Optimizing Monitoring Requirements in Self-Adaptive Systems

Raian Ali[1], Alberto Griggio[2], Anders Franzén[3],
Fabiano Dalpiaz[4], and Paolo Giorgini[4]

[1] Bournemouth University, UK
[2] Fondazione Bruno Kessler, Italy
[3] Jasper Design Automation, Sweden
[4] DISI, University of Trento, Italy

Abstract. Monitoring the system environment is a key functionality of a self-adaptive system. *Monitoring requirements* denote the information a self-adaptive system has to capture at runtime to decide upon whether an adaptation action has to be taken. The identification of monitoring requirements is a complex task which can easily lead to redundancy and uselessness in the set of information to be monitored and this, consequently, means unjustified instalment of monitoring infrastructure and extra processing time. In this paper, we study the optimization of monitoring requirements. We discuss the case of *contextual goal model*, which is a requirements model that weaves between variability of goals (functional and non-functional requirements) and variability of context (monitoring requirements) and is meant to be used for modelling mobile and self-adaptive systems requirements. We provide automated analysis —based on a SAT-solver— to process a contextual goal model and find a reduced set of contextual information monitor guaranteeing that this reduction does not sacrifice the system ability of taking correct adaptation decisions when fulfilling its requirements.

Key words: Requirements; Adaptation; Mobility; Monitoring Optimization

1 Introduction

Monitoring requirements is a notion which refers to the information that a system has to capture at runtime to assess the effectiveness of its current configuration and, if needed, to determine how to adapt to a more suitable configuration. The monitoring activity requires the deployment of an adequate infrastructure, such as sensors and databases, and collects up-to-date values of certain attributes of the system internal state (e.g., available resources, errors and faults, and security breaches) and the surrounding environment (e.g., user's location, user's computing device, user's movement and activity, temperature). Depending on the collected information, certain adaptation actions may be triggered in order to switch the system into different course of execution.

Monitoring is crucial in several areas and for several reasons. In addition to feeding logging and reporting functionalities, which is essential for the off-line processing established by system analysts, monitoring is a fundamental activity of self-adaptive systems [15]. These systems are autonomously capable of observing their environment and internal state and adapting their behaviour in order to keep their functional requirements fulfilled and optimize the satisfaction of their non-functional requirements.

Monitoring is a costly activity, both in terms of resources and processing time. The deployment of a comprehensive monitoring infrastructure may be as expensive as the implementation of the core functionalities of the system. This is especially true when considering online monitoring, such as in self-adaptive systems, where the monitoring infrastructure has to continuously collect and process information at runtime. Mobile information systems [14] are a clear example of an adaptive system which is subject to this problem, as some monitoring functionalities have to run on hand-held devices with low processing power and low-capacity battery. In these systems, optimizing monitoring requirements is essential both to reduce costs and to maximize performance.

In our research, we focus on monitoring information that is relevant at the requirements level, namely that affects requirements activation, applicability, and satisfaction. We adopt the notion of *context* to represent any information related to the system environment which affect system's requirements. Specifically, we build on top of *contextual goal models* proposed by Ali et al. in [1, 2], a requirements engineering framework that allows for representing, analysing, and reasoning about the relationship between context and requirements (goals). The authors of these works stated that the identified and represented contexts can become very complex and this causes redundancy and/or inconsistency problems in both of functional and monitoring requirements.

Example. Let us consider the following three contextual attributes which affect the behaviour of a museum-guide mobile information system: $a1$= "the visitor is in the reception area", $a2$= "there is enough light in the user's location", and $a3$= "the user location is not noisy". Imagine that the context $C1$= $a1 \wedge a2$ is a precondition for a system requirement of showing a demo using certain visualizing settings. In a museum where the light level in the reception area is always high, i.e. $a1 \rightarrow a2$, $C1$ can be reduced to $a1$ and, thus, the system should not monitor the light level and there will be no need to deploy the sensors needed for that. In another case, a behaviour like adopting voice recognition is allowed only in the reception area and feasible when the place is quite, i.e., when $C2$= $a1 \wedge a2$ holds. In a busy museum where the reception is always noisy then $C2$ could be reduced to false and there will be no need to install its monitoring infrastructure unless it helps for monitoring other contexts for other functionalities.

In this paper, we develop a modeling and analysis framework to optimize monitoring requirements in adaptive systems. Our ultimate goal is to minimize the amount and costs of information to monitor in a way that does not compromises the system ability in taking correct adaptation decisions. We study the case of contextual goal models and propose to consider the dependencies between context defined on it as an input for our developed SAT-based solver. We develop algorithms to optimize monitoring requirements and explain our approach on a scenario of a mobile information system for assisting visitors of a museum.

The paper is structured as follows. Section 2 explains our baseline: the contextual goal model proposed in [1, 2]. Section 3 states and analyzes the context dependency importance for optimizing monitoring requirements. Section 4 presents automated reasoning to treat redundancy, triviality, and inconsistency in a monitoring requirements specification. Section 5 discusses related and future work.

2 Background: Contextual Goal Model

Context, the environment surrounding the system [12], is an important aspect in adaptive systems engineering, which has to be considered since the early requirements analysis stage. We consider Goal-Oriented Requirements Engineering (GORE), in which stakeholders goals are identified and analyzed, and alternative system behaviours—sets of tasks the system shall implement—are identified to satisfy such goals. In [2, 1], Ali et al. advocate the importance of integrating contextual factors in goal analysis, so as to allow for the derivation of contextualized goal satisfaction alternatives. Moreover, the authors explained how context itself needs to be modeled and analyzed. Goal analysis provides constructs to hierarchically analyze goals and discover alternative sets of tasks the system can execute to satisfy such goals, while context analysis provides constructs to hierarchically analyze context and discover alternative sets of facts the system has to monitor to verify a context and then act accordingly.

