Task Specification and Reasoning in Dynamically Altered Contexts

George Chatzikonstantinou, Michael Athanasopoulos, and Kostas Kontogiannis

National Technical University of Athens, Greece
{athanm,kkontog}@softlab.ntua.gr, gchatzik@cslab.ece.ntua.gr

Abstract. Software systems are prone to evolution in order to be kept operational and meet new requirements. However, for large systems such evolution activities cannot occur in a vacuum. Instead, specific action plans must be devised so that evolution goals can be achieved within an acceptable level of deviation or, risk. In this paper we present an approach that allows for the identification of plans in the form of actions that satisfy a goal model when the environment is constantly changing. The approach is based on sequences of mutations of an initial solution, using a local search algorithm. Experimental results indicate that even for medium size models, the approach outperforms in execution time the weighted Max-Sat algorithms, while it is able to achieve an almost optimal solution. The approach is demonstrated on an example scenario of re-configuring a dynamically provisioned system.

Keywords: local search, weighted partial max-sat, task specification.

1 Introduction

Complex software systems are prone to continuous change and re-configuration. Software maintenance, hardware upgrades, and dynamic provision of resources in elastic or autonomic systems, are just a few of the factors that drive the need for designing systems that assist administrators to compile action plans. In this context, the focus is to devise *a*) models that represent system goals; *b*) models that associate such goals with tasks and actions ; and *c*) reasoning methodologies that allow for the selection of tasks and actions in order to form coherent plans. The software engineering community has responded with models to represent system-wide functional and non-functional properties as well as, formalisms to associate such functional properties with design decisions, tasks, actions, and stakeholder views. These models include i* [1], KAOS [2,3], the Goal-oriented Requirements Language (GRL) [4], the Extended Enterprise Modeling Language (EEML), and the Unified Modeling Language, to name a few. Similarly, reasoning on these models has emerged as a key problem in order for useful logical and sound conclusions to be reached. Such reasoning approaches are based on logic deductions (rules), propagation of labels, domain specific heuristics or, SAT solvers. SAT solvers in particular have been a focal point of the research conducted in this area. However, there are still open issues to be investigated when

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 625–639, 2014.

the goal models themselves are modified as a result of changes in the operating environment or, as a result of the actions incurred so far.

In this paper, we investigate the use of local search algorithms and boolean expression evaluators to reason with Decision Task Models introduced in Section 3, when labels related to cost and benefit values, for actions and tasks, are altered as a result of context changes. Experimental results indicate that the approach allows for obtaining a solution that is within 90% range of the optimal value at a fraction of time that is required by a Weighted Partial MAX-SAT algorithm to compute the optimal solution for the problem at hand. Applications of such reasoning include the formation of plans to re-configure autonomic systems, plan for software and hardware upgrades in a large scale where the administrators must conform to specific guidelines (e.g. ITIL), or devise alternative plans to meet specific goals and requirements. We illustrate the approach by a running example depicted in Fig. 2, that focuses on a goal model that denotes how high quality of service can be maintained in an elastic cloud based system. The approach has been evaluated on a large number of sizeable goal models that have been compiled by an automated construction process with positive results.

The paper is organized as follows: Section 2 provides an outline of the proposed approach. Section 3 introduces the elements of Decision Task Models (DTMs) and discusses their semantics. In Section 4 a formalization for DTMs is provided along with a process for the transformations of DTMs to CNF formulas. Subsequently, in Section 5 two reasoning approaches are discussed based on operating environment for the task model and in Section 6 experiment results are discussed to provide an evaluation for the proposed framework and algorithms. Finally, related work is discussed in Section 7 and Section 8 concludes the paper.

2 Process Outline

The outline of the proposed framework process is depicted in Fig. 1. Initially, certain tasks and actions, along with the relationships that exist between them are modeled in a Decision Task Model (DTM) like the one presented in Fig. 2, which describes what tasks and actions can be performed in order to maintain the QoS to a required level. Given such a model, we are interested in determining the optimal (or at least, suboptimal) plan to accomplish the root task while the weights assigned to each node change dynamically as a consequence of contextual changes. For example, for the model of Fig. 2, the weights of A5 and T11 change from $C1$ and $B1$ to $C2$ and $B2$ respectively, leading to different optimal solutions as this is summarized in Tables 3 and 4. An impossible low negative weight value (i.e. high cost) deems an action unachievable, while a high positive weight value (i.e. high benefit) deems a task achievable.

The first step towards plan determination is to extract a CNF formula that fully captures the logical dependencies modeled by the DTM. This enables the reduction of plan determination to the SAT problem, and as a consequence allow the use of a SAT solver. Moreover, as nodes in DTMs are annotated with weights which may change as a consequence of certain context changes, we are interested

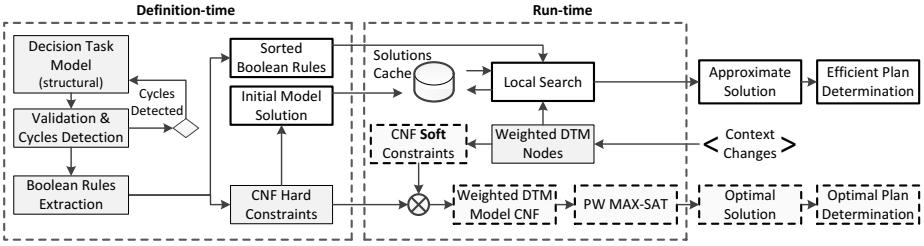

Fig. 1. The proposed framework for optimal and efficient plan determination

in finding an assignment that not only satisfies the CNF formula but also has the best score, in terms of node weights. This optimization problem, referred to as *optimal plan determination*, can be reduced to an instance of Weighted Partial Max-SAT problem, hence a Max-SAT solver can be utilized to solve it. The steps required for the determination of the optimal plan are depicted as dashed rectangles in Fig. 1.

