UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38123 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it

A Goal-based Framework for Contextual Requirements Modeling and Analysis

Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini

February 2010

Technical Report # DISI-10-019

Noname manuscript No.
(will be inserted by the editor)

A Goal-based Framework for Contextual Requirements Modeling and Analysis

Raian Ali · Fabiano Dalpiaz · Paolo Giorgini
DISI, University of Trento, Italy.

Received: date / Accepted: date

Abstract Requirements Engineering (RE) research often ignores, or presumes a uniform nature of the context in which the system operates. This assumption is no longer valid in emerging computing paradigms, such as ambient, pervasive and ubiquitous computing, where it is essential to monitor and adapt to an inherently varying context. Besides influencing the software, context may influence stakeholders' goals and their choices to meet them. In this paper, we propose a goal-oriented RE modeling and reasoning framework for systems operating in varying contexts. We introduce *contextual goal models* to relate goals and contexts; *context analysis* to refine contexts and identify ways to verify them; reasoning techniques to derive requirements reflecting the context and users priorities at runtime; and finally, design time reasoning techniques to derive requirements for a system to be developed at minimum cost and valid in all considered contexts. We illustrate and evaluate our approach through a case study about a museum-guide mobile information system.

Keywords Contextual Requirements · Context Analysis · Goal Modeling · Requirements Analysis

1 Introduction

The advances of computing, sensors, and communication technology helped the realization of new computing paradigms such as ambient, ubiquitous and pervasive computing. These paradigms weave computing systems with humans' living environments to transparently meet their needs [1]. Context, a core element of these settings, can be defined as the reification of the environment, that is whatever provides a surrounding in which a system operates [2]. Context can influence the requirements of a system and the variants a system can adopt to meet its requirements. Moreover, context is by nature variable in these paradigms and it calls for new approaches to create system that can adapt to context changes.

Goal-oriented analysis has been proposed in the RE literature to capture the intentionality behind software requirements [3]. Goals are a useful abstraction to represents stakeholders' needs and expectations and they offer a very intuitive way to elicit and analyze requirements. Context is strongly related to goals, for it changes the current goals of a stakeholder and the possible ways to satisfy them. For example, let us consider a tour guide that has the goal of providing assistance to tourists during an organized tour. The context "tourist has not had lunch today and now it is lunchtime" activates the guide's goal "find a restaurant" and, supposing that a context like "tourist is vegetarian" applies, the guide has to find a restaurant serving vegetarian food. A software system developed to support tour guides has to reflect guides' goals, their rationale and their capability to adapt to the context. This reflection is preliminary for the system to execute useful functionalities such as showing on the map a set of close restaurants that serve vegetarian food.

Goal models (i^* [4], Tropos [5], and KAOS [6]) represent an intentional ontology used at the early requirements analysis phase to explain the *why* of a software system. They have been used to represent the rationale of both humans and software systems [7] and they provide useful constructs to analyze high level goals and ways to satisfy them. Such features are essential for the analysis and the design of a software system supposed to reflect stakeholders' rationale and adaptation to varying contexts [8,9].

In this paper, we propose a RE modeling and reasoning framework for systems operating in and reflecting varying contexts. We propose the *contextual goal model* that extends Tropos goal model introducing variation points where the context may influence the choice among the available variants of goals satisfaction [10, 11]. We also propose a set of modeling constructs to analyze and discover relevant information the system needs to capture, at runtime, in order to verify if a context applies [12,13]. Two reasoning techniques are then proposed. The former concerns the automatic derivation at runtime of goal model variants that reflect context and user priorities. The latter is about the derivation, at design time, of the requirements to be implemented that lead for a system developed with minimum costs and valid in all considered contexts. We illustrate and evaluate our framework through a mobile information system case study.

The paper is structured as follows. Section 2 describes the museum-guide case study; Section 3 introduces the contextual goal model to capture requirements for varying contexts; Section 4 illustrates the reasoning technique to derive requirements for different contexts and users' priorities, while Section 5 shows an approach to derive the core requirements of a system. Section 6 presents our developed automated support tool and the results obtained by applying our framework on the case study. Related work and conclusion are given in Section 7 and Section 8, respectively.

2 Case Study: Museum-guide

In this paper, we use a case study of a museum-guide mobile information system developed within the Laboratory of Mobile Application (LaMA[1]) at the University of Trento. The system is expected to enforce the museum rules by notifying visitors to what they should do in the right moment. Moreover, the system has to figure out if the visitor is interested in a certain piece of art and convey suitable information related to that piece of art. Visitors and museum staff are provided with PDAs as communication and explanation devices. The system consists basically of two components: the monitoring component that captures context, and the functional component that carries out actions reflecting each monitored context.

To initiate the process of conveying information about a piece of art to a visitor, the system has to monitor if the visitor is interested in it. This information can be inferred, for instance, if the visitor has been standing in front of the piece of art for long time. If so, the system has to look for the best way to convey information to the visitor. The delivery of information can be done via information terminals, the PDA the visitor has, or a staff member. For each of the possible ways to convey information, the system is supposed to do certain tasks. For example, to use terminals the visitor must be informed about the existence of such a service, guided to it, and informed about the way to use it. To get information through a staff member, the system has to notify the staff member and establish a call with the visitor, or guide the staff to the visitor's place to give information in person.

Concerning the relationship between context and requirements, context can influence decisions about:

- **Requirements to meet**: if the context "visitor is not interested in a piece of art" applies, the mobile information system does not need to activate the information delivery process. Moreover, if the context "visitor is familiar with the use of terminals and knows one of the languages the terminals support" applies, then informing the visitor about the way of using such terminals is not required and the system has only to inform the visitor about the existence of the service and guide him to a free terminal.
- **Ways to meet requirements**: the system could have two variants to convey information about a piece of art via PDAs: video-based and interactive. Each variant could require a valid context. For example, conveying information via an interactive presentation requires that a context like "visitor has good experience in using PDAs" applies.
- **Quality of each way**: considering staff comfort as a quality measure; conveying information to visitors on person is less comfortable for a staff when a context like "visitor is far away from the staff" applies.

3 Goal-based Contextual Requirements

Goal-oriented approaches have gained popularity in RE community for their simplicity in capturing stakeholders' needs and expectations. Goal models are intentional representations of users goals and the ways users may adopts to satisfy them. Goal models can capture also the quality of each way through the notion of softgoal [3]. Context may have a strong influence on users goals, the way to reach them, and the quality of each way. Consequently, we need to enrich goal models with context to capture such influence. In this section, we propose the *contextual goal model* that accommodates the relation between goals and context.

[1] http://lama.disi.unitn.it/

3.1 Tropos goal modeling: overview

Goal analysis represents a paradigmatic shift with respect to object-oriented analysis. While object-oriented analysis fits well to the late stages of requirement analysis, goal-oriented analysis is more natural for the earlier stages where organizational goals are analyzed to identify and justify software requirements and position them within the organizational system [7]. Tropos goal analysis projects the system as a set of interdependent actors, each having its own strategic interests (*goals*). Goals are analyzed iteratively in a top-down manner, to identify more specific sub-goals needed to reach the upper-level goals. Goals can be ultimately satisfied by means of specific executable processes (*tasks*).

In Fig.1, we show a partial Tropos goal model for the museum-guide case study. Actors (*"Visitor assistance system"* and *"Staff assistance system"*) have a set of top-level goals (*"visitor gets informed about a piece of art"*), which are iteratively decomposed into subgoals by and-decomposition (all subgoals should be achieved to fulfil the top goal) and or-decomposition (at least one subgoal should be achieved to fulfil the top goal – e.g., *"visitor gets information via his PDA"* or *"visitor gets information through museum staff"*. Goals are finally satisfied by means of executable tasks; the goal *"piece of art information is presented to visitor"* can be reached by one of the tasks *"information is presented to visitor via video"* and *"information is presented to visitor interactively"*.

A dependency indicates that an actor (*depender*) depends on another actor (*dependee*) to attain a goal or to execute a task: the actor *"Visitor assistance system"* depends on the actor *"Staff assistance system"* for achieving the goal *"visitor gets info through museum staff"*. Softgoals are qualitative objectives for whose satisfaction there is no clear-cut criteria (*"staff is more comfortable"* is a rather vague objective), and they can be contributed either positively or negatively by goals and tasks: *"staff gives info to visitor in person"* usually contributes negatively to *"staff is more comfortable"*, while *"staff gives info to visitor remotely"* usually contributes positively to it.

