Reasoning with Optional and Preferred Requirements

Neil A. Ernst[1], John Mylopoulos[1], Alex Borgida[2] and Ivan J. Jureta[3]

[1] Department of Computer Science
University of Toronto
{nernst,jm}@cs.toronto.edu
[2] Department of Computer Science
Rutgers University
borgida@cs.rutgers.edu
[3] FNRS & Information Management
University of Namur
ijureta@fundp.ac.be

Abstract. Of particular concern in requirements engineering is the selection of requirements to implement in the next release of a system. To that end, there has been recent work on multi-objective optimization and user-driven prioritization to support the analysis of requirements trade-offs. Such work has focused on simple, linear models of requirements; in this paper, we work with large models of interacting requirements. We present techniques for selecting sets of solutions to a requirements problem consisting of mandatory and optional goals, with preferences among them. To find solutions, we use a modified version of the framework from Sebastiani et al. [20] to label our requirements goal models. For our framework to apply to a problem, no numeric valuations are necessary, as the language is qualitative. We conclude by introducing a local search technique for navigating the exponential solution space. The algorithm is scalable and approximates the results of a naive algorithm.

1 Introduction

Requirements modeling languages such as SADT [19], KAOS [3], and i* [22] have been part of the very core of Requirements Engineering (RE) since the early days. Every requirements modeling language is grounded in an ontology consisting of a set of primitive concepts in terms of which requirements can be elicited, modelled, and analyzed. Traditionally, requirements were viewed as functions the system-to-be ought to support. This view is reflected in SADT and other structured analysis techniques of the 1970s and 1980s. Recently, an intentional perspective on requirements, Goal-Oriented Requirements Engineering (GORE), has gained ground. Requirements are now viewed as also containing stakeholder goals representing the intended purposes for the system-to-be. This deceptively small shift in the underlying ontology for requirements has had tremendous impact on RE research and is beginning to be felt in RE practice [12].

The objective of this paper is to propose an extension to the goal-oriented modelling and analysis framework in order to scalably accommodate priorities in the form of optional goals ("nice-to-have" requirements) and preferences ("requirement A is preferred over requirement B"). The extension is much needed to align RE theory with RE practice, where optionality and prioritization have been routinely used to manage requirements [17].

This paper makes the following contributions:

– An extension of the qualitative goal modeling framework of Sebastiani et al. [20], supporting stakeholder preferences and optional requirements.
– Macro operators for managing the scale of the possible problem and solution spaces.
– Algorithms to generate and compare all possible solutions.
– A local search algorithm to efficiently find solutions in a complex requirements model.
– A case study showing these concepts working on a problem at reasonable scale.

In this paper we make reference to the requirements problem. Our objective is to (efficiently) find solutions to a given requirements problem. According to [10], a solution to the requirements problem is a combination of tasks and domain assumptions. This combination must ensure that the execution of the tasks under the domain assumptions satisfies all mandatory goals, and zero or more optional goals. This paper looks at how this might be achieved.

We start by introducing the requirements modeling language that serves to define a requirements problem for a system of interest. The language is built on top of a formal logic. We begin by sorting its propositions and well-formed formulas (wffs), and then adding extra-logical constructs used to indicate two things: (i) which wffs are preferred over others; (ii) for which wffs is satisfaction optional to the satisfaction of mandatory wffs. Preference and optionality are extra-logical in the sense that they serve in algorithms but are not defined within the logic. We then study the implications of adding preference and optionality when searching for solutions to the requirements problems. We define formally what we mean by 'optional' goals below.

1.1 Case study: Enterprise strategy for portals

Figures 1 and 2 present a subset of the case study we use throughout the paper. We show how the model and the reasoning techniques can be used to model a real life setting concerning enterprise web portal development (ESP). We base our scenario on the documentation that the provincial Ministry of Government Services of Ontario (Canada) (MGS) provides, specifically on the description of the content management requirements. This is a series of reports from a 2003 project to re-work the provincial government's enterprise-wide citizen access portals, as part of an e-Government initiative. The content management requirements detail the specific technical aspects such a portal would have to meet[4]. Our extended