Fig. 1 represents a contextual goal model for a mobile information system that assists the visitors of a museum and interacts with them and assistance staff, mainly through their personal digital assistants (PDAs). The model represents alternative ways to satisfy the top-level goal of the system: giving information to visitors about the pieces of art in the museum. Contextual annotations ($C1..C8$) express the relationship between context and goals, and are related to the following variation points in the goal model:

1. *Or-decomposition*: a contextual annotation on a branch specifies that an alternative sub-goal (task) can be adopted only if that context holds. For instance, to provide information about a piece of art, a visitor can be directed to a dedicated terminal only if such terminal is free and close to the visitor and he/she is able to use and interact with it $(C2)$. The visitor's PDA can be alternatively used to show information when the piece of the art information are not so complicated, and the visitor has the ability and the knowledge to use PDAs $(C3)$. Getting information through an assistance staff requires that the visitor is not able to use his PDA and not familiar with terminals, or that the visitor is classified as an important visitor $(C4)$.
2. *Root goals* . The activation of a root goal depends on a contextual trigger. To activate goal *"visitor gets informed about a piece of art"*, the system needs to verify if the visitor is interested in the piece, does not already know about it, and if there is still time to explain about it $(C1)$.
3. *Means-end*: goals can be ultimately satisfied by means of specific executable processes (*tasks*). The adoptability of each task in means-end decompositions might depend on the context. The visitor can be notified about the availability of information terminals through a voice message when he/she puts the headphones on, and is not talking to somebody or using his/her PDA for a call $(C5)$, while notifying him/her by SMS can be adopted in the opposite context $(\neg C5)$.
4. *Actor dependency*: An actor can attain a goal or get a task executed by delegating it to another actor only in a specific context. The dependency between the two system actors for providing the information to a visitor through an assistance staff requires a staff that is close to and talks a language common to the visitor, and knows enough about the considered piece of art comparing to the visitor knowledge $(C6)$.

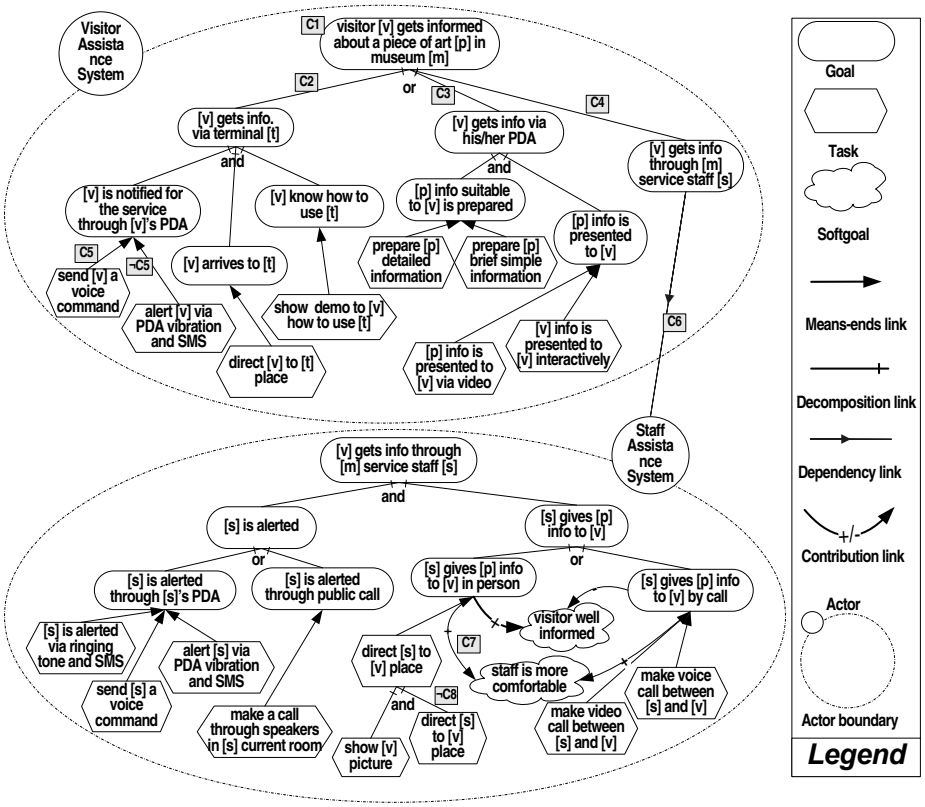

Fig. 1. Tropos goal model for the museum assistance system

5. *And-decomposition*: a sub-goal/sub-task is needed to achieve their parent goal only in a certain context. Guiding the assistance staff to the visitor place is not needed if the visitor stays around and can be seen directly by the assistance staff $(C8)$.
6. *Contribution to softgoals*: softgoals are qualitative objectives having no clear-cut satisfaction criteria, and can be contributed either positively or negatively by goals and tasks. The contribution value can vary from one context to another. Giving the information in person is comfortable if the visitor is in the same room as the assistant $(C7)$, while it is not comfortable when they are in different rooms.

Contextual precondition on the goal model might need to be analyzed, in order to identify, represent, and agree on how the system can monitor and verify if a context holds. The context analysis proposed in [2, 1] provides modeling constructs to hierarchically analyze context. An example of the analysis of the context $C1$ of Fig. 1 is shown in Fig. 2.