Goal analysis allows for different variants to satisfy a goal, but does not specify explicitly when each variant can be adopted. Supporting variants without specifying when to follow each of them raises the question *"why does the system support several variants and not just one?"*. The system may support different variants to goal satisfaction in order to be able of operating in varying contexts. In the next section we specify the relationships between such variants and context through the contextual goal model.

3.2 Context in requirements

Context has been defined in multiple computer science disciplines especially in artificial intelligence (for a survey see [14]). It has been also defined in the literature of emerging computing paradigms, such as ubiquitous, adaptive, and mobile systems [15,2,16], that our requirements engineering framework is developed for. A specific definition of context strongly depends on the domain it is used in. For example, in a context sensitive search engines, a user may search the term "java" that could mean a programming language or an island. To disambiguate the searched term, the engine may look to the context that can be the query history. If the user asked recently for the term "cgi programming", then most probably he is looking for the Java programming language [17]. In the rest of this section, we adapt a definition of context from the perspective of requirements engineering, namely goal-oriented requirements engineering.

As widely accepted, software is a means to meet user requirements [18,19,7,20]. Software is developed to solve a problem in the users world and to help them reach their goals. In line with this view of requirements, Tropos requirements analysis projects a system, either organizational or software, as a set of interdependent actors. Each actor has goals which are partial states of the world an actor attempts to reach. Tropos goal analysis represents alternative sets of tasks that an actor may execute trying to reach its goals. In other words, tasks are not required *per se*, but are means to reach goals. Actors are autonomous in deciding what goals to reach, how, and how well to reach them. We here give a definition of actor, adapted from [5], that is going to be the observer of a context:

Definition 1 (Actor) *an actor is an entity that has goals and can decide autonomously how to achieve them.*

An actor can be of different types such as human actors, software actors, or organizational actors. The main characteristic of an actor is the autonomy in deciding the way to reach its goals. This includes the ability to decide what goals to reach, how, and how well to reach them. For example, an assistance staff is a human actor that may have the goal of conveying appropriately information about pieces of art to visitors. The assistance staff has the ability to decide when to activate this goal and what to do to reach it. The staff may reach such goal by making a phone call with the visitor or by delivering information to him in person and the decision between these two options is left to the assistance staff himself. The decision taken by an actor depends on the state of a portion of the world such actor lives in. We call such a state *context*:

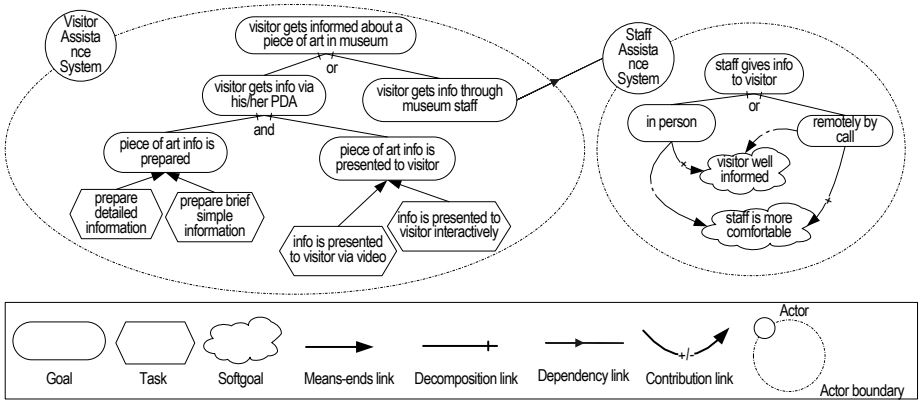


Fig. 1 Tropos goal model example

Definition 2 (Context) *a context is a partial state of the world that is relevant to an actor's goals.*

The decision about the parts of the world that are relevant to an actor decisions is of subjective nature. An actor does not observe the world for the purpose of observation *per se*. An actor does that to decide what goals to reach and what actions to do to reach them. Therefore, such decision is influenced by properties over the world that an actor needs to observe. For example, "visitor is in a room where taking pictures is forbidden" is relevant for a visitor assistance system actor when deciding whether to block his PDA camera. The same context is irrelevant when this actor needs to decide if to convey information about a piece of art. Moreover, there could be always viewpoints about what parts of the world are relevant to a decision. For example, to decide the adoptability of conveying information to a visitor via an information terminal, one staff assistance attempts to verify the context "visitor is very close to a free terminal" and another one may attempts to verify "visitor is close to a terminal or to a map showing the locations of terminals in the museum".

Context is inherently partial and volatile. Actors may have partial view of the state of the world. They may not be interested or able to capture all the information that fully captures such a state. A state of the world may be partitioned into dimensions such as spatio-temporal, personal, tasks, social as proposed in [16]. This partitioning is a way of facilitating the way a state of the world can be described and captured. The world is volatile and could be in different states. A partial state of the world that is uniform does not influence the decisions of an actor. For example, if all the museums do not allow taking pictures to the pieces of art then the museum-guide system does not need to observe if a room contains non-pictured piece of art. The decision is made once while developing the system and applied in all museums the system will operate in.

3.3 Contextual goal model

Goal models allow for variants of goal satisfaction. The applicability of each of these variants can be context dependent. The explicit specification of the context where each variant is applicable allows, amongst other things, for a systematic derivation of variants for various contexts. The enumeration of goal model variants and the specification of contexts for each of them separately is obviously a hard and time consuming task because of the potentially huge number of variants and the complexity of each variant when treated as one block. To avoid enumerating the variants, we propose to define context on a set of variation points at the goal model.

Fig. 2 represents a Tropos goal model for the Museum-guide mobile information system which we have already described. To make the model contextual, we need to explicitly represent the relation between its space of variants and the context. To this end, contexts, that are labeled as $C_1..C_{15}$ on Fig. 2, can be associated with the following variation points of Tropos goal model:

1. *Or-decomposition.* The adoptability of a subgoal or a subtask in an Or-decomposition may require a valid context. For example, to provide information about a piece of art, a visitor can be directed to a dedicated terminal. The terminal, however, has to be available and close to the visitor, while the visitor has to be able to use and interact with such a terminal (C_4). Alternatively, the visitor's PDA can be used to convey information when the piece of the art information is not complicated, and the visitor has the ability and knowledge to use PDAs (C_5). Getting information through an assistance staff requires that the visitor is not able to use PDA and

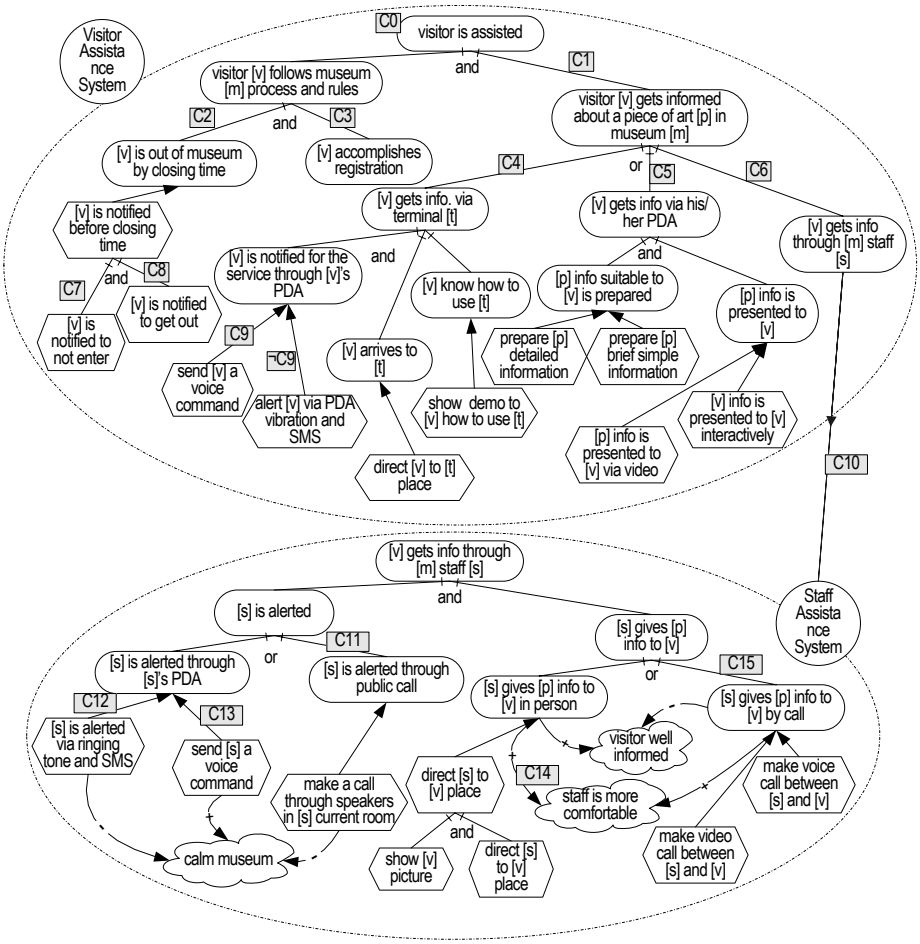


Fig. 2 The goal model of the museum-guide system with context annotation.

not familiar with terminals or that he is classified as an important visitor (C_6). Notifying a museum staff by issuing a special voice message through his room speakers can be adopted if the room does not include audio art contents (C_{11}). The museum staff can give information through a call to visitors when the staff's PDA and the visitor's PDA are not busy (C_{15}).