[4] available at http://www.mgs.gov.on.ca/en/IAndIT/158445.html

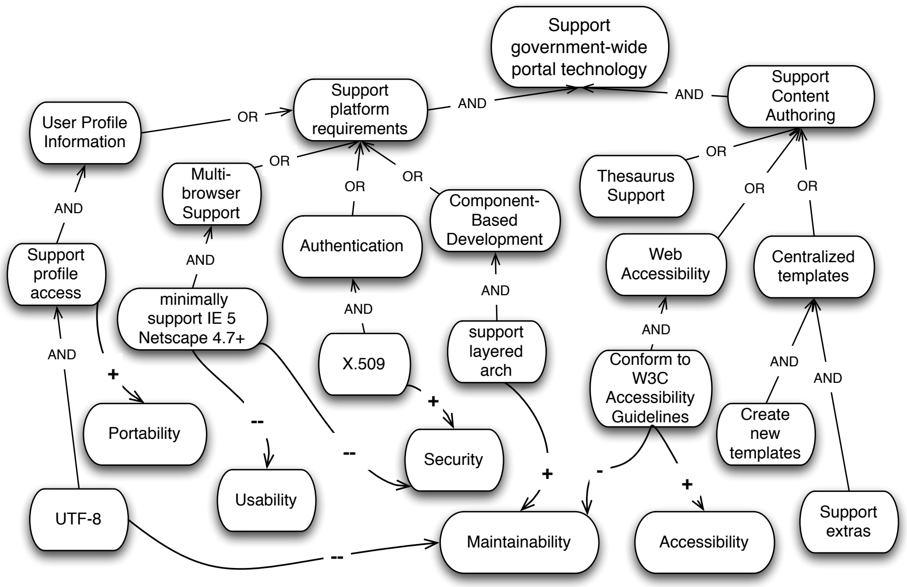

Fig. 1. A partial QGM model of the case study (1.1).

model contains 231 goals and several hundred relationships. This is the same size as real-world problems modeled with KAOS [13, p. 248].

2 pQGM: Extending qualitative goal models with preferences and optionality

Goal modeling is a widely accepted methodology for modeling requirements (cf. [12]). We leverage the qualitative goal modeling framework, which we call 'QGM', as defined in Sebastiani et al. [20]. That paper defines an axiomatization of goal models that transforms the goal labeling problem into a boolean satisfiability (SAT) problem (leveraging the power of off-the-shelf SAT solvers). The goal labeling problem, as stated in that paper, is "to know if there is a label assignment for leaf nodes of a goal graph that satisfies/denies all root goals [20, p. 21]". This approach can be used, among others, to find what top-level goals can be achieved by some set of leaf goals/tasks ("forward"), or conversely, given some top-level goals which are mandated to be True, what combination of tasks will achieve them ("backward"). The framework is however fully general, so that one can give an initial specification that indicates any set of goals as being mandated or denied, and solutions are consistent labelings of all the nodes in the model. Input goals serve as parameters for a given evaluation scenario. Thus a solution to the requirements problem amounts to answering the question: given the input goals, can we satisfy the mandatory goals?

We briefly recapitulate the definitions introduced in [6, 20]. We work with sets of goal nodes G_i, and relations $R_i \subseteq \wp G \times G$.

Relations have sorts *Decomposition*: {*and, or*}, which are (n+1)-ary relations; and *Contributions* {*+s,++s,+d,++d,-s,--s,-d,--d,++,+,--,-*}, which are binary. A goal graph is a pair $\langle \mathcal{G}, \mathcal{R} \rangle$ where $\mathcal{G}$ is a set of goal nodes and $\mathcal{R}$ is a set of goal relations.

Truth predicates are introduced, representing the degree of evidence for the satisfaction of a goal: $PS(G), FS(G), PD(G), FD(G)$, denoting (F)ull or (P)artial evidence of (S)atisfaction or (D)enial for G. A total order on satisfaction (resp. denial) is introduced as $FS(G) \geq PS(G) \geq \top$ (no evidence). This permits an axiomatization of this graphical model into propositional logic, so that the statement "A contributes some positive evidence to the satisfaction of B", represented $A \xmapsto{+s} B$ becomes the logical axiom $PS(a) \rightarrow PS(b)$. The complete axiomatization from which this section is derived is available in [20].

Extra-logical extensions. To manage preferences and optional elements, we add extra-logical elements to QGM. These elements do not affect the evaluation of admissible solutions (i.e., whether there is a satisfying assignment), thus our use of the term "extra-logical". We call our extension *prioritized* Qualitative Goal Models (pQGM). Models in this language form requirements nets (*r-nets*). pQGM *r-nets* allow us to capture stakeholder attitudes on elements in the model. Attitudes (i.e., emotions, feelings, and moods) are captured via optionality and preference relationships.