Accordingly, context is specified as a formula of world predicates. Based on their verifiability by an actor, world predicates can be either *facts* or *statements*. A world predicate is a fact (a statement) for an actor, if that actor can (cannot) verify it. Ver-

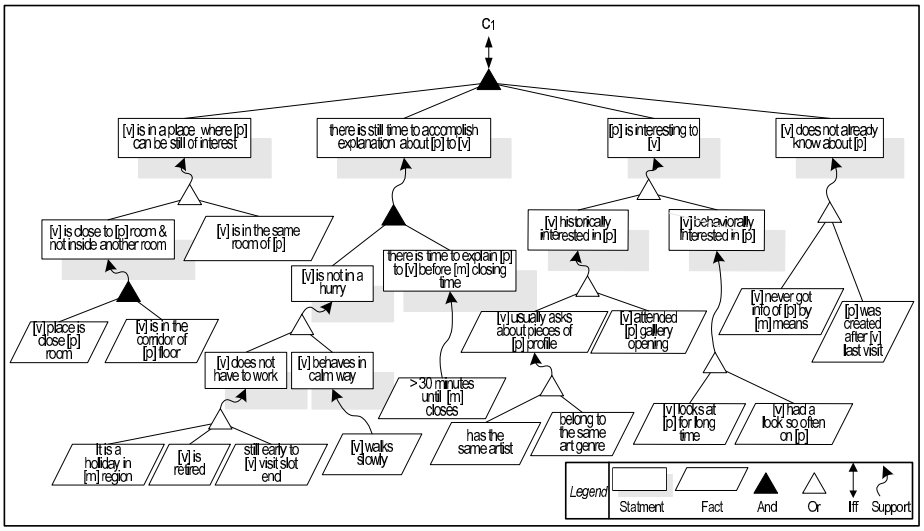

Fig. 2. Context analysis for $C1$: identifying observable facts to judge whether a context holds

ifiability is clearly linked to *monitoring requirements*: a fact requires the deployment of an adequate IT infrastructure (sensors, databases, etc.) to determine its value. Statements, on the other side, cannot be monitored as such: analysts shall further analyze them till reaching observable facts that the system can monitor. Context analysis allows the analyst to specify that a fact (or a formula of facts) is a means to infer evidence that a statement holds. To represent such evidence, context analysis provides the notion of *support* relation.

We explain context analysis constructs with the help of examples from Fig. 2. *"the piece of art [p] artist [a] has lived in the visitor's [v] city of birth"* is a fact, as the system can verify through checking the profile of the artist and the city of birth of the registered visitor. *"the visitor [v] is interested in the piece of art [p]"* is a statement, since the system can not determine its truth value without further analysis. The previous statement is supported if either *"[v] is behaviorally interested in [p]"*, or *"[v] is historically interested in [p]"*. The system can get evidence of the first sub-statement via the support relation from the formula of facts *"[v] looks at [p] for long time"* and *"[v] comes to [p] area and has a look at [p] so often"*.

3 Context Dependency

In this section, we discuss context dependency, show its importance, and argue about its generality and complexity. The context hierarchial analysis we explained in the last section, is easily transformable into a propositional formula consists of the leaf facts as atomic predicates (variables). The dependency, namely the implications, between these facts can make the context formula redundant, trivial, or even inconsistent. Here we give the definition of *redundant* context and its two extremes: the *trivial* and the *inconsistent*.

Definition 1 (Redundant Context) *given the implications between its facts, a context is redundant iff some facts has no effect on its validity.*

Definition 2 (Trivial Context) *given the implications between its facts, a context is trivial iff it is always reduced to true.*

Definition 3 (Inconsistent Context) *given the implications between its facts, a context is inconsistent iff it is always reduced to false.*

Context redundancy makes the context representation more complex without justification and leads to a useless monitoring of facts that have no effect on its validity. Context redundancy motivates us to *optimize monitoring requirements*. Let us take the two facts $f1 =$ *"the visitor is inside a museum room"* and $f2 =$ *"there is enough light at the location of the visitor"*. Consider a context $C = f1 \wedge f1$; if the system is going to operate in a museum that its rooms are well illuminated then $f1 \rightarrow f2$ and C is reduced to $f1$ which means that there is no need to install sensors to capture the level of light in the museum. Alternatively, if $C = f1 \vee f2$ then C is reduced to $f2$ and there will be no need for installing a positioning system to decide if the visitor is inside a museum room. Some base reductions rules are shown in the following table:

Assured Implication	$A \vee B \leftrightarrow$	$A \wedge B \leftrightarrow$
$A \rightarrow B$	B	A
$A \rightarrow \neg B$	$A\ xor\ B$	$false$
$A \leftrightarrow \neg B$	$true$	$false$

While the redundancy of context implies a redundancy in monitoring requirements; context inconsistency adds to that inapplicability in the software functionalities. Besides the uselessness of monitoring their facts, inconsistent contexts deny the adoptability of the software functionalities preconditioned by them. E.g., if a functionality is preconditioned by the context $C = f1 \wedge f2$, and if the museum rooms are not well illuminated for some decoration reasons or to conserve the quality of a piece of art, i.e. $f1 \rightarrow \neg f2$, then such functionality is never adoptable since C is inconsistent.

