Pragmatic Requirements for Adaptive Systems: a Goal-Driven Modelling and Analysis Approach

ABSTRACT

Goal models (GM) have been used in adaptive systems engineering for their ability to capture the different ways to fulfill the requirements and contextual GM extends these models with the notion of context and context-dependent applicability of goals. In this paper, we observe that the interpretation of a goal achievement is itself context-dependent. Thus, we introduce the notion of Pragmatic Goals which have a dynamic satisfaction criteria. We also developed and evaluated an automated framework to validate the Pragmatic CGM. Finally, we performed an experiment to compare the algorithm to human judgment and concluded that the specification of context-dependent goals' applicability and interpretations makes it hard for domain stakeholders to decide whether the model covers all possibilities, both in terms of time and accuracy, thus showing the importance and contribution of our framework.

Categories and Subject Descriptors

D.2.1 [**Software**]: Software Engineering Requirements

General Terms

Design, Languages, Algorithms

Keywords

Requirements Engineering, Goal Model, Adaptation

Algorithm 1 implements the `Refinement` entity's *isAchievable* method from Figure 1. It correlates three context-dependent aspects from the model: (1) the applicable goals, tasks and delegations; (2) the goals' interpretations and; (3) the delivered quality level provided by the tasks.

1. ALGORITHM DESCRIPTION

The algorithm aims to decide whether the root goal is achievable and, if so, which execution plan(s), *i.e.*, the set of all tasks to be executed, is likely to achieve it under the desired quality constraints. The algorithm is recursive with

Algorithm 1: isAchievable(CGM cgm, Context current, QualityConstraint qualReq)

```
Input: CGM, current context and desired QCs
Result: An execution plan, if possible
1  Goal root ← cgm.getRoot();
2  if !root.isApplicable(current) then
3      return NULL;
4  if (root.myType() == task) then
5      if (root.canFulfill(qualReq)) then
6          return new Plan(root);
7      else
8          return NULL;
9  QualityConstraint consideredQualReq;
10 consideredQualReq ←
   root.interp.stricterQualityConstraint(root.qualReq, qualReq);
11 Plan complete ← NULL;
12 deps ← root.getRefinements(cgm, curContext);
13 foreach Refinement d in deps do
14     Plan p ← d.isAchievable(cgm, context,
       consideredQualReq);
15     if (p != NULL) then
16         if root.isOrDecomposition() then
17             return p;
18         if (root.isAndDecomposition()) then
19             complete ← addPlanToPlan(p, complete);
20     else
21         if (root.isAndDecomposition()) then
22             return NULL;
23 return complete
```

a linear complexity with respect to the number of nodes in the CGM, building on the fact that the CGM is a tree-structured model without loops.

The algorithm considers the root node of the CGM (line 1) and checks whether the root goal is itself applicable under the current context (line 2), returning NULL if it is not (line 3). In the particular case when the variant's root node is a task (line 4) it can readily decide on the achievability. This is because the task nodes knows the expected QoS it can deliver for each metric under the context considered in the CGM. By comparing the delivered QoS and required QCs (line 5), the node can decide whether it is capable or not of delivering such QCs. If it can, it will return a plan consisting only of this task (line 6), otherwise it will return `NULL` (line 8) and indicate its inability to fulfill the goal's interpretation.

If the root is not a Task, the algorithm will define its quality requirement as the stricter `Quality Constraints` between its own and the QCs passed on as parameters (line 10) and begin laying out an execution plan to fulfill such QCs (line 11). For each of the applicable refinements, it will evaluate if it is achievable (line 14). If the refinement is achievable then, for OR-decompositions, the algorithm returns this plan immediately (lines 16 - 17) and for AND-decompositions it is added to the *complete* plan (lines 18-19). Otherwise, if the refinement is unachievable it will immediately return NULL for AND-decompositions (line 22). Finally, for AND-decompositions, should all refinements are achievable it will return the *complete* plan (line 23).

As an outcome, an execution plan is returned for achievable goals and for unachievable goals the NULL value is returned to indicate the inability of fulfilling the required constraints allowing for alternate means of achieving higher level goals to be explored.

2. ALGORITHM COMPLEXITY

The Algorithm 1 runs in time $O(n)$ where n is the number of nodes in the CGM. We will prove this using finite induction during the following paragraphs over the height h of the CGM tree.

Induction basis: Single task as CGM root (tree height $h = 0$).

The hypothesis is then well defined. We assume that the algorithm runs in linear time with regard to the number of nodes in the CGM. Our induction will be made on the height of the CGM tree. Our basis is the trivial case where the CGM has height 0. In this particular case, the root node is a task - due to the CGM restriction that states that any leaf node is a task. If we apply the algorithm in this scenario, the IF-clause at line 5 will be triggered and the algorithm will make a simple comparison of the quality constraints passed as input and the quality that the task can provide. If this task cannot fulfil the quality required then it returns NULL. This is done at constant time. If, otherwise, the task is able to deliver the quality level required, it then creates a plan p and includes itself in it. This is also done in constant time.

One particular case that must also be taken into consideration its when the root goal is not applicable. In this situation the algorithm simply returns `NULL` in constant time as well.

Induction step.

Assume that we have proven that:

Definition 1. For any tree of height h' where $h \geq h' \geq 0$ the algorithm runs in time $O(n)$ where n is the number of nodes in the CGM tree.

Let's now prove that the algorithm will also be executed in $O(n)$ for a tree of height $h + 1$.

Since the tree has a height greater than 1 ($h \geq 0$), the IF-clause at line 5 will not be triggered. So, the next step in the algorithm will be to define the **`consideredQualReq`**, which is simply to compare the quality requirement and choose the stricter one. This is done in constant time. Then the algorithm checks whether the decomposition is an OR-decomposition or an AND-decomposition. It extracts from the CGM the node's Refinements which is done in $O(m)$ time - where m is the number of Refinements of the root goal.

Finally, for each Refinement d in $refinements$ the algorithm recursively invokes itself for the CGM of the Refinement (sub goal, task or delegation). Since this invocation is performed on trees of height lower or equal to h we can use the fact that the root node has m refinements and Definition 1 to state that this is performed in $O(n_k)$ time, where n_k is the amount of nodes in the k sub tree.

The elapsed time for the whole `for each` can be calculated by Equation 1.

$$Time_{\texttt{foreach}} = \sum_{k=1}^{m} O(n_k) \tag{1}$$

Since n is the amount of nodes in the whole tree (all subtrees plus the root goal node ($n = n_1 + n_2 + \ldots + n_k + 1$) and $O(n_1) + O(n_2) + \ldots + O(n_k) = O(n_1 + n_2 + \ldots + n_k)$, we can conclude that for a tree of height h, the algorithm is executed in time $O(n)$ where n is the number of nodes in the tree. Thus proving the hypothesis that for any tree of height $h \geq 0$, the algorithm runs in linear time with regard to the number of nodes in the CGM.