Optionality is an attribute of any concept indicating its optional or mandatory status. Being *mandatory* means that the element e in question must be fully satisfied in any solution (in the QGM formalization, $FS(e)$ must be true) and not partially denied ($PD(e)$ must be false). (In general, element e will be said to be "satisfied" by a solution/labeling iff $FS(e) \wedge \neg PD(e)$ is true.) Being *optional* means that, although a solution does not have to satisfy this goal in order to be acceptable, solutions that do satisfy it are more desirable than those which do not, all else being equal.

We consider non-functional requirements (NFRs) like *Usability* to be ideal candidates for being considered optional goals: if not achieved, the system still functions, but achievement is desirable if possible.

Alternatives arise when there is more than one possible solution of the problem (i.e., more than one possible satisfying assignment). In goal models with decomposition, this typically occurs when there is an OR-decomposition. We treat each branch of the OR-decomposition as a separate alternative solution to the requirements problem. Note that even for very small numbers of OR-decompositions one generates combinatorially many alternatives.

Preferences are binary relationships defined between individual elements in an *r-net*. A preference relationship compares concepts in terms of desirability. A preference from a goal to another goal in a pQGM *r-net* indicates that the former is strictly preferred to the latter. Preferences are used to select between alternatives.

We illustrate the role of these language elements in the sections which follow.

3 Finding requirements solutions

The purpose of creating a pQGM model is to use it to find solutions to the requirements problem it defines. We now define a procedure to identify these solutions. Briefly, we identify admissible models ('possible solutions'); satisfy as many optional requirements as we can; identify alternatives; then filter the set of alternatives using user-expressed preferences. We give both a naive and a local search algorithm for this process.

What are the questions that pQGM can answer? We want to know whether, for the mandatory goals we defined – typically the top-level goals as these are most general – there is some set of input nodes which can satisfy them. Furthermore, we want these sets of inputs to be strictly preferred to other possible sets, and include or result in as many options as possible. We show some answers to these questions in Section 4, below. With respect to the pQGM model shown in Fig. 2, a dominant solution (stippled nodes) consists of the goals *Support government-wide portal technology, Support Content Authoring, Web Accessibility, Conform to W3C Accessibility Guidelines, Support platform requirements, Authentication, X.509, User Profile Information, Support profile access, UTF-8, Accessibility, Security, Portability.* Goals such as *Minimally support IE5* are alternatives that are dominated, and therefore not included (in this case, because they break the *Usability* and *Security* options). Deriving this is the focus of the remainder of the paper.

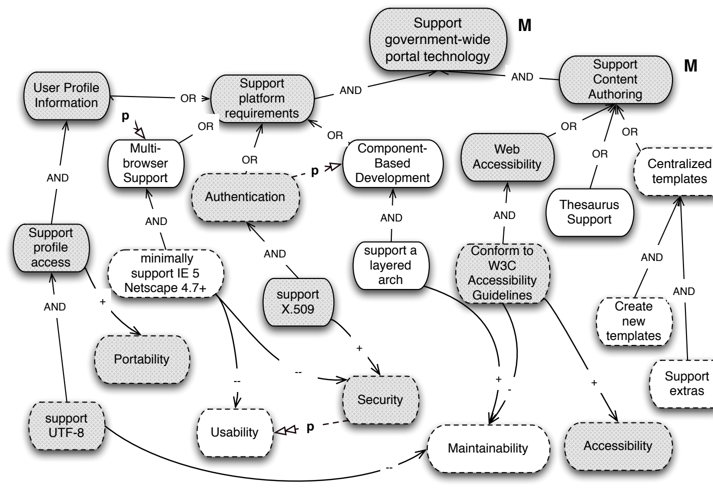

Fig. 2. An example pQGM *r-net*. Options have dashed outlines and mandatory elements are labeled 'M'. Preferences are represented with 'p' arcs on double-headed arrows. $\bar{R}$ consists of the elements outlined in solid borders and $\bar{S}$ the stippled elements (a candidate solution).