After showing its influence, now we argue about the generality of the context dependency problem and that it is not tied to or caused by our context analysis and goal model. We expect any self-adaptive system to monitor several pieces of information regarding its context that could be also combined through logical relations to conform logical expressions. Assuring some implications between these information pieces might reveal a problem of redundancy, triviality, or inconsistency. Let us take the following generic pseudo-code that can be part of a decision making process of a self-adaptive system:

```
1: if (A ∨ B) ∧ C then
2:    if D ∧ E then
3:       adopt alternative set of actions (action_set 1)
4:    else
5:       if F ∨ G then
6:          adopt another set of actions (action_set 2)
7:       end if
8:    end if
9: end if
```

The three contexts boolean abstractions we have here are $(A \vee B) \wedge C$, $D \wedge E$, and $F \vee G$, that involve monitoring the set of facts $S = \{A, B, C, D, E, F, G\}$. The model has two alternative set of actions *action_set 1* and *action_set 2* each fitting to a certain context. When we assure the implication $C \rightarrow A$ then $(A \vee B) \wedge C$ can be reduced to just C and therefore there will be no need to monitor A and B. If there is no implication between D and E then $D \wedge E$ alone is not redundant, but this does not mean that *action_set 1* is adoptable or D and E are not redundant; suppose we have the implication $C \rightarrow \neg E$, then the accumulated context at line 2, which is $C \wedge D \wedge E$, is inconsistent and reduced to *false*, and the *action_set 1* is not adoptable. In case we assure that $C \rightarrow \neg G$ then the accumulated context at line 5, $C \wedge (F \vee G)$, can be reduced to $C \wedge F$ which means that G is redundant and has no effect on the validity of the accumulated context.

The implications between facts can be *absolute* or *dependent* on the characteristics of the system environment. Absolute implications are applied wherever the system has to operate, e.g., $f3 \rightarrow \neg(f4 \wedge f5)$ where $f3$ =*"piece of art [p] is always exclusive to museum [m]"*, $f4$ =*"visitor is visiting the museum [m] for the first time"* and $f5$=*"visitor has seen [p] some date before today"*. Other implications depend on the nature of the environment the system is going to operate in, like the aforementioned implication between the light level and being inside a museum room. Moreover, the environment itself assures that some contexts are always true or false, so we have to consider a special kind of implications between the system environment and context analysis facts, i.e. $Env \rightarrow facts\_formula$. E.g., if the museum opens only in holidays, so the fact *"the day is holiday in museum region"* is always true.

Facts verification has costs; costs are those related to the facts verification process itself and to getting the data needed to make the verification possible, such as installing physical equipments like sensors, inserting data by human operators, having enough storage, processing time and so on. When we have more than one possibility to reduce contexts, we should choose the one that minimizes the costs.

After the above explanation of the context dependency and its effects, we now explain two main specific problems that we need to face in order to optimize monitoring requirements:

– *Optimizing monitoring requirements*: checking and fixing the redundancy, triviality, and inconsistency of contexts lead to minimizing the costs of the system as it avoids us sensing, storing, processing data to verify facts that have no effect on any decision, and developing software functionalities that are preconditioned by inconsistent contexts until such inconsistencies are fixed.
  Let us consider the contextual goal model of the top-left of Fig. 3. Whenever the analyst defines a direct context at each variation point ($C1$, $C2$, $C3$, $C4$, $C5$, $C6$), the automated reasoning has to check if this direct context alone and if the accumulated context ($C1$, $C1 \wedge C2$, $C1 \wedge C3$, $C1 \wedge C4$, $C1 \wedge C5$, $C1 \wedge C3 \wedge C6$ respectively) are consistent and non trivial, and notify the analyst to fix any error before repeating the check and proceeding with the next contexts. However, this check has also to be done progressively for the accumulated context on the alternatives in the goal model to know if they can be adopted together, e.g., if $C1 \wedge C2 \wedge C5$ is inconsistent then the root goal satisfaction alternative $A2$= $\{G5, G3, G8\}$ is never adopted.

After defining contexts at all of the variation points and passing the consistency and triviality check, we can start to optimize the monitoring requirements. Optimization takes the set of all contexts associated to the different goal satisfaction alternatives $\{A1, A2, A3, A4\}$and softgoals $\{SG1\}$, i.e. $\{A1C, A2C, A3C, A4C\}$, and $\{SG1C\}$ and gives equivalent reduced contexts $\{A1C', A2C', A3C', A4C'\}$, and $\{SG1C'\}$ that can be verified on a sub-set of facts with minimum monitoring costs. The analyst has to study the set of facts of the resulted formulas and elicit the data that the monitoring system has to obtain.

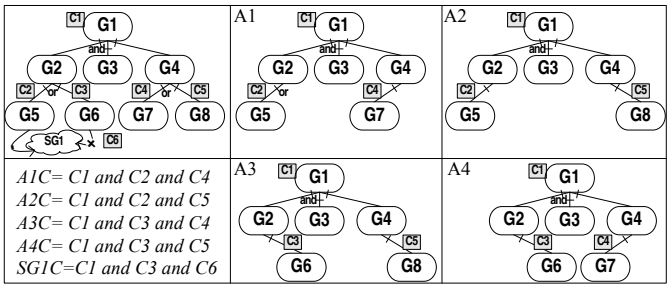

Fig. 3. The accumulated context for the root goal satisfaction alternatives and softgoals

The automated reduction has to minimize the total cost of monitoring all the reduced contexts facts, as doing it separately for each context of $\{A1C, A2C, A3C, A4C\}$, and $\{SG1C\}$ does not guarantee, in the general case, having the minimal total monitoring costs. The problem of optimizing a set of contexts together to reduce the overall cost is highly expensive as we explain later. In the next section, we provide an algorithm, based on SAT techniques and greedy algorithms, that takes a context formula together with the implications between facts (assumptions), checks its consistency and redundancy, and produces an equivalent formula with less costs.

– *Efficient specification of implications*: when the number of facts is high, it will be hard for the analyst to specify even the binary implications between facts. Moreover, the specified implications themselves might be wrong and inconsistent. Designing a supporting tool that helps the analyst to correctly specify, with a minimum number of interactions, the implications between facts is another main problem. We expect such tool to make a kind of facts analysis and asks the analyst to specify the relation where the probability of implication is high. In this paper, we do not provide solution for this problem and we aim to address it in future work.