2. *Root goals* . Depending on the context, an actor may decide to reach a root goal. For example, to reach the root goal *"visitor is assisted"*, the visitor has to be inside the museum area (including parking places and public square in front of museum) and the visitor should have accepted the assistance by the mobile information system (C_0).
3. *Means-end*. Goals can be ultimately satisfied via specific executable processes (*tasks*). The adoptability of each task in means-end analysis might depend on the context. For example, the visitor can be notified about the availability of information terminals through a PDA voice message when he puts the headphones on and is not using his PDA for a call (C_9); while, notifying him by SMS can be adopted in the opposite context ($\neg C_9$). Notifying a museum staff by ringing tone and SMS is adoptable when he is not calling (C_{12}), while notifying him by a PDA voice command is adoptable when he is not calling and is putting the headphone on (C_{13}).
4. *Actors dependency*. An actor can attain a goal or get a task executed by delegating it to another actor only in a specific context. For example, the dependency in Fig. 2 requires an available staff member that talks the language of the visitor and knows enough about the piece of art (C_{10}).
5. *And-decomposition*. A subgoal/subtask in an And-decomposition might (not) be needed only in a certain context, i.e., some subgoals/subtasks are not always mandatory to fulfil the top-level goal/task in an And-decomposition. For example, the subgoal *"visitor gets informed about a piece of art"* has to be reached if the visitor is still inside the gallery building and he is interested in the piece of art (C_1).

The subgoal *"visitor is out of museum by closing time"* needs to be reached when closing time is approaching (C_2), and the goal *"visitor accomplishes registration"* has to be reached when visitor has entered the museum building (C_3). The task *"visitor is notified to not enter"* is needed when the visitor is on the way to enter the museum building (C_7), and the task *"visitor is notified to get out"* is needed when the visitor is still inside the museum building and is not walking towards the exit (C_8).

6. *Contribution to softgoals.* Softgoals are qualitative objectives for which there is no clear cut criteria for their satisfaction. They can be contributed either positively or negatively by goals and tasks. The contributions to softgoals can vary from one context to another. For example, giving the information in person is comfortable to the assistance staff if the visitor is close to him (C_{14}), while it is not comfortable when they are far away from each other.

In the rest of the paper, we use the term **"context of a goal model variant"** to refer to the conjunction of contexts at the variation points of the first five kinds. If the context of a goal model variant applies, this means that the variant is applicable. The contexts associated with contributions to softgoals are used the evaluate the quality of each goal model variant. In Fig. 3, we show a variant of the museum-guide goal model and its corresponding context.

3.4 Context analysis

Similar to goals, context may need to be analyzed. On the one hand, goal analysis allows for a systematic way to discover alternative set of tasks an actor may execute to reach a goal. On the other hand, context analysis should allow for a systematic way in discovering alternative sets of facts an actor may verify to judge if a context applies.

We specify context as a formula of world predicates. The syntax for this formula is shown in Code 1 using the EBNF notation:

Code 1 EBNF grammar for world predicates formulae

```
Formula :- World_Predicate | (Formula) | Formula AND Formula
| Formula OR Formula
```

We classify world predicates, based on their verifiability by an actor, into two kinds, *facts* and *statements*:

Definition 3 (Fact) *a world predicate F is a fact for an actor A iff F can be verified by A.*

Definition 4 (Statement) *a world predicate S is a statement for an actor A iff S can not be verified by A.*

An actor has a clear way to verify a fact. It has the ability to capture the necessary data and compute the truth value of a fact. A fact is not a subject of viewpoints. In other words, when a fact is true for an actor it will be also true for others. For example, world predicates such as "visitor is in the same room as a piece of art", "visitor is in the corridor of the same floor as the piece of art" are facts that the museum guide information system can compute their truth values based on the visitor's location, which can be obtained by a positioning system, and the topology of museum.

Some world predicates are not verifiable by an actor. We call such predicates *statements*. A world predicate can not be verified by an actor for reasons such as:

- lack of information: an actor may be unable to verify a world predicate because of the inability to capture the information necessary to verify it. For example, "visitor does not know about a piece of art" is a statement from the perspective of an actor such as the assistance staff in a museum. The staff can not obtain all the information needed to verify this statement. The staff can not monitor if a visitor has read about the piece of art somewhere on the web or has been told about it by a friend.
- abstract nature: some world predicates are abstract by nature and do not have clear criteria to be evaluated against. For example "visitor is interested in a piece of art" is a world predicate that an actor, such as an assistance staff, has no precise way to judge if it holds and be certain of the judgement. It is a concept that refers to a visitor's mood that there is no way to verify it by an actor rather than the visitor himself.

Some decisions that an actor takes may depend on contexts specifiable by means of only facts, while some other decisions may depend on contexts that include also statements. For example, to decide if to convey information about a piece of art to a visitor via an assistance staff, the system (visitor assistance system) has to judge if the context C_6 applies. This includes deciding the truth of the world predicate wp="visitor is not familiar with information terminals". Such world predicate is a statement that the system can not verify. However, this statement can be refined into a formula of facts and other statements. For example, the refinement could consider the behavior of the visitor while using a terminal. A slow or an unsuccessful interaction between the visitor and a terminal may indicate little familiarity in using such terminals, i.e. indicate the truth of wp. We

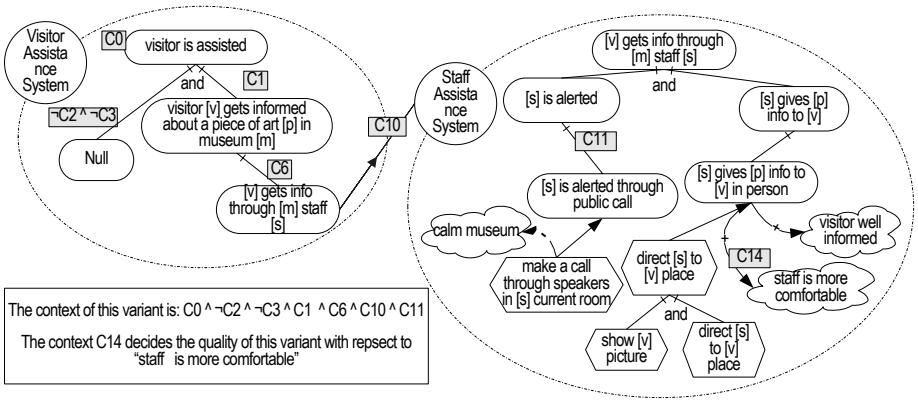


Fig. 3 A variant of the museum-guide goal model and its context.

call the relation between such a formula of word predicates and a refined statement *Support*, and we define it as following:

Definition 5 (Support) *a statement S is supported by a formula of world predicates φ iff φ provides evidence in support of S.*

In an iterative way, a statement could be ultimately refined to a formula of facts that supports it. That is to say, the relation *support* is transitive. If a formula φ_1 supports a statement S_1 and $S_1 \wedge \varphi_2$ supports S_2, then $\varphi_1 \wedge \varphi_2$ supports S_2. However, refining a statement to a formula of facts is not always possible. We may have statements that could be unrefinable to facts. For example, "visitor never visited other similar museums" is a world predicate that can not be verified by an assistance staff due to lack of information. Moreover, the staff would not be able to find a formula of facts that he can verify to support such a statement. In our contextual goal model, we allow only for contexts that are specified by means of facts and/or statements that are supported by facts. We call the kind of statements and contexts that we deal with as *monitorable statements* and *monitorable contexts* and we define them as follows:

Definition 6 (Monitorable Statements) *a statement S is monitorable iff there exists a formula of facts φ that supports S.*

Definition 7 (Monitorable context) *a context C is monitorable iff C can be specified by a formula of facts and monitorable statements*

A monitorable context, specified by a world predicate formula φ, applies if all the facts in φ and all the formulae of facts that support the statements in φ are true.