3.1 Identifying admissible models

Our first step is to find satisfying label assignments to the model elements. We rely on the solution to the backwards propagation problem of [20]. We take an attitude-free *r-net* $\bar{R}$ (*attitude-free* meaning one without optional elements, edges leading to/from them, and without considering preferences), and label the model. The labeling procedure encodes the user's desired output values (Φ_{outval}), the model configuration (Φ_{graph}) and the backwards search axiomatization $\Phi_{backward}$ (cf. [20, p. 8])[5]. The output of this is a satisfying truth assignment μ, or *None*, if there is no assignment that makes the mandatory nodes $FS \wedge \neg PD$.

If there was a satisfying assignment, we say our attitude-free model $\bar{R}$ is admissible, as our mandatory nodes were satisfied, and we call the resulting labeled goal model $\bar{R}_m$. At this point, it suffices to identify that there is admissibility. Subsequent stages handle alternatives.

3.2 Satisfying optional requirements

We now turn to consideration of the optional goals of our initial *r-net* R. In our diagrams, a node is marked optional by having dashed outlines. (This could of course be formalized by using a meta-predicate **O**.) Although a solution does not have to satisfy such a goal in order to be admissible, solutions that do satisfy it are more desirable than ones that do not, all else being equal. For example, in Fig. 2, we would prefer to satisfy all the dashed nodes (*Conform to W3C accessibility guidelines*, etc.). However, the use of optional goals allows us to accept solutions which only satisfy *some* of these nodes.

Consider the example in Fig. 2. In this example we have an NFR *Usability* and incoming concepts which can satisfy the goal. By specification, our algorithm should find the maximal sets of optional elements that can be added to the mandatory ones while preserving admissibility. It is important to note that in our framework optional elements can interact. This means that adding a optional goal to R_m could render a) the new model inadmissible b) previously admissible optional goals inadmissible. This reflects the notion that introducing an optional goal has consequences.

Option identification. We describe our naïve implementation in Algorithm 1. We are given an admissible r-net, R_m, e.g., a set of mandatory goals, along with a map, $\mathcal{T}$, from elements of R_m to the set of label assignments for R_m, e.g, $\{PD, PS, FD, FS\}$. We are given a set, $\mathcal{O}$, the set of optional elements, with $\mathcal{O} \cup R_m = R$. In our example, a subset of $\mathcal{O}$ is equal to the set of goals culminating in *Usability.*

For each subset *input* of $\mathcal{O}$ (i.e., element of $\wp(\mathcal{O})$), with the exception of the empty set, add it to $\bar{R}_m$, the admissible solution. We then use the SAT solver to find a satisfying assignment, checking for admissibility. If we find an admissible

[5] In the case of nodes e representing "hard" goals and especially tasks, one might require only truth assignments that satisfy axioms forcing binary "holds"/"does not hold" of e, thru axioms $FS(e) \rightarrow \neg FD(e)$ and $FS(e) \vee FN(e)$, forming additional sets $\Phi_{conflict}$ and $\Phi_{constrain}$ in [20].

```
**Input**: A solution $R_m$ and a set of optional goals, $\mathcal{O}$
**Output**: All maximal admissible option sets that can be added to $R_m$
$\mathcal{O}_m \leftarrow \emptyset$ **foreach** input $\in \wp(\mathcal{O})$ **do**
    // in a traversal of sets in non-increasing size
    $value \leftarrow admissible(input \cup R_m)$
    **if** value $==$ $\boldsymbol{T}$ **then**
        $\mathcal{O}_m \leftarrow input$
        Remove subsets of *input*
    **end**
**end**
**return** $\mathcal{O}_m$
```

Algorithm 1: NaiveSelect

solution (an option set O such that $O \cup R_m$ is admissible), we can add this to our set of acceptable solutions, $\mathcal{S}_a$. Now, because of the order in which we traverse subsets, this solution is a maximal set, and the proper subsets of O contain fewer optional goals. We therefore remove these sets from the collection being traversed.

The running time of this naive approach is clearly inpractical since in the worst-case, we must check all 2^n subsets of the set $\wp(\mathcal{O})$, where n is the number of optional elements. (The worst-case could arise if all optional goals invalidate all other optional goals.) This is on top of the complexity of each SAT test! Furthermore, the result set might be quite large. We therefore show some mechanisms to improve this.

