that impose constraints on allowable evolutions, such as minimizing (implementation) effort or maximizing (user) familiarity. We have demonstrated how to model such requirements in terms of CGMs and how to reason with them in order to find optimal evolutions.

Our future plans for this work include further evaluation with larger case studies, as well as further exploration for new kinds of evolution requirements that can guide software evolution.

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 *CAiSE*, volume 7328 of *LNCS*, pages 382–397, 2012.
3. M. S. Feather, S. Fickas, A. Finkelsteiin, and A. V. Lamsweerde. Requirements and specification exemplars. *Automated Software Engineering*, 4(4):419–438, Oct. 1997.
4. M. M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. In *Proceedings of the IEEE*, pages 1060–1076, Sept. 1980.
5. C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos. Multi-objective reasoning with constrained goal models. *Requirements Engineering Journal*, pages 1–37, 2016.
6. C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos. Requirements Evolution and Evolution Requirements with Constrained Goal Models. Short paper. In *Proceedings of the 37nd Int. Conference on Conceptual Modeling, ER'2016*, volume 9974 of *LNCS*. Springer, 2016. Available at `http://disi.unitn.it/rseba/papers/er16.pdf`.
7. C. M. Nguyen, R. Sebastiani, P. Giorgini, and J. Mylopoulos. Modeling and reasoning on requirements evolution with constrained goal models. Extended version of this paper. Available as `http://disi.unitn.it/rseba/papers/sefm17_extended.pdf`, 2017.
8. R. Sebastiani and S. Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), March 2015.
9. R. Sebastiani and P. Trentin. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Computer-Aided Verification, CAV*, volume 9206 of *LNCS*. Springer, 2015.
10. V. E. S. Souza. *Requirements-based Software System Adaptation*. Phd thesis, University of Trento, 2012.
11. A. van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learned. In *Proc. RE'95 - 2nd Int. Symposium on Requirements Engineering*, pages 194–203. IEEE, 1995.