Context analysis aims to discover if a context is monitorable and to find the formula of facts that specifies it. Context analysis starts with specifying a world predicate formula that represents a context. This formula may contain both facts and statements. For example, taking the context C_1 of the contextual goal model shown in Fig 2, this context can be specified as a formula of world predicates $C_1 = wp_1 \wedge wp_2$ where wp_1="visitor is inside the gallery building" and wp_2= "visitor is interested in getting explanation about a piece of art". Obviously, the world predicate wp_1 is a fact that the system can verify based on obtainable data (position of the visitor can be obtained through a positioning system) while wp_2 is a statement and we need to find if it is refinable into a formula of facts.

To see if a context is monitorable, the statements in the formula specifying that context need to be refined into formulae of facts that support them. A statement can be analyzed iteratively to ultimately discover a formula of facts that an actor can visualize in the world and that gives evidence in support of the analyzed statement. In Fig. 4, we analyze the context C_1. In this figure, *statements* are represented as shadowed rectangles and *facts* as parallelograms. The relation *support* is represented as curved filled-in arrow, and the *and*, *or*, *implication* logical operators are represented as black triangles, white triangles, filled-in arrows, respectively.

As we mentioned earlier, we consider the relation *support* as a transitive relation. For example, as shown in Fig. 4, the formula $w_1 \wedge w_2 \wedge w_3 \wedge w_4$ supports the statement wp_2, the formula $(f_1 \wedge f_2) \vee f_3$ supports the statement w_1, then the formula $((f_1 \wedge f_2) \vee f_3) \wedge w_2 \wedge w_3 \wedge w_4$ supports the statement wp_2. Consequently, a statement may be refined iteratively to reach the level of facts. In the same figure, we show the formula of facts that supports the statement wp_2. The visitor assistance system can verify this formula to judge if wp_2 applies.

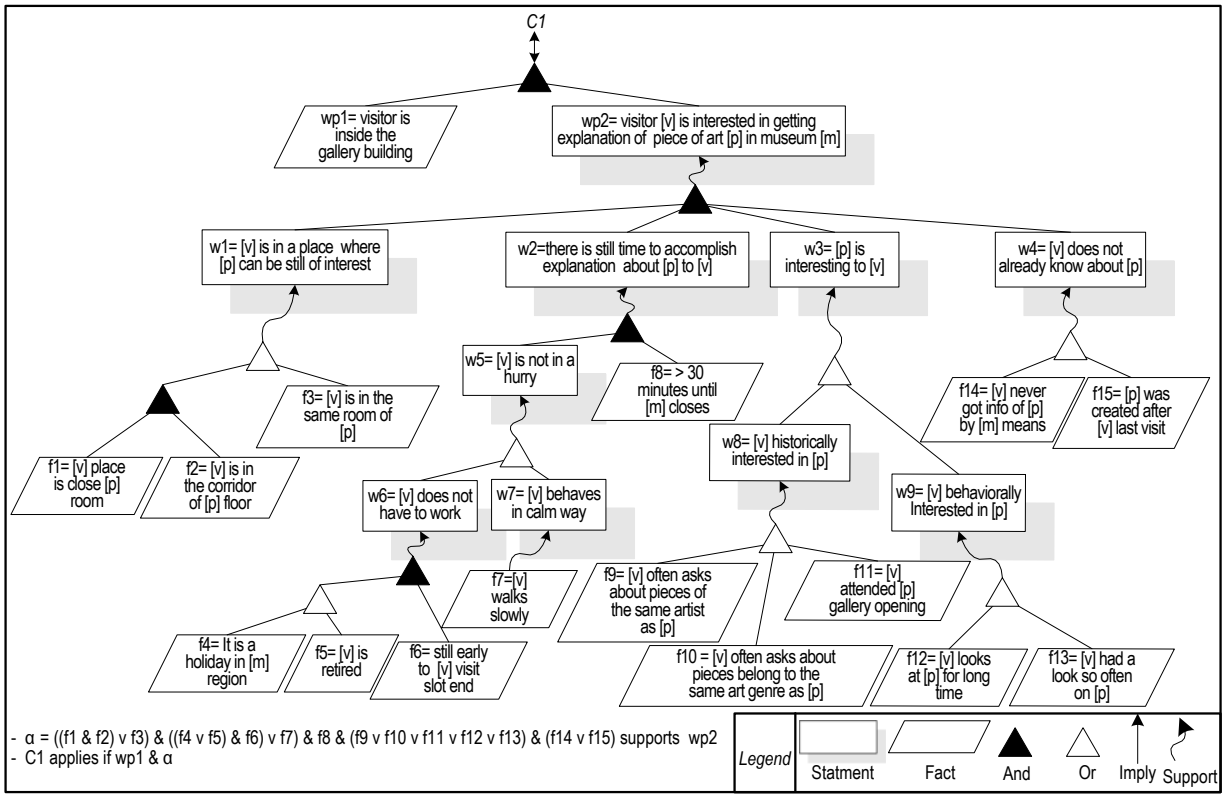


Fig. 4 The context analysis for C_1

Analyzing context allows us to discover what data an actor has to collect of the world. The analysis allows us to identify the facts that an actor has to verify. These facts are verifiable on the basis of data an actor can collect of the world. For example, taking the facts $f_9..f_{13}$ that support the statement w_3= *"piece of art [p] is interesting to visitor [v]"* of Fig. 4 , we could develop a data conceptual model, shown in Fig. 5, that the museum-guide system has to implement and maintain in order to verify facts, judge if the analyzed contexts apply, and take decisions at the corresponding variation points of the goal model.

4 Deriving Requirements in Varying Contexts

Goal models allow for a systematic analysis of variants for goal satisfaction and an implemented system may support all or a subset of them. This is a design decision that can be taken on the basis of different criteria. For example, the designer can decide to minimize the overall development costs and therefore to reduce the number of implemented variants. Alternatively, the designer may decide to make the system flexible and highly variable, that will require a much higher number of variants to be implemented [21]. In any case, each variant can be applicable in certain contexts and the system has to implement runtime mechanisms to decide which variant to adopt when more than one variant is applicable in the actual context. For this decision, users' prioritization over goal model variants can be an effective criteria to be used at runtime. However, specifying such prioritization introduces two main problems at the analysis phase:

- the potentially huge number of goal model variants, i.e., specifying prioritization over the enumerated variants could be extremely time consuming.
- when the variants contain a large number of nodes, it could be hard for users to comprehend the variants and the differences between them.

Instead of asking users to specify their prioritization over variants, prioritization can be expressed over the quality measures, i.e., softgoals. Users can express prioritization on softgoals and bypass the large number of goal model variants. Besides this advantage, softgoals allow users to express their prioritization using their own terms. For example, users can easily specify that *"more comfort"* has high priority while *"less disturbance"* is not such important. The quality contexts of a variant are those on the contribution links between the goals/tasks of that variant and softgoals. The truth value of quality contexts determines the quality of each variant.