Pruning the set of optional goals. We introduce two ways to reduce the number of option sets the naive algorithm finds, in order to reduce problem size. One approach prunes the initial sets of optional goals, and another reduces the admissible option set presented to the user. The intuition is to support a form of restriction on database queries; some requirements might be useful in some scenarios but not others, yet we would like to retain them in the original model. For example, we might implement the ESP model in an environment which does not need authentication.

Our first approach defines simple operations over optional goals – operators on $\wp(\mathcal{O})$ that act as constraints. Each element (a set) in $\wp(\mathcal{O})$ has two variables, cardinality and membership. We allow constraints on option set cardinality (boolean inequalities), and option set membership (inclusion or exclusion). An equivalent expression in SQL might be `DELETE FROM options WHERE Size $n` and `DELETE FROM options WHERE $x IN options`.

Our second approach is to make use of pQGM's preference relations to remove option sets which are dominated by other sets. In our example, with P =$\{$(*Authentication*, *Component-based development*), (*Security*, *Usability*)$\}$, if we found admissible solutions S_1 containing $\{$*Security*, *Authentication*$\}$, and S_2 containing $\{$(*Component-based development*)$\}$, we would discard S_2 in the case where both are admissible. This makes use of the Dominate function we define in Section 3.4.

```
Input: solution $R_m$, set of options $\mathcal{O}$, time limit tlim, tabu expiration expire
Output: A locally optimal set of optionsets $\mathcal{O}_t$
while  $time < tlim$ and $candidate \notin tabu\_list$ do
    $\mathcal{O}_t = \mathcal{O}_t + $ TabuMove($\emptyset$, $\mathcal{O}$, $\mathcal{O}_t$, tabu, time)
    if $time \ \%\ expire \leq 1$ then
        tabu $\leftarrow \emptyset$
    end
end
foreach $o', o'' \in \mathcal{O}_t$ do
    if $o' \subset o''$ then
        $\mathcal{O}_t$ - $o$
    end
end
```

Algorithm 2: TabuSearch

Section 4 shows the success of these techniques in reducing model evaluation times.

Local search. The naive approach must search through each member of the set of possible options, an exponential worst-case running time. This is clearly infeasible. A more tractable approach is to define a local search algorithm to find, in a bounded amount of time, a locally optimal solution. We implemented Tabu search [7] (Algorithms 2 and 3).

The algorithm iteratively searches for improvements to the cardinality of an admissible option-set. We start with a randomly chosen option, and add (random) remaining options singly, checking for admissibility. If we reach a point where no admissible options are found, we preserve the best result and randomly restart. A *tabu list* prevents the searcher from re-tracing steps to the same point it just found. A *tabu tenure* details for how many moves that point will be considered *tabu*. One commonly lists the last few moves as *tabu*. An iteration limit ensures the algorithm terminates. IsAdmissible represents a call to the SAT solver with the given optionset merged with the existing admissible solution, R. Although it is not necessarily the case that there are single options which are admissible, in practice this is common. If this isn't the case, our algorithm performs a random search, beginning with 1-sets of options.

3.3 Identify solution alternatives

For comparison purposes, our example model with 231 non-optional nodes and 236 relations is translated into a SAT expression consisting of 1762 CNF clauses. This is well within the limits of current SAT solvers.

The input into our penultimate phase consists of the admissible, labeled *r-net* R_m, along with a set, possibly empty, of option sets that can be joined to that *r-net*, $\mathcal{O}$. The number of current solutions, then, is the size of $\mathcal{O}$+1.

The goal of this phase is to identify alternatives that are created at disjunctions in the model. We do this by converting each admissible solution $s \in S$:

```
Input: candidate, remainder, solution, tabu, time
Output: solution
step = 0
while step < radius do
    tmp = candidate
    selected = Random.Choice(remainder)
    tmp = tmp + selected
    step = step + 1
    if selected ∉ tabu_list and tmp ∉ solution then
        break
    end
end
if  length(candidate) == initial then
    solution = solution + candidate
    return solution // base case
end
remainder = remainder - selected
candidate = candidate + selected
if time > tlim then
    return solution
end
time = time + 1
if IsAdmissible(candidate) then
    solution = solution + TabuMove(candidate, remainder)
end
else
    candidate = candidate - selected
    tabu_list = tabu_list + selected
    remainder = remainder + selected
    solution = solution + TabuMove(candidate, remainder)
end
return solution
```
Algorithm 3: TabuMove

$R_m \cup O \in \mathcal{O}$ to a boolean formula (a traversal of the AND-OR graph), and then converting this formula to conjunctive normal form. This is the accepted format for satisfiability checking (SAT). We pass this representation of the *r-net* as the SAT formula Φ to a SAT solver, store, then negate the resulting satisfiability model μ and add it as a conjunct to Φ. We repeat this process until the result is no longer satisfiable (enumerating all satisfying assignments). This produces a set of possible alternative solutions, e.g. $\mu_i, \mu_{i+1}, .., \mu_n$. We convert this to a set of sets of concepts that are solutions, $\mathcal{S}_a$.