4 SAT-based Redundancy Elimination

In this section we describe our algorithm for determining whether a context is inconsistent or trivial, and for identifying redundant facts in a context. The algorithm is based on propositional satisfiability (SAT), and in particular on SAT-based techniques for the enumeration of all the models of a propositional formula. Before describing the algorithm, we recall some necessary definitions and notions from propositional logic.

Definition 4 (Model, Satisfiability, Equivalence) *Let φ be a propositional formula,and $V(\varphi)$ be the set of its atomic predicates. Let μ be a function $\mu : V(\varphi) \to \{0, 1\}$, and let ν be a function from propositional formulas to the set $\{0, 1\}$ defined as:* [1]

$$\nu(P) = \mu(P), P \in V(\varphi) \qquad \nu(\neg\phi) = 1 - \nu(\varphi) \qquad \nu(\phi \wedge \psi) = min(\nu(\phi), \nu(\psi))$$

μ is a model *for φ if $\nu(\varphi) = 1$. φ is* satisfiable *if it has at least one model,* unsatisfiable *otherwise.*

Two formulas ϕ and ψ are equivalent *if and only if they have the same models. A formula ϕ* entails *another formula ψ, denoted as $\phi \models \psi$, if all the models of ϕ are also models of ψ, but not vice versa.*

In what follows, we might denote a model μ as a *set of literals* μ_S, such that for each variable P, if $\mu(P) = 1$ then $P \in \mu_S$, and if $\mu(P) = 0$ then $\neg P \in \mu_S$. Analogously, we might denote μ as a *formula* μ_F which is the conjunction of the literals in μ_S.[2]

Example 1. Let φ be the formula $(P \vee Q) \wedge (R \vee \neg S) \wedge (S \vee P)$. Then $\mu := \{P, Q, \neg R, \neg S\}$ is a model for φ.

Definition 5 (Equivalence under assumptions) *Let ξ and φ be two formulas. Then a formula φ' s.t. $\xi \models \varphi \leftrightarrow \varphi'$ is said to be equivalent to φ under the assumption of ξ.*

Example 2. Let P and Q be predicates. Given the definition of equivalence under assumptions, $P \wedge Q$ is equivalent to P under the assumption $P \to Q$ since $P \to Q \models (P \wedge Q) \leftrightarrow P$. There are other formulas which will be equivalent, e.g. $P \wedge Q$ is trivially equivalent to itself.

By substituting every fact (a predicate) in a context with a fresh propositional variable (*fact variable*) we obtain the *boolean abstraction* of a context. In the same way, we can obtain the boolean abstraction of the assumptions which are known to be true in a context. Given the boolean abstraction for a context φ and the corresponding assumptions ξ we can express the problem of reducing redundancy of contexts as the problem of finding an equivalent context φ' which is equivalent to φ under the assumptions ξ.

In order to obtain such a φ', we exploit SAT solvers, and in particular techniques for generating all the models of a boolean formula. The pseudo-code of our algorithm is reported in Fig. 4. The algorithm enumerates the models of the boolean abstraction φ of the context, and for each such model μ it checks whether μ is compatible with the assumptions ξ (which express the known implications between facts). If μ is compatible with the assumptions, the algorithm tries to reduce μ by removing literals from it as long as it is still a model for φ *under the given assumptions*, that is, as long as $\mu \wedge \xi \models \varphi$, or in other words as long as $\mu \wedge \xi \wedge \neg\varphi$ is unsatisfiable. Then, the reduced context is given by taking the disjunction of all the reduced models that are compatible with the assumptions.

Theorem 1. *Let ξ and φ be two formulas, and let φ' be the result of applying the algorithm of Fig. 4 to φ and ξ. Then φ' is equivalent to φ under the assumptions ξ.*

[1] We define ν only for the connectives $\neg, \wedge$ since they are enough to express all the others.

[2] Moreover, we shall drop the subscripts $_S$ and $_F$ when they are clear from the context.

Input: context φ, assumptions ξ
Output: reduced context φ'
1: $\varphi' \leftarrow \bot$
2: **for all** models μ of φ **do**
3: **if** Is_Satisfiable($\mu \wedge \xi$) **then**
4: **for all** literals $l \in \mu$ **do**
5: $\mu' \leftarrow \mu \setminus \{l\}$
6: **if not** Is_Satisfiable($\mu' \wedge \xi \wedge \neg\varphi$) **then**
7: $\mu \leftarrow \mu'$
8: **end if**
9: **end for**
10: $\varphi' \leftarrow \varphi' \vee \mu$
11: **end if**
12: **end for**
13: **return** φ'

Fig. 4. Pseudo-code of the context reduction algorithm

Proof. We have to show that:

1. every model of φ that is compatible with ξ is also a model of φ'; and
2. for each model μ of φ', all its extensions to all the variables in $V(\varphi) \setminus V(\varphi')$ that are compatible with ξ are models of φ.