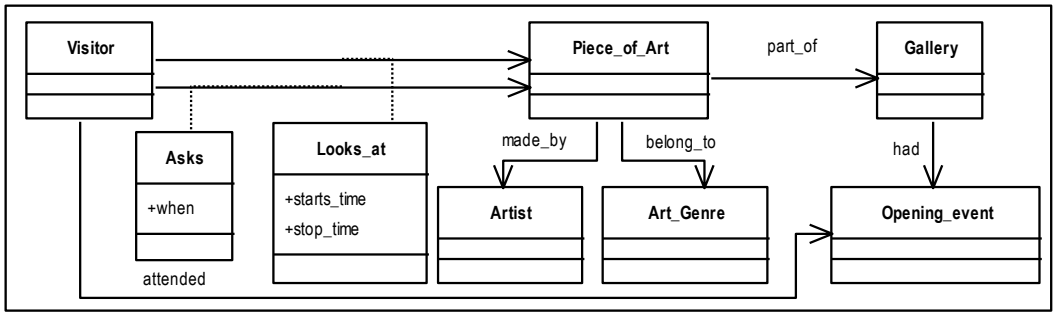


Fig. 5 The conceptual model of data needed to verify W_3 leaf facts

We adopt an approach similar to the one proposed in [22] to specify prioritization over softgoals. We consider binary contributions to softgoals (positive or negative). Stakeholders can specify the priority of each softgoal by selecting an integer in the range [0,n]. Priority 0 corresponds to *"the user does not care about the softgoal"*, priority n means *"the user considers the soft-goal very important"*. The average contribution value for a variant v to a softgoal sg_i is computed as the difference between the number of positive and negative contributions from tasks/goals of v to sg divided by the total number of contributions in v:

$$avg(sg, v) = \frac{|contr(sg, pos)| - |contr(sg, neg)|}{|contr(sg, pos)| + |contr(sg, neg)|}$$

The priority of a variant v is computed as the weighted sum of the average contributions divided by the sum of the priorities of that variant's softgoals:

$$priority(v) = \frac{\sum_{sg_i \in v} \ avg(sg_i, v) \times priority(sg_i)}{\sum_{sg_i \in v} priority(sg_i)}$$

The formulae that define *avg* and *priority* are expressed as ratios, in order to normalize possible values within the [-1,+1] range, where -1 and +1 are the worst and the best values, respectively. Consequently, the derivation of goal model variants for a given context and user prioritization is a two steps process that the system follows at runtime:

1. *Deriving the variants applicable in the current context*: the truth values of contexts at the variation points decide the set of goal model variants that are applicable. As we have shown earlier, context analysis allows us to discover a formula of facts that specifies a context (see Fig. 4). The system, at runtime, has to monitor the environment and collect data (Fig. 5) and compute the truth value of the formulae of facts at each variation point of the goal model. This, in turn, filters the space of goal model variants leaving those that are applicable in the current context.
2. *Ranking the applicable variants based on user's prioritization*: at certain contexts, there could be more than one applicable goal model variant. In other words, there could be more than one variant to meet the same requirements. To select between them, user prioritization could be considered by the system at runtime. To this end, users are asked, at design time, to prioritize the set of softgoals. The system computes the value of contextual contributions and the priority of each applicable variant according to the formulae above. The adopted variant is the one with the highest priority, i.e., the one that better contributes to the highly prioritized softgoals.

Example 1 Suppose that the current context allows for the two variants partially shown in Fig. 6. The system has the possibility to guide a staff to meet a visitor in person (variant V') or the possibility to establish a call between them so as to communicate remotely (variant V''). Delivering information in person to a visitor contributes negatively to the softgoal *"staff feels more comfortable"*, as the staff is not close to the visitor (presuming that C_{14} is false), and positively to the softgoal *"visitor is well-informed"*. The second variant, delivering the information by a remote call, contributes conversely to the two mentioned softgoals. If a stakeholder, such as the administration of the museum, gives staff comfort a priority higher than the quality of information delivered to visitors, then the variant V' would be adopted, and vice versa.

5 Deriving Requirements for Minimum Development Costs

In the previous section, we have studied the derivation of goal model variants for a given context and user priorities. Such reasoning is of high importance for systems that support multiple goal model variants and where more than one variant is adoptable in certain contexts. On the other side, and for reasons such as budget and

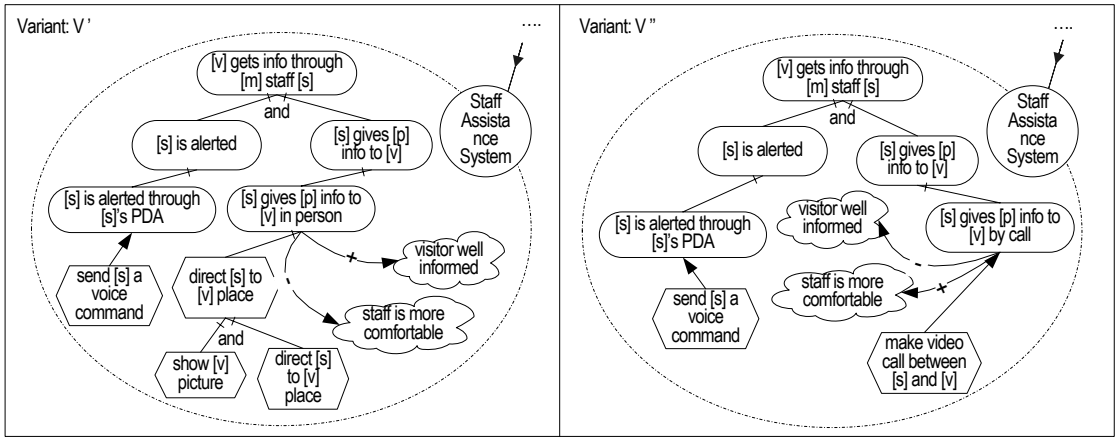


Fig. 6 Two instantiated goal model variants with different qualities

timing constraints, we may want a system developed with minimum costs sacrificing the quality and flexibility gained by supporting the whole set of goal model variants. In other words, the system has to support a set of variants that is enough to meet users' goals in all considered contexts and developed with minimum costs. To this end, we have developed a reasoning in three steps to be used at design time: (i) we exclude the variants that are unadoptable because of unsatisfiability in their contexts; (ii) we exclude the variants that can be always replaced by others; (iii) and finally, we reason about the remaining variants to extract those leading to a system developed with minimum costs and that is able to meet user goals in all analyzed contexts.

5.1 Deriving the unadoptable variants

A goal model variant is unadoptable when its context specification formula is unsatisfiable. We need to check such unsatisfiability early to save costs and fix errors given that unadoptable variants may lead to software functionalities that are never used or incorrectly specified. In this section, we develop SAT-based [23] reasoning techniques to detect unsatisfiability of contexts associated with goal model variants.

As we mentioned earlier, the context of a goal model variant is accumulative. It is the conjunction of the contexts at the variation points in that variant (see Fig. 3). We have also specified context as a world predicate formula. In order to check the satisfiability of a formula expressing a context, we need also to take into consideration all possible contradictions among its variables (world predicates). For example, in Fig. 2 we have $C_8 = wp_{8.1} \wedge wp_{8.2}$ where $wp_{8.1}$ = *"visitor is inside the museum"* and $wp_{8.2}$= *"he is not walking towards the exit"*, and $C_7 = wp_{7.1}$ where $wp_{7.1}$= *"visitor is on the way to enter the museum"*. In this example, $C_7 \rightarrow \neg C_8$ because $wp_{8.1} \rightarrow \neg wp_{7.1}$, so any goal model variant that whose context includes $C_7 \wedge C_8$ will never be applicable.

The logical relations between formulae of world predicates (contexts) can be *absolute* or *dependent* on the characteristics of the operational environment of the system:

1. *Absolute* relations apply wherever the system operates. For example, suppose we have the three world predicates wp_1= *"staff [s] of museum branch [m] has never worked in another museum branch"*, wp_2= *"visitor [v] is for the first day in [m]"* and wp_3= *" [s] assisted [v] some date before today"*, then $wp_1 \rightarrow \neg(wp_2 \wedge wp_3)$ applies in whatever museum the system could operate in.
2. *Operational environment dependent* relations hold in a particular environment where the system operates without any guarantee that such relations apply in all operational environments. For example, suppose we have the two world predicates wp_1 = *"there is enough light at the visitor location"* and wp_2 = *"visitor is inside a museum gallery room"*. If the museum keeps the light level inside the gallery rooms low, for decorating reasons or to conserve the pieces of art, then $wp_1 \rightarrow \neg wp_2$ applies always in this particular museum. Moreover, the operational environment itself assures that some contexts are always true or always false, so we have to consider a special kind of environment dependent relations of the form $Env \rightarrow world\_predicates\_formula$. For example, if the system is going to operate in a museum where elevators are available for public use, then the relation $Env \rightarrow \neg wp_3$ where wp_3 = *"visitor can not use a museum elevator"* will always hold at that museum.