3.4 Solution selection

Our final step is to prune the sets of solution *r-nets*, $\mathcal{S}_a$, using stakeholder valuations over individual elements – expressed as preferences and (possibly) costs. We do not prune before finding optional goals since we might discard a dominated set in favour of one that is ultimately inadmissible. Similarly, we do not risk the possibility of discarding an optionset that is nonetheless strictly preferred to another set, since by definition, if set O' is admissible and yet not selected, there was a set $O \supset O'$ that contains the same elements (and therefore preferences) and was admissible.

Selection using preferences. We will use in the text the notation pq if the r-net indicates that node p is preferred to node q, and let $\geq$ be the transitive reflexive closure of .

We use this to define the function *Dominate(M,N)*, mapping $\mathcal{S}\ x\ \mathcal{S}$ to booleans as follows:

$$Dominate(M, N) = True \iff \forall n \in N.\exists m \in M : m \geq n$$

Intuitively, every element of a dominated set is equal to a value in the dominant one, or is (transitively) less preferred. Note that the *Dominates* relation is a partial order, and we can therefore choose maximal elements according to it.

Selection using cost. Although not shown in the case study, our framework provides for solution selection using a simple cost operator, whenever such numbers are available. Clearly there are many cost dimensions to consider. For a given cost function, we define a *min_cost(value, increment,set)* function which ranks the proposed solutions using the cost of the solution as a total ordering. The meaning of *value* is as an upper cost threshold ('return all solutions below *value*'), and *increment* as a relaxation step in the case where no solutions fall under the threshold. Note that we are ranking admissible solutions, and not just requirements.

4 Evaluation

We now present experimental results of our technique on our large goal model presented in Section 1.1. Our ESP model contains 231 non-optional nodes and 236 relations.

We demonstrate the utility and scalability of our technique by presenting evaluation results for this model in various configurations of options, using different pre-processing steps to reduce the problem scale (Table 1). The first row of results reflects the raw time it takes to evaluate this particular model with no options (using the SAT solver). The third column shows that adding the set cardinality heuristic (a call to Naive with a maximal set size of 8 and minimal size of 5), results in some improvement in running times. For example, with 15 options, the running time is approximately 30% shorter (while returning the same options, in this case). Similarly, while we note the exponential increase in evaluation time for our naive algorithm, TabuSearch clearly follows a linear trend. However, the tradeoff is that TabuSearch misses some of the solutions, although in our tests, assuming an average-case model configuration, it found half of the maximal sets of options.

Table 1. Comparing naïve vs. heuristic option selection. The fifth column represents the number of calls to the SAT solver; the last column indicates how many solutions the local search found: *max*, the number of maximal sets, *sub* the remaining non-optimal sets. We used a time step limit of 400.

Options	**Naive**	**Naive(8,5)**	**TabuSearch**	# calls	solns
0	0.062 s	–	–	–	–
4	0.99	-	0.08s	2800	all
6	3.97	–	0.11	2800	1 max, 1 sub
9	31.8	23.6s	0.14	2942	1 max, 2 sub
12	4m23s	3m6s	0.16	3050	1 max, 2 sub
15	33m	21m	0.18	3165	1 max, 2 sub
20	-	-	0.19	3430	1 max, 2 sub

Quality of solutions. While the naive approach returns all solutions (the Pareto-front of non-dominated solutions) the Tabu search heuristic will only return an approximation of this frontier. TabuSearch returned, in the case with 15 options, one maximal set (of two), and 2 subsets. This in turn affected the number of alternatives that were found. Using the naive approach with 15 options, we identified 7 dominant solutions and 8 other alternatives (unrelated via preferences). Using TabuSearch, with the smaller option sets, we only return 4 dominant solutions and 6 other alternatives. These are individual solutions; we permit combinations, so the total number is much higher (the powerset). However, we feel the greatly reduced running time will allow for more model exploration than the naive approach.