– Let μ be a model of φ compatible with ξ (that is, $\mu \wedge \xi$ is satisfiable). Then, by construction (lines 4-10 of the algorithm) φ' contains a subset of μ as a disjunct. Therefore, μ is a model for φ'.
– Let μ be a model of φ' compatible with ξ. Since φ' is a disjunction of conjunctions of literals, μ must be a superset of the set of literals σ in one of such conjunctions. We can assume w.l.o.g. that σ is the smallest such set, because clearly if μ satisfies $\psi \wedge l$, then μ satisfies also ψ. Moreover, the variables occurring in μ are a subset of the variables of φ. Consider any extension μ' of μ to all the variables of φ, such that μ' is compatible with ξ, and suppose that μ' is not a model for φ. Then μ' can be turned into a model for φ by flipping [3] some of the literals in $\mu' \setminus \sigma$, since by construction the literals in σ occur in a model of φ compatible with ξ (lines 3-10 of the algorithm). Let η be a minimal set of literals to flip to obtain a model μ'' of φ from μ'. By construction, $\mu'' \wedge \xi \wedge \neg\varphi$ is unsatisfiable, and for all the literals l in η, $(\mu'' \setminus \{l\}) \wedge \xi \wedge \neg\varphi$ is satisfiable. [4] But then, none of the literals in η would have been removed from μ'' by the algorithm (lines 4-9) when processing μ'' (which must have been processed since it is a model of φ), and so η must be a subset of σ, which is a contradiction. Therefore, μ' is a model for φ.

Example 3. Let the context be $\varphi = (P \wedge Q) \vee R$, and we wish to reduce this formula under the assumption $\xi = (P \rightarrow \neg Q) \wedge (P \rightarrow R)$

To obtain a reduced context we can enumerate all models of φ and reduce given the assumption in this way:

[3] Flipping a literal here means replacing l with $\neg l$ or vice versa.
[4] Because $((\mu'' \setminus \{l\}) \cup \{\neg l\}) \wedge \xi \not\models \varphi$, so $((\mu'' \setminus \{l\}) \cup \{\neg l\}) \wedge \xi \wedge \neg\varphi$ is satisfiable, and so also $(\mu'' \setminus \{l\}) \wedge \xi \wedge \neg\varphi$ is satisfiable.

1. We set up the algorithm by setting $\varphi' \leftarrow \bot$
2. φ is satisfiable, and the first model returned is e.g. $\mu = \{\neg P, \neg Q, R\}$
   a) $\neg P \wedge \neg Q \wedge R \wedge ((P \rightarrow \neg Q) \wedge (P \rightarrow R))$ is satisfiable (line 3), so the model is compatible with the assumptions.
   b) Since $R \wedge (P \rightarrow \neg Q) \wedge (P \rightarrow R) \wedge \neg((P \wedge Q) \vee R)$ is unsatisfiable, both $\neg P$ and $\neg Q$ are redundant in this model, so they are removed from μ' in lines 4-9 of the algorithm.
   c) Update $\varphi' \leftarrow R$
3. the second model of φ returned is e.g. $\mu = \{P, Q, \neg R\}$
   a) As $P \wedge Q \wedge \neg R \wedge ((P \rightarrow \neg Q) \wedge (P \rightarrow R))$ is unsatisfiable (line 3), the model is not compatible with the assumptions, so we skip lines 4-10.
4. the other two models returned are $\mu = \{P, \neg Q, R\}$ and $\mu = \{\neg P, Q, R\}$. As above, they can be reduced to $\{R\}$ only, since $R \wedge (P \rightarrow \neg Q) \wedge (P \rightarrow R) \wedge \neg((P \wedge Q) \vee R)$ is unsatisfiable (line 6).

The resulting reduced context becomes $\varphi' = R$, and we have found that P and Q are redundant for this context.

We remark that the above algorithm can be used also to detect inconsistent or trivial contexts: in the former case, none of the models will be compatible with ξ, so φ' will be always equal to $\bot$; in the latter case, $\xi \wedge \neg\varphi$ will be always unsatisfiable, so in the loop of lines 8-10 all the literals would be removed from μ, which will therefore be reduced to $\top$. However, for efficiency reasons it might be preferable to check for inconsistency and triviality before entering the main loop of lines 2-12, by checking the unsatisfiability of the formulas $\xi \wedge \varphi$ and $\xi \wedge \neg\varphi$ respectively.

Efficiency of the algorithm The algorithm enumerates all models, and in the worst case there are an exponential number of them. For each model, we solve a number of SAT problems. So in the worst case, we need to solve an exponential number of NP-complete problems.

Despite this, the cost of the calls to a SAT procedure can be greatly reduced by using an incremental SAT solver such as MiniSat [9]. The call on line 3 will use the same formula ξ in every iteration of the outer loop, only varying the model μ. In this case, one single solver instance containing ξ can be reused from one iteration to the next. In the same way, the call on line 6 uses the same formula $\xi \wedge \neg\phi$ in each call, only varying the model μ. A single SAT solver instance can be reused for all these calls. The advantage of using an incremental SAT solver for each of these three cases is that everything learnt from the formulas in one iteration of the outer loop can be reused for all following iterations and will not have to be rediscovered. Lastly, enumerating all models can be done with an efficient algorithm for the all-SAT problem.

Further optimizations are possible. E.g. the number of iterations in the loop enumerating all models on line 2 can be reduced by *blocking clauses* gained from the reduction. We can conjunct the negation of the reduced model μ computed on lines 4–9 to the formula ϕ after each iteration. In example 3 above, this improvement would remove the two last iterations.

4.1 Greedy Strategies for Cost Reduction

In the problem of reducing contexts, we wish to remove redundant facts from the context. This corresponds to producing a formula φ' with less variables than φ. In fact, we want to reduce the *cost* of monitoring facts in a context. If we associate a cost (a real number) to each fact variable in the boolean abstraction of a context, our aim is that of finding a φ' such that the sum of the costs of the variables occurring in φ' is smaller than the sum of the costs of the variables occurring in φ.

As presented, our context reduction algorithm does not take costs into account. One simple possibility to make it aware of costs is to apply some *greedy* strategies when determining the order in which variables are eliminated from the current model (line 4 of Fig. 4). For example, one strategy could be to sort the variables in the model μ according to their cost, to try to eliminate more expensive variables first. A more sophisticated strategy could also consider whether a variable already occurs in the current φ' constructed so far, to try to keep the set of variables $V(\varphi')$ as small as possible.