However, as the context (and thus node weights) may change dynamically, computing an approximate solution may be a preferable alternative to optimal plan determination, as Max-SAT is a known NP-hard problem. In this paper, we propose the use of a local search algorithm as an effective technique for efficient plan determination. The steps required for the determination of an effective plan are depicted as bold rectangles in Fig. 1.

3 Decision Task Modeling

In order to model the actions that realize certain tasks, and to express semantic and temporal relationships that exist between tasks and actions, we propose a metamodel which borrows notions from the goal model theory. The proposed DTM metamodel, an instance of which is illustrated in Fig. 2, retains the AND/OR-decomposition schema used in goal models. However, additional modeling elements are used to capture logical and temporal relationships, increasing thus the expressiveness of the proposed notation. In the rest of this section we briefly describe the semantics of the various modeling artifacts.

Nodes: DTM nodes may be either *Tasks* or *Actions*. The former may be AND/OR decomposed and represent a collection of actions or subordinate tasks, where either more than one subtasks or actions should be combined together, or one or more subtasks or actions may be alternatively selected to accomplish the task. In contrast, action nodes represent atomic activities. In the example DTM of Fig. 2, tasks are depicted as ellipses and Actions as hexagons.

Links: The DTM metamodel contains three types of links between nodes namely, *Logical Preconditions* ($\xrightarrow{lp}$), *Temporal Preconditions* ($\xrightarrow{tp}$), and *Contributions*. The former two links interconnect task and action nodes and express temporal dependencies. More specifically, $\xrightarrow{lp}$ links resemble precedence links originally

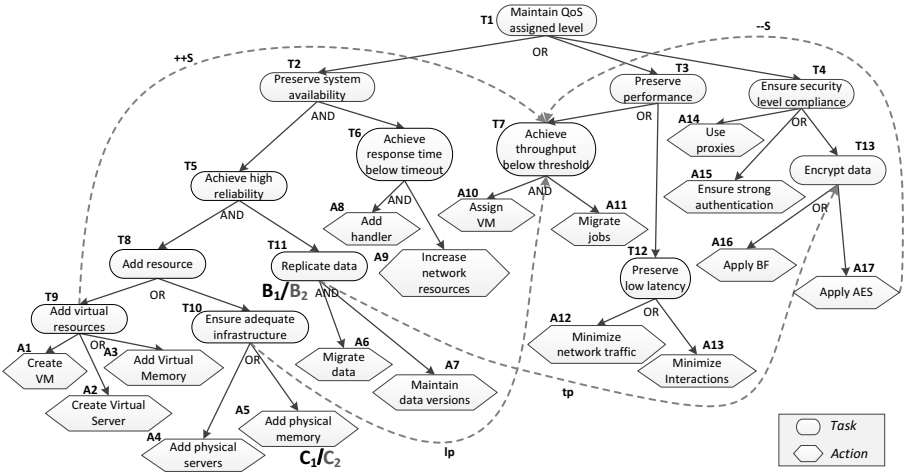

Fig. 2. An Example Decision Task Model

introduced in [5], and indicate the fact that the target node can only be performed as long as the source node has already been performed. In contrast, $\xrightarrow{tp}$ links denote a weaker notion of precondition, which implies that in case both the target and the source node participate in a plan (i.e. a sequence of actions), then the target task/action must performed after the source task/action.

Finally, in a similar manner as in [6], we consider four types of contributions namely, $++S/--S$ meaning that the target node is satisfied/denied when the source node is satisfied, and; $++D/--D$ meaning that the denial of the source node leads to the denial/achievement of the target node. However, in the context of this paper, contribution links can terminate only to task nodes as an action is only satisfied when the corresponding atomic activity is performed and it cannot be fulfilled otherwise.

4 DTM Formal Definition and CNF Generation

In this section we are going to introduce a process that transforms DTMs into CNF formulas that fully capture the constraints modeled by the DTM. This reduces the problem of plan determination to an instance of the SAT problem, and allows for the use of SAT solvers. However, before going into the details of CNF formula generation, we are going to formally define DTMs.

4.1 DTM Formal Definition

A DTM contains a set of task or action nodes which are connected with each other through decomposition rules or rules that describe binary relations on the set of nodes i.e. *binary rules* such as *Precondition*, and *Contribution* links. Hence, we formulate the following definition for DTMs:

Definition 1. *A Decision Task Model is a tuple of the form* $\langle N, R_d, R\rangle$*, where* $N = N_t \cup N_a$ *with* N_t *and* N_a *denoting the sets of task and action nodes respectively,* R_d *is the set of decomposition rules, and* R *the set of binary rules.*

In the above definition, a decomposition rule $r_d \in R_d$ describes the way a *parent* task node p is AND/OR decomposed to a set $\{c_1, c_2, \cdots c_n\}$ of *child* task or action nodes. There must be one decomposition rule for each task node p, which is formally written as:

$$r_d = \langle T, p, \{c_1, c_2, \cdots, c_n\}\rangle \quad \text{where} \quad T \in \{\text{AND}, \text{OR}\}.$$

For example, task node $T7$ ("Achieve throughput below threshold") in Fig. 2 is AND-decomposed to action nodes $A10$ ("Assign VM") and $A11$ ("Migrate Jobs"), so the corresponding decomposition rule is $\langle \text{AND}, T7, \{A10, A11\}\rangle$.