We apply SAT-based techniques to check if a boolean formula is satisfiable under a set of assumptions. Given a boolean formula expressing a context and a set of logical relations between its variables[2], a SAT-solver is exploited to check if there exists a truth assignment for all variables that makes the conjunction of the context formula and the logical relations formula satisfiable. If such assignment exists, then the formula is satisfiable, otherwise it is unsatisfiable under the assumed logical relations. The pseudo-code of the algorithm (*Check-SAT*) is reported in Fig. 7.

Input: context φ
Output: $\bot$ ($\top$) if φ is unsatisfiable/satisfiable
1: $\xi := get\_logical\_relations(\xi)$
2: **if** Is_Satisfiable($\varphi \wedge \xi$) **then**
3: **return** $\top$
4: **else**
5: **return** $\bot$
6: **end if**

Fig. 7 Checking context satisfiability under assumptions (CheckSAT).

Example 2 The variant shown in Fig. 8 has an unsatisfiable context due to the contradiction between C_7 ("the visitor is on the way to enter the museum shortly before the closing time"), and C_1 ("the visitor is in the gallery building and interested in getting explanation about a piece of art"). A design decision has to be taken to accept this kind of unsatisfiability, i.e. to confirm that the model variant is indeed not needed, or to modify the model and fix it. In fact, and in this particular example, the unsatisfiability is not a modeling error but it is a side-effect of the goal model hierarchy. This hierarchy compactly represents a large number of variants in one model and it, at the same time, may include variants that are never applicable. The tasks of the unadoptable variants, such as the variant of our example, could appear in other variants with satisfiable contexts and, therefore, these tasks are not necessarily unusable if implemented in the final system. A task could be implemented in the system-to-be if it appears in, at least, one goal model variant with a satisfiable context.

5.2 Deriving the (non-)core variants

Core requirements are system requisites that can not be bargained on. There could be different perspectives to categorize requirements into core and non-core. Concerning a system supported by variants to operate in and reflect varying contexts, the variants having no alternative variants at certain contexts are core. Discovering core variants is useful for several reasons. It helps to know the parts of the system that are critical and whose failure can not be remedied by adopting other variants at certain contexts. Also, it helps to know the part of the system that needs to be developed first and can not be delayed to get a system operable in all considered contexts. The latter reason is the focus of this paper.

We develop a reasoning mechanism to derive the (non-)core goal model variants as a basic step to decide the variants to include in the system to be. The goal model variants that are preconditioned by unsatisfiable contexts will be never adopted. The developed software has to only consider the variants with satisfiable contexts. The goal model variants with unsatisfiable contexts are obviously non-core as such variants are never adoptable. Moreover, the *implications* between the contexts of goal model variants could make some variants core and others non-core. Similar to the contradictions between contexts, the implications can be absolute or dependent on the operational environment of the system. We first give some basic definitions and then develop an algorithm for processing a contextual goal model and deriving the core variants.

Definition 8 (Core variant) *a variant V_i with a context specified by a formula φ_i is core iff φ_i is satisfiable and $\nexists$ variant V_j with a context specified by a satisfiable formula φ_j: $(\varphi_i \rightarrow \varphi_j) \wedge \neg(\varphi_j \rightarrow \varphi_i)$.*

From this definition, any variant that is non-core has a set of core variants applicable in all contexts where it is itself applicable, but not vice versa. A reason for keeping such non-core variants is that at certain contexts they might assure better quality[3]. The core variants are grouped on the basis of their contexts equivalence (direct equivalence or equivalence under assumptions) to construct core groups of variants.

Definition 9 (Core groups set) *is the set of core variants partitioned on the basis of context equivalence.*

Definition 10 (Core group of variants) *is an element of the core groups set.*

In Fig. 9, we propose an algorithm that, given a contextual goal model, returns the set of all core groups of variants. The algorithm excludes the non-core variants and organizes the rest of variants in groups based on context equivalence. The algorithm excludes unadoptable variants, i.e., variants with unsatisfiable contexts,

[2] In this paper, we assume that the relations between contexts are provided by the analyst.

[3] The selection of non-core variants to support in the system-to-be is out of the scope of this paper.

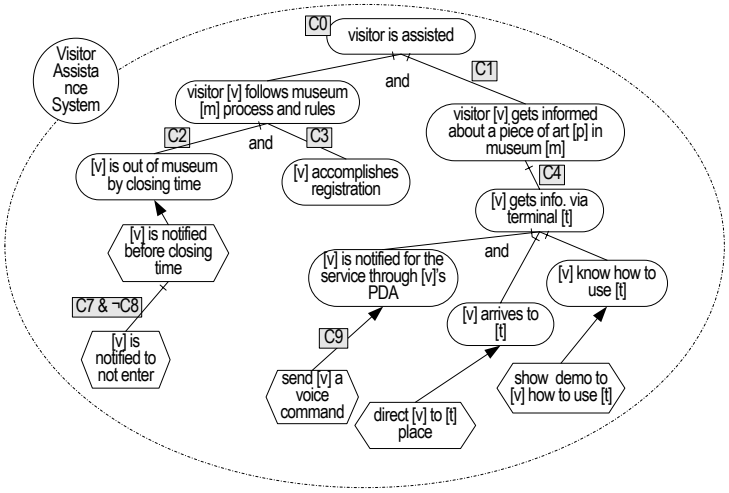


Fig. 8 A partial model variant with an unsatisfiable context

as they are obviously not core (Line 1). The algorithm then extracts the core groups of variants (Line 3–10). To this end, the algorithm partitions the set of variants based on context equivalence (Line 5). The algorithm shown in Fig.7 can be also used to check the equivalence between formulae expressing contexts. Given the logical relations (implications) (ξ) between the variables of two formulae φ_1 and φ_2 then $\varphi_1 \rightarrow \varphi_2$ iff $\neg(\varphi_1 \rightarrow \varphi_2)$ is unsatisfiable under the assumptions ξ. Then the algorithm checks if each group is core (Line 7) and adds it in the output set if it is like that (Line 8).

Input: S: *the set of all goal model variants*
Output: S'': *the set of all core gruops of variants*
1: $S' := \{V \in S : CheckSAT(V.context) = \top\}$
2: $S'' := \emptyset$
3: **while** $|S'| > 0$ **do**
4: $V := pop\_element(S')$
5: $temp := \{V\} \cup \{V' \in S' : CheckSAT(\neg(V.context \leftrightarrow V'.context)) = \bot\}$
{i.e. Check if V.context ↔ V'.context}
6: $S' := S' \setminus temp$
7: **if** $\nexists\, V' \in S' : V.context \rightarrow V'.context$ **then**
8: $S'' := S'' \cup \{temp\}$
9: **end if**
10: **end while**
11: **return** S''

Fig. 9 Extracting core groups of variants.

Example 3 In Fig 10, we show two partial contextual goal model variants $\{V_1, V_2\}$ each including a different set of tasks to implement. Both contexts of the two variants are satisfiable and $V_2.context \rightarrow V_1.context \wedge \neg(V_1.context \rightarrow V_2.context)$. This means that V_2 is non-core since there is always the variant V_1 that can replace it in all considered contexts. In the space of these two partial variants, the task *"send [s] a voice command"* and *"make voice call between [s] and [v]"* are non-core, while the tasks *"[s] is alerted via ringing tone and SMS"*, *"show [v] picture"*, and *"direct [v] to [s] place"* are core and essential to implement in order to achieve the goal *"[v] gets info through [m] staff [s]"* in all considered contexts.

5.3 Deriving the variants for minimal costs system

Developing a system that supports multiple variants to reach its requirements is desirable for several reasons such as flexibility and fault tolerance. In the previous section (Section 4), we have shown how such approach can accommodate the priorities of different users. For different reasons, such as timing and budget constraints, we may be required to develop just an operable system, i.e. a system that operates in all considered contexts. In this section, we develop the final step of the reasoning about a contextual goal model to derive a subset of its leaf tasks that leads to a system able to operate in all considered contexts and developed with minimum costs. These tasks may not implement the whole set of goal model variants, but those that are implemented will allow the system to reach its goals in all considered contexts.