5 Related work

We focus our comparison on the process of deriving a high-level solution from an initial model.

Tropos [2] uses the syntax of i* to generate early requirements models that are then the source for eventual derivation of a multi-agent system design. The 'early requirements' phase generates goal models using the notions of decomposition, means-ends and contribution links. These three notions bear no semantic distinction in the eventual reasoning procedure: they all propagate truth values. Partial satisfaction (denial) of stakeholder goals is modeled with the qualitative 'positive contribution' (denial) links, allowing goals to be partially satisfied (denied). While this is used to model preference, it is not clear what to make of a partially satisfied goal (with respect to preference). Although formalized (in [20] and others), the reasoning still relies on stakeholder intervention to evaluate how 'well' a solution satisfies the qualities (e.g., is it preferable that security is partially satisfied while usability is partially denied?). From [2, p. 226]: "These [non-deterministic decision points] are the points where the designers of the software system will use their creative [sic] in designing the system-to-be."

KAOS [3] uses a graphical syntax backed by temporal logic. KAOS has a strong methodological bent, focusing on the transition from acquisition to specification. It is goal-oriented, and describes top-down decomposition into operationalized requirements which are assigned to actors in the system-to-be. Alternative designs can be evaluated using OR decompositions, as in Tropos. In [16], a quantitative, probabilistic framework is introduced to assign *partial* satisfaction levels to goals in KAOS, which allows tradeoffs to be analyzed, provided the quantification is accurate. The notion of gauge variables is hinted at in [14], which seem to be measures attached to goal achievement. Obstacle analysis [15] provides a mechanism to identify risks to normal system design.

There are several techniques that can be used to generate pairwise preferences over lists of requirements, surveyed in [11, 8]. These included iterative pairwise comparisons, economic valuations, and planning games. Here, all requirements are assumed to be achievable and there are no interactions between them. This is a very simple model of requirements. For example, what if one requirement is blocked if another is implemented? Or if there is a trade-off analysis required?

The QGM implementation in [20] describes a variant using a minweight SAT solver; weights are assigned using qualitative labels. It searches for a solution that minimizes the number/cost of input goals (in essence, the tasks a solution must accomplish). Our approach doesn't consider weights, relying on the user's judgement to evaluate the optimal solution they prefer. As mentioned, it is simple to add cost as a factor in our solution finding, but we don't assume that this is available.

Finally, researchers have pointed out that assuming stakeholders understand the cost of a given requirement is dangerous [16]. Arguably it is equally dangerous to assume prioritization is possible; our approach assumes that such preferences will have to be iteratively provided, as stakeholders are presented with various solutions (or no solutions).

Search-based SE focuses on combinatorial optimization, using evolutionary and local search algorithms to efficiently find solutions, e.g [1, 5]. DDP [4] supports quantitative reasoning and design selection over non-hierarchical requirements models, and the latest iteration uses search-based heuristics to find system designs [9]. The principal difference is in the nature of the underlying model. Our framework uses goal models to provide for latent interactions between requirements that are satisfied.

An alternative to formalized requirements models are workshop techniques (e.g., [18]). Here, requirements problems and solutions can be explored and evaluated without the use of formal reasoning, which can be difficult to construct and display to end users. The motivation for formalization is that it is more amenable to rapid prototyping and semi-autonomous execution, particularly in the context of evolving systems. Workshops are expensive and difficult to organize. A good formal methodology will ensure end users don't use the model, but merely get the results, in this case the set of optimal solutions.

6 Conclusions and Future work

With pQGM, we have introduced an extension to a well-known formal goal reasoning procedure. Our extension allowed us to compare solutions to the requirements problem using preferences and optional goals. We described some techniques for generating solution alternatives and comparing them, and showed that our techniques can scale with the addition of simple constraints. We view the modeling of requirements problems in pQGM as iterative: identify a core set of *mandatory* elements and verify admissibility; identify *optional* elements; generate solutions; refine the model to narrow the solution space (using preferences or costs).

We hope to extend this work to accommodate incremental revisions of the set of solutions as new information becomes available in the model. Finally, we would like to highlight the enduring problem of propagation of conflicting values. Although separate in QGM, many goals are only partially satisfied and often also partially denied. We have been working on a paraconsistent requirements framework that addresses this issue.