A binary rule $r \in R$ between source node s and target node t is denoted as:

$$r = \langle T, s, t\rangle \quad \text{where} \quad T \in \{\text{lp}, ++S/D, --S/D\}.$$

where as discussed above, for contribution rules the target node $t \in N_t$. In the example DTM in Fig. 2, task node $T7$ participates as the target node of three binary rules, namely $\langle ++S, T9, T7\rangle$, $\langle --S, A17, T7\rangle$ and $\langle \text{lp}, T10, T7\rangle$.

Finally, given a DTM $\langle N, R_d, R\rangle$ and a node $b \in N$, we can define for each type T of binary rule the following set of nodes:

$$N^{[T]}(b) = \{s \in N | \exists r = \langle T, s, b\rangle \in R\} \quad \text{where} \quad T \in \{\text{lp}, ++S/D, --S/D\},$$

which includes the source nodes of all rules of type T for which node b is the target node. Given the DTM of Fig. 2, the following sets can be defined for node $T7$: $N^{++S}(T7) = \{T9\}$, $N^{--S}(T7) = \{A17\}$, and $N^{\text{lp}}(T7) = \{T10\}$.

4.2 Boolean Rules and CNF Formula Extraction

Given a DTM, a corresponding set of Boolean rules that capture all the constraints in the model can be generated. The semantics of those rules as well as their mappings to CNF clauses are originally presented in [7], and for the sake of presentation completeness summarized in Table 2. The required CNF formula can then be easily extracted by taking the conjunction of the CNF clauses corresponding to each individual Boolean rule.

The generation of Boolean rules starts by extracting set $N^{\text{lp}}(p)$, and also the following two sets for each task node $p \in N_t$ of the DTM :

$$N^{\text{pos}}(p) = N^{++S}(p) \cup N^{--D}(p) = \{e_1, \cdots, e_k\} \cup \{f_1, \cdots, f_l\} \tag{1}$$

$$N^{\text{neg}}(p) = N^{--S}(p) \cup N^{++D}(p) = \{g_1, \cdots, g_m\} \cup \{h_1, \cdots, h_o\} \tag{2}$$

which, along with the decomposition rule $r_d = \langle T, p, \{c_1, c_2, \cdots, c_n\}\rangle$ for node p, determine the set of Boolean rules that must be generated for this node. According to whether some or all of those sets are empty, a different set of rules

Table 1. AND/OR rules generation for task and action nodes. There is always a decomposition rule $r_d = \langle T, p, \{c_1, c_2, \cdots, c_n\}\rangle$ for each task node $p \in N_t$.

$N^{pos}(p)$	$N^{neg}(p)$	$N^{lp}(p)$	Generated AND/OR Rules	
			$p \in N_t$	$p \in N_a$
$=\emptyset$	$=\emptyset$	$=\emptyset$	$p \leftarrow \mathrm{T}(c_1, c_2, \cdots, c_n)$	-
$=\emptyset$	$=\emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	$p \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_leaf})$
$=\emptyset$	$\neq\emptyset$	$=\emptyset$	$p \leftarrow \mathrm{AND}(\mathsf{p\_d}, \neg\mathsf{p\_c\_neg})$	-
$=\emptyset$	$\neq\emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{AND}(\mathsf{p\_}\ell, \neg\mathsf{p\_c\_neg})$ $\mathsf{p\_}\ell \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	-
$\neq\emptyset$	$=\emptyset$	$=\emptyset$	$p \leftarrow \mathrm{OR}(\mathsf{p\_d}, \mathsf{p\_c\_pos})$	-
$\neq\emptyset$	$=\emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{OR}(\mathsf{p\_}\ell, \mathsf{p\_c\_pos})$ $\mathsf{p\_}\ell \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	-
$\neq\emptyset$	$\neq\emptyset$	$=\emptyset$	$p \leftarrow \mathrm{OR}(\mathsf{p\_d}, \mathsf{p\_c})$ $p \leftarrow \mathrm{AND}(\mathsf{p\_c\_pos}, \neg\mathsf{p\_c\_neg})$	-
$\neq\emptyset$	$\neq\emptyset$	$\{b_1 \cdots b_q\}$	$p \leftarrow \mathrm{OR}(\mathsf{p\_}\ell, \mathsf{p\_c})$ $\mathsf{p\_c} \leftarrow \mathrm{AND}(\mathsf{p\_c\_pos}, \neg\mathsf{p\_c\_neg})$ $\mathsf{p\_}\ell \leftarrow \mathrm{AND}(b_1 \cdots b_q, \mathsf{p\_d})$	-

is generated as this is illustrated in Table 1. Additionally, the following apply for the pseudo-variables p_c_pos (contributions that positively affect the target node p), p_c_neg (contributions that negatively affect the target node p), and p_d (decomposition rule for node p) that appear in Table 1:

$$\mathsf{p\_c\_pos} \leftarrow \mathrm{OR}(e_1, \cdots, e_k, \neg f_1, \cdots, \neg f_l) \qquad \mathsf{p\_d} \leftarrow \mathrm{T}(c_1, c_2, \cdots, c_n)$$
$$\mathsf{p\_c\_neg} \leftarrow \mathrm{OR}(g_1, \cdots, g_m, \neg h_1, \cdots, \neg h_o)$$

where T is the type of the decomposition rule and nodes e_i, f_i, g_i and h_i correspond to the ones in equations (1) and (2). It is important to note that p_c_pos (p_c_neg) is substituted by e_1 or $\neg f_1$ (g_1 or $\neg h_1$) in case $k = 0$ or $l = 0$ ($m = 0$ or $o = 0$) respectively, while if both of k and l (m and o) are equal to zero, set N^{pos} (N^{neg}) is empty, and the pseudo-variable p_c_pos (p_c_neg) does not appear in the rules. For example, given task node $T7$ of Fig. 2 for which $N^{\mathrm{lp}}(T7) = \{T10\}$, $N^{\mathrm{pos}}(T7) = N^{++S}(T7) = \{T9\}$, and $N^{\mathrm{neg}}(T7) = N^{--S}(T7) = \{A17\}$, the following Boolean rules are generated based on the last case of Table 1:

$$T7 \leftarrow \mathrm{OR}(\mathrm{T7\_}\ell, \mathrm{T7\_c}) \quad (3) \qquad \mathrm{T7\_c} \leftarrow \mathrm{AND}(T9, \neg A17) \quad (5)$$
$$\mathrm{T7\_}\ell \leftarrow \mathrm{AND}(T10, \mathrm{T7\_d}) \quad (4) \qquad \mathrm{T7\_d} \leftarrow \mathrm{AND}(A10, A11) \quad (6)$$

Table 2. Mapping of Boolean rules to CNF Clauses

Boolean Rule	Equivalent Constraints	CNF Clauses
$o \leftarrow \text{AND}(i_1, i_2, \cdots, i_n)$	$\neg i_1 \Rightarrow \neg o, \cdots, \neg i_n \Rightarrow \neg o,$ $i_1 \wedge i_2 \wedge \cdots \wedge i_n \Rightarrow o$	$(i_1 \vee \neg o) \wedge \cdots \wedge (i_n \vee \neg o) \wedge$ $(\neg i_1 \vee \neg i_2 \vee \cdots \vee \neg i_n \vee o)$
$o \leftarrow \text{OR}(i_1, i_2, \cdots, i_n)$	$i_1 \Rightarrow o, \cdots, i_n \Rightarrow o,$ $i_1 \wedge i_2 \wedge \cdots \wedge i_n \Rightarrow o$	$(\neg i_1 \vee o) \wedge \cdots \wedge (\neg i_n \vee o) \wedge$ $(i_1 \vee i_2 \vee \cdots \vee i_n \vee \neg o)$

which can be directly mapped to CNF clauses as this is illustrated in Table 2. For example the following CNF formula corresponds to rule (5):

$$(T9 \vee \neg \text{T7\_c}) \wedge (\neg A17 \vee \neg \text{T7\_c}) \wedge (\neg T9 \vee A17 \vee \text{T7\_c})$$

Finally, Boolean rules are also generated for action nodes when they are the target of $\xrightarrow{lp}$ links. This case is also presented in Table 1 (second row).

5 Reasoning

The extracted CNF formula for the DTM provides a formal model of logical relationships between the DTM nodes. Such a model can be used to apply reasoning techniques in order to identify possible DTM model resolutions as combinations of actions, that if performed in coordination, can realize the root task of the DTM model. In this respect, finding such a task resolution equals to solving the SAT problem for the extracted CNF which would assign truth values to both tasks and actions, and then executing all actions assigned as *true*. However, as discussed above, DTM nodes may be annotated with weights indicating either benefit (for task nodes) or cost (for action nodes). So we are not only interested in finding an assignment that satisfies the CNF formula, but also an assignment that has the best score taking into account the cost/benefit of each *true* node.

Given the nature of the problem, different strategies can be utilized based on whether the model resides in static or dynamic environments. The former are characterized by absent or extremely rare changes of assigned benefits and costs to DTM nodes, while the latter by frequent context changes that lead to changes for the weights associated to all or to a subset of the model's nodes.

5.1 Reasoning in Static Environment

Optimal plan determination is an optimization problem that can be reduced to an instance of Max-SAT problem called *Weighted Partial Max-SAT* (WP-MAXSAT). In WP-MAXSAT we have two sets of clauses namely, hard and soft clauses. The former are the clauses that a solution assignment must satisfy, while the latter are clauses that have weights and the solution must satisfy only those that ensure the maximum total weight.

In our case, the generated CNF corresponds to the hard constraints of the WP-MAXSAT, while the weight-node pairs are used to build the soft constraints of the problem. For example, given the weights illustrated in Table 3 two single literal soft clauses namely, $(T2)$ and $(A8)$ with weights 97 and -412 respectively are generated for nodes $T2$ and $A8$.

5.2 Reasoning in Dynamic Environment

An exhaustive WP-MAXSAT algorithm can be applied to get an optimal plan determination; however, having to apply such an algorithm for each context change can limit significantly the applicability of the approach, especially when working with task models of significant size and complexity in dynamically altering environments. In such cases, computing an approximate to optimal solution may be a preferable alternative instead of attempting to compute the optimal solution, for two primary reasons. First, the average time between any two context changes may be less than the average time required to compute the optimal solution, making the application of typical WP-MAXSAT algorithms impractical. Secondly, even when the time between any two context changes is adequate for WP-MAXSAT, the extra time required to compute the optimal solution may ultimately impose higher aggregate costs than utilizing a good-enough, approximate solution to perform the task right after the context change occurs. Before examining a search process to approximate optimal solutions for task models, we introduce certain concepts required for the definition and application of a local search algorithm that can efficiently explore the solutions space.