Costs are not related to goals but to tasks as tasks represent executable processes while goals are just desires of an actor. Each task needs certain development resources (equipments, programmers, software packages, and so on). Each of these resources has a cost. We need to specify the resources needed for each task development and the costs of each resource to enable our target reasoning. A resource may be a part of the development of multiple tasks which means that the devel-

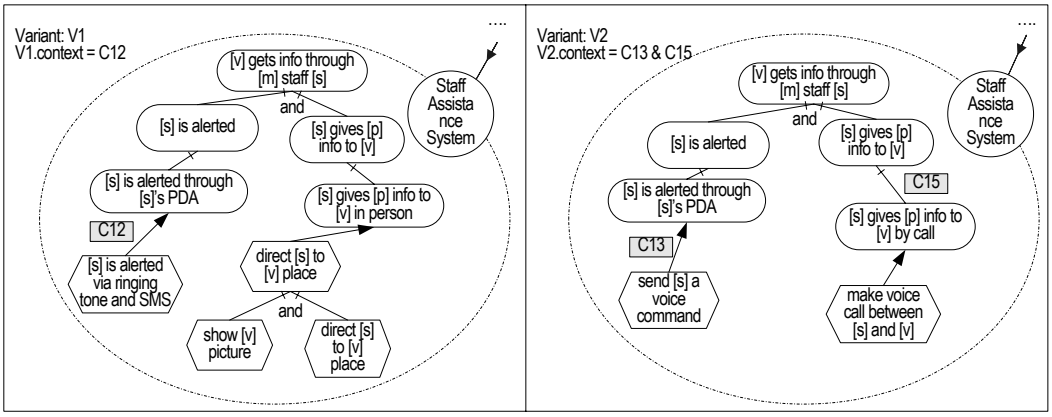


Fig. 10 V_2 is non-core because $V_2.context \rightarrow V_1.context \wedge \neg(V_1.context \rightarrow V_2.context)$

opment costs of tasks may overlap. For example, both of the tasks *"direct visitor to terminal location"* and *"direct staff to visitor location"* need almost the same resources. They both need a positioning system, communication system, and preparing a digital map of the museum. The development of the two tasks *"piece of art information is presented to visitor via video"* and *"piece of art information is presented to visitor interactively"* share the resources of gathering data about the pieces of art and preparing pictures, videos, and audio explanation to be presented, and programming the presentation.

Defining the resources needed for each task and the costs of these resources is the basic step to decide which tasks to develop. The second step is getting the core groups of variants of the contextual goal model (we have already explained this reasoning). Then we need to elicit a subset of tasks that implements, at least, one variant of each core group of variants targeting for a minimal total cost. A naive approach to do that can be based on the cartesian product of the core groups of variants and then selecting the combination of variants of minimum cost. Such approach is obviously time consuming and suffers exponential blow-up. Moreover, our experiments evidenced that it can not deal even with small-medium size goal models. Thus, we need to replace the naive approach with an optimized algorithm.

We can significantly reduce the complexity of our reasoning by exploiting the nature of the problem as shown in the algorithm reported in Fig. 11. First, the algorithm calculates the set of tasks that are mandatory for all possible combinations of variants (Lines 1–4). A task is mandatory if it is included in all the variants of, at least, one core group of variants. To reduce the number of core groups of variants to be involved in the cartesian product, the algorithm makes two checks (Lines 5–12) and produces a reduced core groups of variants set V'. A core group that includes, at least, a variant implementable using a subset of the mandatory tasks will not be added to V' (Line 7). Various core groups of variants, after excluding the mandatory tasks of them, may become equivalent and we only add one of them to V' (Line 8). The rest of the algorithm deal with the cartesian product of the sets in V' and returns the combination of variants with minimum costs (Lines 13–25).

Input: S: *the set of all core groups of variants*
Output: $MinCostTasks$: *a set of tasks that implement, at least, one variant of each core group with total minimum cost*

1: $MTasks := \emptyset$ {MTasks stands for Mandatory Tasks}
2: **for all** $CG \in S$ **do**
3: $\quad MTasks := MTasks \cup \{\bigcap\{V.tasks : V \in CG\}\}$
4: **end for**
5: $S' := \emptyset$
6: **for all** $CG \in S$ **do**
7: $\quad$ **if** $\exists V \in CG : V.tasks \setminus MTasks <> \phi$ **then**
8: $\quad\quad$ **if** $\nexists CG' \in S' : CG'.Excluding(MTasks) = CG.Excluding(MTasks)$ **then**
9: $\quad\quad\quad S' := S' \cup CG$
10: $\quad\quad$ **end if**
11: $\quad$ **end if**
12: **end for**
13: $P := S'[1].variants \times \ldots \times S'[n].variants$
14: $MinCost := +\infty$
15: $MinCostTasks := MTasks$
16: **for all** $Option \in P$ **do**
17: $\quad Tasks := Option.tasks \cup MTasks$
18: $\quad Res := \bigcup\{Task.resources : Task \in Tasks\}$
19: $\quad Cost := \sum\{R.cost : R \in Res\}$
20: $\quad$ **if** $Cost < MinCost$ **then**
21: $\quad\quad MinCost := Cost$
22: $\quad\quad MinCostTasks := Tasks$
23: $\quad$ **end if**
24: **end for**
25: **return** $MinCostOption.tasks$

Fig. 11 Extracting variants for minimum development costs.

Example 4 In Fig. 12, we show a part of the goal model shown in Fig. 2. We provide estimations for the costs of each task development aside. We show the set of variants after excluding the non-core variants as we explained in the last section. The remaining variants are grouped based on context equivalence to create core groups of variants. The relation between tasks based on the shared resources are reported. Include(T_1, T_2): the work done to gather simple information of the pieces of art is included in that needed for gathering more detailed information. Intersect(T_3, T_4, A): the interactive presentation (T_4) includes videos (the resource A) that are also needed for video-based presentation (T_3). Intersect(T_3, T_5, B), Intersect(T_4, T_5, B): all these tasks need a server and PDA for communication (the resource B). Intersect (T_4, T_8, C): we presume that T_8 is interactive which means that both of T_8 and T_4 require PDA with touch screen and the corresponding programming packages for getting user input in this way (the resource C). After this specification, we show the set of tasks to develop and the variant that are implemented on them and the final minimized costs.

6 Automated Support Tool: RE-Context

In order to support the analyst in the reasoning techniques we described in the previous two sections, we have developed a prototype automated reasoning tool called *RE-Context*. It takes as input a contextual goal model expressed as an input file for DLV[4], a disjunctive Datalog [24] implementation. At the moment, we do not provide a graphical goal modeling editor neither automated translation to the DLV input format.

Code 2 shows how part of the goal model of Fig. 12 is translated to the input format for RE-Context. Goal and context labels begin with a lowercase letter because leading uppercase letters represent variables in DLV. The top-level goal G_1 is Or-decomposed to sub-goals G_2 and G_3 (line 1). Line 1 shows the syntax that allows for DLV to select either G_2 or G_3 if G_1 is chosen. If G_2 is selected, then C_5 should apply (line 2); if G_3 is selected, then C_6 should be valid. Goal G_2 is and-decomposed to G_4 and G_5 (lines 4-5). There are two tasks that are means-end linked to G_4: T_1 and T_2; DLV should choose among them, as expressed in line 6. Similarly (line 7), in order to achieve G_5 DLV should select either T_3 or T_4. Line 8 is the input for DLV to start planning: it states that G_1 should be achieved.

The preliminary step of RE-Context is to derive all variants, and this consists of running the DLV reasoner using it as a planner on the goal model: the output consists of all the valid models that satisfy the rules in the input file. Each variant consists of a set of tasks to execute and the set of contexts required for each variant.