Example 4. Consider the following context formula φ and assumptions ξ:

$$\varphi = ((\neg P \vee \neg R) \wedge (\neg Q \vee \neg S)) \vee (\neg P \wedge S)$$
$$\xi = (P \leftrightarrow Q) \wedge (R \leftrightarrow S) \wedge (S \rightarrow Q)$$

Suppose that the cost of P is 3, that of Q is 1, that of R is 5 and that of S is 4. Moreover, suppose that the first model found by the algorithm of Fig. 4 that is compatible with ξ is $\mu_1 = \{P, Q, \neg R, \neg S\}$.

If the algorithm does not consider costs, μ_1 might get reduced in lines 4-9 to $\{\neg R\}$. Therefore, after the first iteration of the loop of lines 2-12, $\varphi' = \neg R$. The only other model of φ that is compatible with ξ is $\mu_2 = \{\neg P, \neg Q, \neg R, \neg S\}$. In this case, μ_2 might get reduced to $\{\neg P\}$, and thus the resulting reduced context φ' would be $\neg P \vee \neg R$, whose cost is 8.

However, if costs are considered, in the process of reducing μ_1 and μ_2 the algorithm would try to eliminate first the more expensive variables, resulting in the reduced models $\{\neg S\}$ and $\{\neg Q\}$ respectively. Therefore, in this case the reduced context φ' would be $\neg S \vee \neg Q$, whose cost is 5.

Finally, if the algorithm takes into account also the presence of variables in the current φ', in the process of reducing μ_2 the order in which literals are processed in the loop of lines 4-9 would be $\neg R, \neg P, \neg Q, \neg S$, as S is already in φ' (because of μ_1). With this order, also μ_2 would be reduced to $\{\neg S\}$, and so in this case the final φ' would be $\neg S$, whose cost is only 4.

Efficiency of the algorithm The algorithm has the same complexity as the algorithm without costs, since we are only modifying the order in which we try to eliminate variables. We can therefore expect similar performance.

4.2 Finding an Optimal Solution and Reducing Multiple Dependent Contexts

The algorithm of Fig. 4 (and its greedy variant) computes *one* reduction for the input context formula, but it does not find (in general) the reduction with minimum cost.

Input: context φ, assumptions ξ
Output: reduced context φ'
1: $\varphi' \leftarrow \varphi$
2: **for all** subsets S of variables $V(\varphi)$ **do**
3: **for all** formulas $\psi(S)$ over S **do**
4: **if** $\xi \models \psi(S) \leftrightarrow \varphi$ **then**
5: **if** cost of $\psi(S)$ is lower than cost of φ' **then**
6: $\varphi' \leftarrow \psi(S)$
7: **break**
8: **end if**
9: **end if**
10: **end for**
11: **end for**
12: **return** φ'

Fig. 5. Naive algorithm for finding the context with minimum cost

Clearly, finding such optimal context wrt. costs would be very desirable. However, solving this problem is far from trivial. A naive algorithm/solution for it is shown in Fig.5.

This algorithm works by enumerating up to *all* the formulas $\psi(S)$ that are equivalent to the context formula φ under the assumptions ξ, and picking the one with minimum cost. Such exhaustive enumeration is prohibitively costly: the outer loop of lines 2-10 is executed $2^{|V(\varphi)|}$ times, and, since the number of different boolean formulas over k variables is 2^{2^k}, the inner loop of lines 3-9 is executed up to $2^{2^{|S|}}$ times. Moreover, checking whether $\psi(S)$ and φ are equivalent under the assumptions ξ (line 4) is an NP-complete problem. Therefore, the naive algorithm would require to solve up to $\sum_{S \in 2^{V(\varphi)}} 2^{2^{|S|}}$ NP-complete problems.

In practice, the algorithm can be improved by performing a branch-and-bound search [13] (on the sum of costs of the variables) instead of enumerating all the subsets of variables, thus avoiding to enumerate (and check) formulas over variables whose cost is known to be higher than the best solution found so far. However, in the worst case the complexity would not improve.

5 Discussion and Future Work

In this paper, we have proposed a framework to optimize monitoring requirements, so as to minimize the monitoring infrastructure a system has to deploy. We have presented the context dependency problem, which may lead to trivial, redundant, and inconsistency in monitoring requirements, and proposed algorithms to detect such problems as well as to suggest ways to fix redundancies. In our approach, monitoring requirements are identified as a result of contextual goal analysis. First, relevant contexts are identified by means of contextual goal models [2]. Second, the contexts in such models are analysed, via context analysis technique, in order to identify monitorable facts, which constitute the monitoring requirements. Such requirements can suffer from different problems; they can be (i) redundant, if the monitoring infrastructure is going to observe more data than necessary; (ii) inconsistent, if a context to be monitored is always false;

(iii) trivial, if a context to be monitored is always true. In order to detect these problems and to fix them, by producing an equivalent monitoring requirements specification with reduced cost, we propose techniques based on state-of-the-art SAT-solvers. Research on monitoring requirements (and their optimization) is still at an early stage. To the best of our knowledge, ours is the first approach that argues for the importance of optimizing monitoring requirements in adaptive systems by detecting redundancies, inconsistencies, and trivialities. Thus, we will also consider literature on requirements monitoring and contextual requirements, which provides useful insights for our research.

Salifu et al. [16] have clarified the importance of considering monitoring requirements. In particular, they relate monitoring requirements to what an application shall monitor in order to check whether a requirement has failed or is being met. They consider the existence of alternative means to perform monitoring, and the choice among such alternatives. Our approach, instead, focuses on minimizing the amount of information to be monitored: this is a particularly important concern when considering mobile and adaptive systems with a large space of variations (alternatives).