Boolean Rules Evaluation. Boolean rules, which have been introduced in the previous section, consist of a set of input variables (i.e. variables denoted as i_1 to i_n in Table 2) and a single output parameter (i.e. variable o in Table 2). Given a Boolean rule B_r, we are going to use the notations $In[B_r]$ and $Out[B_r]$ to signify the input variables and the output parameter of rule B_r respectively.

Definition 2. *We say that a Boolean rule* B_{r_a} directly requires *rule* B_{r_b}*, denoted as* $B_{r_a} \xrightarrow{req} B_{r_b}$ *iff* $Out[B_{r_b}] \in In[B_{r_a}]$.

For example, for the rules presented in the previous section, rule (4) *directly requires* rule (6) as variable $T7\_d$ appears in the input variables of the former and is also the output parameter of the latter rule.

Definition 3. *We say that a Boolean rule* B_{r_a} requires *rule* B_{r_b}*, denoted as* $B_{r_a} \xrightarrow{req*} B_{r_b}$ *iff* $B_{r_a} \xrightarrow{req} B_{r_b}$ *or there exist a Boolean rule* B_{r_k} *such that* $B_{r_a} \xrightarrow{req} B_{r_k}$ *and* $B_{r_k} \xrightarrow{req*} B_{r_b}$.

Rule (3) $\xrightarrow{req*}$ (6) as (3) $\xrightarrow{req}$ (4) because of $T7\_\ell$, and (4) $\xrightarrow{req}$ (6) as we have previously mentioned.

Furthermore, the $\xrightarrow{req*}$ operator provides a mechanism that allows us to create sequences of Boolean rules in which every rule depends *only* on rules that appear earlier in the sequence. More precisely:

Definition 4. *We say that a sequence of Boolean rules* B_{r_1}, B_{r_2}, $\cdots$, B_{r_n} *is a* proper *one if for every pair* B_{r_i}, B_{r_j} *of Boolean rules in the sequence,* B_{r_i} *appears earlier than* B_{r_j} *in case* $B_{r_j} \xrightarrow{req*} B_{r_i}$.

Hence, sequence (6)(5)(4)(3) is a *proper* sequence of Boolean rules, while sequence (3)(6)(5)(4) is not, as (3) $\xrightarrow{req*}$ (6) and the latter appears after the former in the sequence. If circular dependencies exist in a set of Boolean rules, no proper sequence of those rules exists. However, as the DTMs used have no cycles because of the validation phase illustrated in Fig. 1, we ensure that no circular dependencies exist in the Boolean rules sets generated from DTMs, and so there is always a proper sequence for them. Proper sequences of Boolean rules can be computed at definition-time through typical topological sorting algorithms.

Additionally, given a set $B = \{B_{r_1}, B_{r_2}, \cdots, B_{r_n}\}$ of Boolean rules we define the following two sets of variables:

$$L[B] = \bigcup_{i=1}^{n} In[B_{r_i}] - \bigcup_{i=1}^{n} Out[B_{r_i}] \qquad I[B] = \bigcup_{i=1}^{n} Out[B_{r_i}] \tag{7}$$

where the former contains the variables that appear only as input while the latter those that appear as output parameters in the Boolean rules of set B. We call variables in $L[B]$ and $I[B]$ *leaves* and *inner* respectively and we assign weights to them as follows: those that correspond to nodes of the initial model have the same weight as the node, while those that correspond to pseudonodes (i.e. nodes added during CNF generation) have a zero weight. Finally, using a proper sequence of Boolean rules and an assignment on $L[B]$ elements we can propagate the truth assignment to $I[B]$ elements, and by adding the weights of all true variables we can calculate the total weight of the given sequence.

Local Search on DTM Leaves. Motivated by the above scenarios, we propose a parameterized local search algorithm that can be applied as soon as context changes occur and provide solutions whose quality generally converge to the optimal after a number of iterations. The iterations of the search algorithm can be configured through input parameters so that the search process can be tunned to run within acceptable execution times. The defined search algorithm operates on $L[B]$ and attempts to rapidly reach solutions of improving quality. It should be noted that while a WP-MAXSAT algorithm has to be applied to the whole weighted model (that is, variables corresponding to both inner and leaf nodes) to compute the optimal solution, the proposed search algorithm constructs truth assignments for leaves only by mutating previous ones. In this way, the search space is considerably smaller, particularly for task models of significant depth and complexity (i.e. high inner to leaf nodes ratio). The search process begins by examining a fixed size pool of cached solutions that fulfill the model's hard constraints, and selects the best-performing one. For the process to start even one cached solution suffices, and this solution can be obtained with low computation cost by utilizing a simple SAT solver applied once and offline. The solutions pool is enriched with new solutions as these are discovered at execution-time. By keeping the solution pool with a fixed size we allow for better computation

Algorithm 1. Leaves Local Search

Input: X: Initial solution, B: Boolean rules, $L[B]$: Leaves of B, FF: flips-factor, NF: neighborhood-factor