Most modeling languages are focused on finding single solutions to the problem. They use priorities and evaluation to find that solution. We have defined some ways to select from many solutions. The model assessment procedure defined a) ways to narrow the search space and b) ways to select between solutions once found. In other modeling languages, such as KAOS [3] or Tropos [2], the aim of the methodology is to generate a single solution which best solves the problem.

Why do we want to model multiple solutions? First, we think it is unrealistic to expect to find a single solution. Many practitioners, particularly from the lean and agile paradigm, prefer to wait until the 'last responsible moment' before making decisions [21]. Secondly, in a changing system, we should not presume to know what single solution will always apply. Finally, this method allows the user – with appropriate tool and language support – to define the conditions under which a solution is acceptable.

References

1. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization, Operations research/Computer Science Interfaces, vol. 45. Springer Verlag (2008)
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8, 203–236 (May 2004)
3. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1-2), 3–50 (Apr 1993)
4. Feather, M.S., Cornford, S.: Quantitative risk-based requirements reasoning. Requirements Engineering J. 8, 248–265 (Nov 2003)
5. Finkelstein, A., Harman, M., Mansouri, S., Ren, J., Zhang, Y.: A search based approach to fairness analysis in requirement assignments to aid negotiation, mediation and decision making. Requirements Engineering J. 14(4), 231–245 (Dec 2009)

6. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. Journal on Data Semantics 2800, 1 – 20 (2003)
7. Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 13(5) (1986)
8. Herrmann, A., Daneva, M.: Requirements Prioritization Based on Benefit and Cost Prediction : An Agenda for Future Research. In: Intl. Conf. Requirements Engineering. pp. 125–134. Barcelona (Sep 2008)
9. Jalali, O., Menzies, T., Feather, M.S.: Optimizing requirements decisions with KEYS. In: International Workshop on Predictor Models in Software Engineering. pp. 79–86. Leipzig, Germany (2008)
10. Jureta, I.J., Mylopoulos, J., Faulkner, S.: Revisiting the Core Ontology and Problem in Requirements Engineering. In: Intl. Conf. Requirements Engineering. pp. 71–80. Barcelona (Sep 2008)
11. Karlsson, L., Höst, M., Regnell, B.: Evaluating the practical use of different measurement scales in requirements prioritisation. In: Intl. Conf. Empirical Software Engineering. pp. 326–335. Rio de Janeiro, Brasil (2006)
12. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Intl. Conf. Requirements Engineering. pp. 249–263. Toronto (2001)
13. van Lamsweerde, A.: Requirements engineering: from craft to discipline. In: Intl. Conf. Foundations of Software Engineering. pp. 238–249. Atlanta, Georgia (Nov 2008)
14. van Lamsweerde, A.: Reasoning About Alternative Requirements Options. In: Borgida, A., Chaudhri, V.K., Giorgini, P., Yu, E.S.K. (eds.) Conceptual Modeling: Foundations and Applications, Lecture Notes in Computer Science, vol. 5600, pp. 380–397. Springer (2009)
15. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. Trans. Soft. Eng. 26, 978–1005 (2000)
16. Letier, E., van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. In: Intl. Conf. Foundations of Software Engineering. pp. 53–62. Newport Beach, CA (2004)
17. Liaskos, S., Mcilraith, S.A., Mylopoulos, J.: Goal-based Preference Specification for Requirements Engineering. In: Intl. Conf. Requirements Engineering. Sydney (Sep 2010)
18. Maiden, N., Robertson, S.: Integrating Creativity into Requirements Processes: Experiences with an Air Traffic Management System. In: Intl. Conf. Requirements Engineering. Paris, France (2005)
19. Ross, D.: Structured Analysis (SA): A Language for Communicating Ideas. Trans. Soft. Eng. 3(1), 16–34 (1977)
20. Sebastiani, R., Giorgini, P., Mylopoulos, J.: Simple and Minimum-Cost Satisfiability for Goal Models. In: Intl. Conf. Advanced Informations Systems Engineering. pp. 20–35. Riga, Latvia (Jun 2004)
21. Thimbleby, H.: Delaying commitment. IEEE Software 5(3), 78–86 (1988)
22. Yu, E.S.: Towards modelling and reasoning support for early-phase requirements engineering. In: Intl. Conf. Requirements Engineering. pp. 226–235. Annapolis, Maryland (1997)