Requirements monitoring is about insertion of a code into a running system to gather information (mainly about computational performance), so as to determine whether the running system is meeting its design objectives and reconcile the system behaviour to requirements if a deviation occurs [11]. In [10], the GORE (Goal-Oriented Requirements Engineering) framework KAOS [8] is integrated with an event-monitoring system (FLEA [6]) to provide an architecture that enables the runtime automated reconciliation between system goals and system behaviour with respect to a priori anticipated or evolving changes of the system environment. Their work does implicitly specify monitoring requirements; thus, our approach could be applied to their modelling framework in order to minimize the monitoring infrastructure to be deployed.

Wang et al. [17] propose an approach to requirements monitoring based on goal models, wherein tasks and goals are associated with pre- and post-condition. A failure occurs if (i) a post-condition is met while the respective pre-condition does not hold, or (ii) an event representing a precondition occurs and at the next time-step the postcondition does not hold. They argue for and show the importance of requirements monitoring trade-off, especially when their approach is applied to multi-layer monitoring, e.g. in service-oriented architectures. They conduct experiments on how different granularity of monitoring affects the accuracy of the diagnosis. Our approach focuses on minimizing the amount of contextual information to monitor, while leading to optimal diagnosis.

Baresi et al. [5] propose Dynamo, an approach that provides dynamic monitoring in web services. In their framework, monitoring requirements are specified by an analyst an monitorable rules. Dynamic monitoring is provided by the monitoring manager component: depending on the current context, the component decides whether a rule is to be monitored or not. Like the previously mentioned approaches, however, they do not focus on identifying a minimal monitoring infrastructure and contextual information which do not sacrifice the system ability of taking correct adaptation decisions.

In future work, we aim to develop a supporting tool for our framework that assists the analysts for building correctly our proposed models and simulating the system behaviour. We plan to integrate our techniques in frameworks for requirements-driven self-adaptive software; a good candidate is the work by Dalpiaz et al. [7], which already captures the relation between context and requirements. The role of users in monitoring

a system's environment and quality is being recognized through the concepts of Social Sensing [4] and Social Adaptation [3]. We also plan to allow users to report meta-data about the data they are asked to monitor so we can further optimize monitoring requirements. Moreover, we will apply our framework to case studies to understand how our optimization techniques save costs in practice and reduce their complexity.

Acknowledgement The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grants n o 257930 (Aniketos), 256980 (Nessos), and 258109 (FastFix), and Science Foundation Ireland under grant 10/CE/I1855 and also from Provincia Autonoma di Trento and the European Community's FP7/2007-2013 under grant agreement Marie Curie FP7 - PCOFUND-GA-2008-226070 "progetto Trentino", project ADAPTATION.

References

1. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A Goal Modeling Framework for Self-Contextualizable Software. In *Proceedings of EMMSAD'09*, pages 326–338, 2009.
2. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A Goal-based Framework for Contextual Requirements Modeling and Analysis. *Requirements Engineering*, 15:439–458, 2010.
3. Raian Ali, Carlos Solis, Inah Omoronyia, Mazeiar Salehie, and Bashar Nuseibeh. Social Adaptation: When Software Gives Users a Voice. In *Proceedings of ENASE'12*, 2012.
4. Raian Ali, Carlos Solis, Mazeiar Salehie, Inah Omoronyia, Bashar Nuseibeh, and Walid Maalej. Social Sensing: When Users Become Monitors. In *Proceedings of ESEC/FSE '11*, pages 476–479, 2011.
5. Luciano Baresi and Sam Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *proceedings of ICSOC 2005*, number 3826 in LNCS, pages 269–282. Springer, 2005.
6. Don Cohen, Martin S. Feather, K. Narayanaswamy, and Stephen S. Fickas. Automatic Monitoring of Software Requirements. In *Proceedings of ICSE'97*, pages 602–603, 1997.
7. Fabiano Dalpiaz, Paolo Giorgini, and John Mylopoulos. Adaptive Socio-Technical Systems: a Requirements-driven Approach. *Requirements Engineering*, 2012. To appear.
8. Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.
9. Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, pages 333–336. Springer, 2004.
10. Martin S. Feather, Stephen S. Fickas, Axel Van Lamsweerde, and Cristophe Ponsard. Reconciling System Requirements and Runtime Behavior. In *Proceedings of the 9th international workshop on Software specification and design (IWSSD'98)*. ACM, New York, USA, 1998.
11. Stephen S. Fickas and Martin S. Feather. Requirements Monitoring in Dynamic Environments. In *Proceedings of RE'95*, page 140. IEEE Computer Society, 1995.
12. Anthony Finkelstein and Andrea. Savigni. A Framework for Requirements Engineering for Context-Aware Services. In *Proceedings of STRAW'01*, 2001.
13. Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Operations Research*, volume 6. McGraw-Hill New York, NY, 1990.
14. John Krogstie, Kalle Lyytinen, Andreas Lothe Opdahl, Barbara Pernici, Keng Siau, and Kari Smolander. Research Areas and Challenges for Mobile Information Systems. *International Journal of Mobile Communications*, 2(3):220–234, 2004.
15. Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4:14:1–14:42, May 2009.
16. Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh. Specifying Monitoring and Switching Problems in Context. In *Proceedings of RE'07*, pages 211–220, 2007.
17. Yiqiao Wang, Sheila McIlraith, Yijun Yu, and John Mylopoulos. Monitoring and Diagnosing Software Requirements. *Automated Software Engineering*, 16(1):3–35, 2009.