1: $S \leftarrow X$
2: $x \leftarrow \{x_n\}$, where $x_n = \langle n, v \rangle$, $x_n \in X \wedge n \in L \wedge v \in \{TRUE, FALSE\}$
3: **for** $i = 1$ **to** $FF * |L[B]|$ **do**
4: $y \leftarrow x$
5: $d \leftarrow$ random integer $\in [1, NF * |L[B]|]$
6: **for** $j = 1$ **to** d **do**
7: $y_m \leftarrow \neg x_n$, where n random leaf
8: **end for**
9: **if** $evaluate(y, R)$ **then**
10: $Y = propagate(y, R)$
11: **if** *weight(Y)≥weight(X)* **then**
12: $S \leftarrow Y$
13: $x \leftarrow y$
14: **end if**
15: **end if**
16: **end for**
17: **return** S

complexity for seed selection when a context change occurs. Once such a solution is selected, the local search algorithm is applied on $L[B]$ in order to gradually reach better solutions.

The proposed Leaves Local Search (LLS) algorithm begins with setting as current solution the one provided and proceeds by computing an initial leaves truth assignment based on the provided solution (lines 1-2). Then, the algorithm applies a set of assignment mutating steps for a fixed number of times which is equal to a specified flips factor parameter (FF) times the number of leaves. During the mutating steps, the assignment's values are flipped in random pairs, for up to a different number of pairs in each iteration which is equal to a specified neighborhood factor (NF) parameter times the number of leaves (lines 5-8). The new leaves truth assignment is evaluated against a precomputed proper sequence of Boolean rules to examine whether it leads to a solution of the model, i.e. root R is satisfied (line 9). If the new leaves assignment leads to a solution for the model, then the respective DTM solution's weight is compared to the currently selected solution and provided that the former is better, it becomes the selected solution, while the next iteration of the mutating process will be applied to the leaves assignment that led to the new solution (lines 10-15). Finally, the algorithm returns the best solution met throughout the search.

5.3 Reasoning Example

To illustrate the application of the WP-MAXSAT algorithm and the execution of the LLS algorithm for different contexts (i.e. different weights t_1, t_2) and for different FF values, we use the DTM presented in Fig. 2. In this respect, Table 3 contains two weight assignments (t_1 and t_2) in the DTM example, where the second assignment (t_2) reflects changes in the initial weights (t_1) of nodes $A5$ and $T11$ as a result of a context change. As discussed before, task nodes are annotated with positive weight values indicating the benefit included in fulfilling the task while action nodes are annotated with negative values reflecting the associated

Table 3. Weights assignment for two context changes (DTM in Fig. 2)

	A11	A17	A4	**A5**	A6	A8	A9	**T11**	T2	T4	T5	T6	T7
Weight (t_1)	-709	-359	-26	**-957**	-841	-412	-79	**0**	97	608	643	664	976
Weight (t_2)	-709	-359	-26	**-593**	-841	-412	-79	**599**	97	608	643	664	976

Table 4. Results on Running Example

	Leaves Local Search							WP-MAXSAT
FF	0	1	2	3	4	5	≥ 6	-
Weight (t_1)	-395	1022	1172	1757	1731	1584	1757	1757
Weight (t_2)	568	1546	1520	2255	2255	2255	2255	2255

cost for executing the activity. The model's nodes that are not included in Table 3 have weights that are equal to zero. Based on the above weight annotations, we apply WP-MAXSAT and compute an optimal solution for the problem with total weights equal to 1757 for context t_1 and 2255 for context t_2.

In order to apply the LLS algorithm we utilize a solution computed by a SAT solver which will be used as a seed. Table 4 presents the results of the LLS algorithm with different FF values for both the initial weight assignment as well as the one after the context change. As the number of iterations increases the acquired solution converges to the optimal one. Also, the LLS algorithm required less iterations ($FF \geq 3$ for context t_2 vs. $FF \geq 6$ for context t_1) in order to reach the optimal solution during the second run. This relationship between the rate of context changes and the solution quality performance of LLS is examined in the next section. In this respect, in order to evaluate and assess the performance of the algorithm with regard to solution quality, as well as the computational requirements we discuss a series of experiments in the following section.

6 Case Studies and Experiments

In order to evaluate the applicability and the performance of the proposed framework we conducted a series of experiments with randomly generated task models of varying size and complexity. Using these models we evaluated the application of a WP-MAXSAT algorithm with regards to execution time required for models of different size. Additionally, using the same models, we evaluated and compared the performance of the search process presented in the previous section with regard to the execution time required as well as the quality of acquired solution for different FF values.

In order to evaluate the proposed framework we implemented a random task model generation mechanism which, given certain parameters, such as model size, task vs. actions ratio, AND vs. OR decompositions ratio and maximum binary rules coverage returned a randomly generated task model. Due to space limitations, the results presented and discussed in this paper were acquired with

the following task model generation configuration: *a*) model size ($|N|$): 20 - 300, with an interval of 20 nodes *b*) task vs. actions ratio: 1 *c*) AND vs. OR decomposition ratio: 1 *d*) Maximum binary rules coverage: 30% (percentage of nodes participating in on or more binary rules) We used the above configuration and acquired 10 randomly generated models per model size, for 15 model sizes. For each model, we simulated five context changes each of which assigned different weight values to 10 percent of the nodes and for each context change we ran the WP-MAXSAT algorithm to get the optimal solution, as well as the WP-MINSAT algorithm to get the worst solution for the model with regard to total weight. In order to evaluate and compare the search process performance vs. the WP-MAXSAT results, we used the following measures:

- $\mathsf{t}_{\mathsf{wp-maxsat}}$: WP-MAXSAT algorithm execution time,
- $\mathsf{t}_{\mathsf{lls}}^{\mathsf{FF}}$: LLS algorithm execution time with flips factor FF, and
- $\mathsf{Q_S} = \frac{\mathsf{W_S} - \mathsf{W_{worst}}}{\mathsf{W_{optimal}} - \mathsf{W_{worst}}}$: solution quality,

where $\mathsf{W_S}$ is the total weight of the solution, and $\mathsf{W_{optimal}}$ and $\mathsf{W_{worst}}$ being the total weights of the solutions computed by WP-MAXSAT and WP-MINSAT algorithms respectively. For each generated model the search process was executed for the following parameter values: $FF \in \{5, 10, 15, 20, 25, 30, 35, 40\}$, $NF = 0.2$ and average values were computed for the above measures.

6.1 Results

The presentation of the results is split into: (a) evaluation of *time performance* through considering $\mathsf{t}_{\mathsf{wp-maxsat}}$ and $\mathsf{t}_{\mathsf{lls}}^{\mathsf{FF}}$, and (b) evaluation of *solution quality* through computing $\mathsf{Q_S}$ for different model sizes and flips factor values.

Time Performance. Fig. 3 depicts the average $\mathsf{t}_{\mathsf{wp-maxsat}}$ versus model size. The execution time required by the WP-MAXSAT algorithm scales exponentially with regard to model size, which is expected due to the nature of the problem. Additionally, Fig. 3 depicts the average $\mathsf{t}_{\mathsf{lls}}^{5}$ as well as the average $\mathsf{t}_{\mathsf{lls}}^{40}$ for all examined model sizes ($FF = 5$ and $FF = 40$ are the minimum and maximum flips factor values considered in our experiments) while results for all intermediate FF values lie between the results for the two extreme values depicted. $\mathsf{t}_{\mathsf{lls}}^{5}$ is generally proportionate to FF; however, independent of the FF value, the average $\mathsf{t}_{\mathsf{lls}}$ scales almost linearly to the number of nodes. For large models, LLS is significantly faster compared to applying WP-MAXSAT. For instance, for model size equal to 300 nodes and flips factor value equal to 40, LLS required on average $\frac{1}{1000}$ of the time that WP-MAXSAT required and compute solutions of high quality ($\mathsf{Q_S} = 0.938$) as will be discussed in the next section.

Solution Quality. Fig. 4 presents how $\mathsf{Q_S}$ varies versus FF values for model sizes of 100 and 300 nodes. The quality of the initial solution is depicted for $FF = 0$ and it is around 0.5 for both model sizes. From that point on, as FF increases, the quality converges asymptotically to the optimal solution getting its maximum value for $FF = 40$ (0.961 for size = 100 and 0.938 for size = 300).

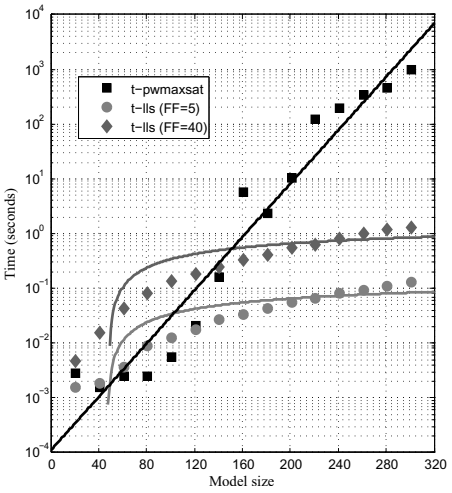

Fig. 3. Execution time

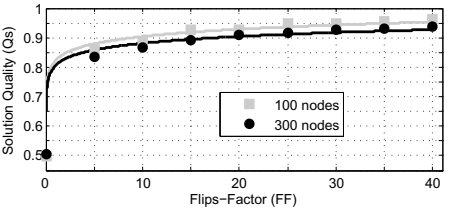

Fig. 4. Q_S vs. FF

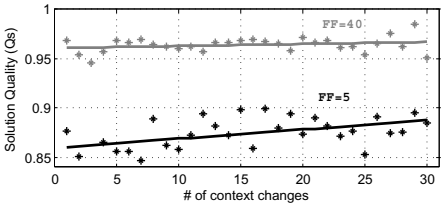

Fig. 5. Q_S vs. number of context changes

Finally, Fig. 5 depicts the average solution quality for the proposed search process, with regard to the number of consecutive context changes using average values from 3 models of 200 nodes and 30 context changes. As the number of consecutive context changes in a dynamic environment increases and the LLS algorithm is applied more times, the solution quality provided by the algorithm is improved for each run. Also, the lower the FF, the more evident the improvement effect is. This effect can be explained by the fact that context changes are gradual, allowing for the algorithm's seed solutions that originate from the solutions' cache to be of higher quality. In this respect, as the starting point for each application of the algorithm gets better, the outcome with regard to solution quality is improved, indicating thus in environments with high rates of gradual context changes, using the proposed search technique can provide further benefit.

7 Related Work

Decision support and action selection for a given context and a given set of goals is a key problem that still motivates researchers and practitioners to devise solutions for. There are two main facets to this problem, namely decision support for static environments, and dynamic decision support for dynamic environments. In the first category, [8] discusses a qualitative approach as well as a numerical approach for reasoning with goal models. In [9] an approach of evaluating qualitative or quantitative satisfaction levels of goals and tasks through the propagation of appropriate values via the goal model links, is presented.

A variation to these approaches are utility based decision support approaches that aim also to maximize or minimize a utility function such as benefit or cost for each task and action, or the number of constraints that are satisfied for each possible plan. Assuming that a goal or other type of model can be represented

as logic formulas a number of reasoning approaches based on SAT reasoners are applicable. In [10] an algorithm that implements Weighted Partial MAX-SAT by successively invoking a SAT solver and by attempting to minimize the penalty for not satisfying soft-constraints, is presented.

In [11], GRASP a search algorithm for propositional satisfiability is proposed. The search algorithm is based on the concept of identification of assignments that cause conflicts at a given level, and then non-chronologically backtrack to earlier levels to improve pruning of the search space. In [12] an extension of the GRAPSP algorithm augmented with path re-linking that attempts to intensify and focus the search around good-quality isolated solutions that have been originally computed by the GRASP algorithm is presented. The basic difference of the GRASP and also the path re-linking augmented GRASP algorithms from the approach here is that, in our approach we do not utilize backtracking or a and we are solely based on mutations of the possible assignments of truth values to the variables in the given set of clauses. The quality of the produced solution is based on the number of mutations (i.e. iterations) and the size of possible mutations in each flip. The augmented GRASP algorithm guarantees a better approximate solution to the Weighted MAX-SAT than ours, as it already starts from known good (elite) solutions, but on the other hand requires the use of GRASP as its initial input.

In the second category, the environment is considered dynamic in the sense that actions of a plan may become impossible once the plan is devised and started being enacted because the environment is dynamically altered. Work in this category relates to approaches proposed in autonomous agents and autonomic software systems. In [13] an approach that allows for reasoning about partial satisfaction of soft-goals is discussed. The approach is based on the annotation of softgoals with reward (e.g. benefit), and penalty (e.g. cost) functions. The approach utilizes Dynamic Decision Networks (DDN) in order to identify a selection of softgoals that provide an optimal decision with respect to softgoal satisfaction and the utility functions used. The difference from our approach is that we do not require the compilation of intermediate models such as a DDN, and we allow for utility functions (rewards, and penalties) to vary dynamically as the system operates. In [14] a framework that implements an adaptation manager for autonomic systems is proposed. The framework is based on the Goal-Attribute-Action model and allows for a decision to be reached regarding the selection of actions in order to adapt or re-configure an autonomic system according to specified goals that need be reached.

8 Conclusion

In this paper, we investigated the use of local search algorithms and boolean expression evaluators to reason with DTMs when labels related to cost and benefit values for actions and tasks are altered as a result of context changes. The approach allows for obtaining a solution that is within 90% of the optimal value at a fraction of time that is required by a Weighted Partial MAX-SAT algorithm to compute the optimal solution for the problem at hand. The applicability and the performance of the

proposed framework was evaluated with promising results by conducting a series of experiments with randomly generated task models of varying size and complexity.

Acknowledgment. The research of G. Chatzikonstantinou is co-funded by the European Union (European Social Fund ESF) and Greek National funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund. The research of M. Athanasopoulos is supported by IBM Canada CAS Research under the Research Fellowship Project No. 754.

References

1. Yu, E.: Modelling strategic relationships for process reengineering. PhD thesis, University of Toronto Toronto (1996)
2. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Trans. Software Eng. 26(10), 978–1005 (2000)
3. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. 20(1-2), 3–50 (1993)
4. Amyot, D.: Introduction to the user requirements notation: Learning by example. Comput. Netw. 42(3), 285–301 (2003)
5. Liaskos, S., Khan, S.M., Litoiu, M., Jungblut, M.D., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. Inf. Syst. 37(8), 767–783 (2012)
6. Chopra, A.K., Dalpiaz, F., Giorgini, P., Mylopoulos, J.: Reasoning about agents and protocols via goals and commitments. In: AAMAS, pp. 457–464 (2010)
7. Velev, M.N.: Efficient translation of boolean formulas to cnf in formal verification of microprocessors. In: Proceedings of the 2004 Asia and South Pacific Design Automation Conference, ASP-DAC 2004 (2004)
8. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with goal models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)
9. Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., Yu, E.S.K.: Evaluating goal models within the goal-oriented requirement language. Int. J. Intell. Syst. 25(8), 841–877 (2010)
10. Ansótegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial maxsat. In: AAAI (2010)
11. Silva, J.P.M., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Trans. Computers 48(5), 506–521 (1999)
12. Festa, P., Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: GRASP with path-relinking for the weighted maximum satisfiability problem. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 367–379. Springer, Heidelberg (2005)
13. Bencomo, N., Belaggoun, A.: Supporting decision-making for self-adaptive systems: From goal models to dynamic decision networks. In: Doerr, J., Opdahl, A.L. (eds.) REFSQ 2013. LNCS, vol. 7830, pp. 221–236. Springer, Heidelberg (2013)
14. Salehie, M., Tahvildari, L.: A weighted voting mechanism for action selection problem in self-adaptive software. In: SASO, pp. 328–331 (2007)
15. Ernst, N.A., Mylopoulos, J., Borgida, A., Jureta, I.J.: Reasoning with optional and preferred requirements. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 118–131. Springer, Heidelberg (2010)