Code 2 Part of the goal model of Fig. 12 expressed in DLV as input for RE-Context.

```
1 ach(g2) v ach(g3) :- ach(g1).
2 c5 :- ach(g2).
3 c6 :- ach(g3).
4 ach(g4) :- ach(g2).
5 ach(g5) :- ach(g2).
6 todo(t1) v todo(t2) :- ach(g4).
7 todo(t3) v todo(t4) :- ach(g5).
8 ach(g5).
```

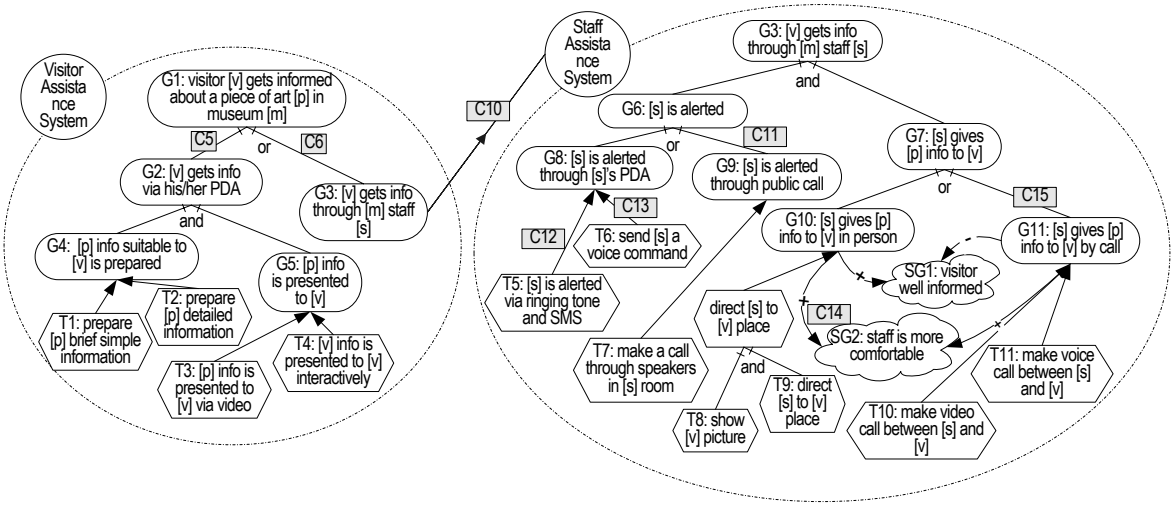
The next mandatory step is to check context satisfiability for each variant. This corresponds to run the *CheckSAT* algorithm described in Fig. 7. To verify the satisfiability of a context, RE-Context uses the state-of-the-art SMT solver MathSAT[5]. RE-Context loads the definition of the contexts from separate files: for instance, context C_1 is represented in the code as `c1)` and is defined in the file `c1.txt` as a boolean formula expressed over a set of variables. The variables of this formula are the leaf facts of C_1 context hierarchy. Variants with unsatisfiable contexts are not considered in the later reasoning steps, since they can not be adopted in any context.

After these two steps are completed, RE-Context can be run in two usage modes, each corresponding to one of the reasoning techniques we described in this paper: (i) deriving the variants for a given context and user prioritization and (ii) deriving the variants leading to minimum development costs.

Deriving variants for varying contexts. The input for this activity includes the contexts that apply and the user prioritization of softgoals. The latter input is provided by representing (contextual) contributions to soft-goals and the importance given by the user to the various soft-goals (0 = "I don't care", 5 = "I care very much"). Code 3 shows how to express user prioritization or softgoals for the example in Fig. 12. Line 1 says that context C_{14} is true; lines 2-3 express the interest of the user in both soft-goals SG_1 (visitor well informed) and SG_2 (staff is more comfortable). Lines 4-7 show the contributions from goals G_{10} and G_{11} to the soft-goals; in particular, line 5 shows a contextual contribution from G_{10} to SG_2: the contribution is positive only if context C_{14} is true. RE-Context returns the best variant, namely the one that better contributes to the soft-goals the user cares about.

[4] http://www.dbai.tuwien.ac.at/research/project/dlv/

[5] http://mathsat4.disi.unitn.it



The non-core variant	The variants excluding the non-core variants	The core groups of variants	The cost relations	The min-cost core requirements
NV1= {T6, T10} **NV2**= {T6, T11} Both can be replaced by **V2** due to the implications: C13→C12 and the trivial C15 → true.	**V1**= {T1, T3} **V2**= {T1, T4} **V3**= {T2, T3} **V4**= {T2, T4} **V5**= {T5, T8, T9} **V6**= {T7, T8, T9}	**Core1**= {V1, V2, V3, V4} **Core2**= {V5} **Core3**= {V6}	**Cost**(T1,30) , **Cost**(T2,40), **Cost**(T3,60) , **Cost** (T4,80), **Cost**(T5,25) , **Cost**(T6,35), **Cost**(T7,50) , **Cost** (T8,30), **Cost** (T9,50) , **Cost** (T10,50), **Cost** (T11, 30). **Include**(T2, T1), **Intersect**(T3, T4, 40), **Intersect** (T3, T5, 20), **Intersect**(T4, T5, 20), **Intersect** (T4, T9, 30) **Cost of developing all tasks= 340**	**The tasks to develop=** {T1, T4, T5, T7, T8, T9} **Costs=** 215 **The variants implemented:** { V2, V5, V6}

Fig. 12 Illustration of the minimum-cost core requirements extraction.

Code 3 User preferences in Fig. 12 expressed in the RE-Context input format.

```
1 phi(c14).
2 softgoal(sg1,3).
3 softgoal(sg2,1).
4 contrib(g10,sg1,pos).
5 contrib(g10,sg2,pos) :- phi(c14).
6 contrib(g11,sg1,neg).
7 contrib(g11,sg2,pos).
```

Deriving variants for minimum development costs. The first step to reason about minimum development cost is to get rid of non-core variants; this task is carried out by running the SAT-solver based tool to check whether there are replaceable variants (see Def. 8). Subsequently, RE-Context groups the variants in core groups, where each core group contains variants whose contexts are equivalent; RE-Context runs the SAT-solver based tool to identify equivalent contexts. Once core groups are identified, the minimum development cost should be computed, by choosing one variant from each core set that lead to a total minimum costs. Costs are expressed for each task on the basis of the resources they need. Lines 1-5 in Code 4 shows the development cost for tasks T_1 and T_2 (taken from Fig. 12). Task and resources labels are represented with a leading lowercase letter due to DLV syntactic rules. Lines 1-2 define the cost for resources R_1 and R_2, respectively. Lines 3-5 define the relation between tasks and development resources: task T_1 requires only resource R_1, whereas T_2 requires both R_1 and R_2. Therefore, there is an intersection between T_1 and T_2 for R_1; more precisely, there is an inclusion relation (T_2 includes T_1), given that all resources needed by T_1 are also needed by T_2.

Code 4 Development cost for tasks $t1$ and $t2$ in Fig. 12.

```
1 cost(r1,30) :- needed(r1).
2 cost(r2,10) :- needed(r2).
3 needed(r1) :- todo(t1).
4 needed(r1) :- todo(t2).
5 needed(r2) :- todo(t2).
```

6.1 Evaluation

We have organized a seminar to present our framework, invited four requirements engineers with good expertise in goal modeling, and explained our framework to them. We have then invited an expert in mobile applications from the Laboratory of Mobile Application (LaMA) to

describe the museum-guide scenario. Then we asked the requirements engineers to use our framework to model the museum guide requirements. Together with the domain expert, we have answered the questions the engineers have raised during the session. We have then formalized the contextual goal model the engineers have drawn, then we ran our tool and obtained the results summarized in Fig. 13. RE-Context has been installed on a computer equipped with a AMD Athlon(tm) 64 X2 Dual Core Processor 5000+, 4 GB RAM, Sun Java JRE 1.6.0_07-b06, Linux Debian 2.6.18.dfsg.1-12.

Fig. 13 shows the results obtained by running the automated reasoning techniques on the case study. The first two columns show the time required to develop (TD) and formalize (TF) the goal model; clearly, the time taken by these activities can be reduced with the aid of a CASE tool. Then, the table presents data concerning the goal model size in terms of the number of actors (NA), goals (NG), tasks (NT), soft-goals (NSG), variation points (NVP), and variants (V).

Then, the figure contains data collected by running the tool. First, we report the number of iterations where the tool asked us to fix or accept unsatisfiability. Whenever an unsatisfiable context is accepted, i.e. the variant is indeed unadoptable, the tool also excludes the other variants that contain it. After unsatisfiability checking, the tool processes the variants with satisfiable contexts: the number of non-core variants (NCV), the total cost to develop all tasks aside (TC), the shared cost among all tasks (SC), the cost of developing all tasks (CAS=TC-SC), the number of core groups of variants (CGV) and the minimum cost for a system working in all considered contexts (MC).

After running RE-Context on the original goal model, we tested its scalability on goal models of different sizes, as shown in Fig. 14. The original goal model is that of the case study. To get models of smaller and larger sizes, we have taken sub-trees and cloned them, in similarly to the approach in [25]. The first two columns in Fig. 14 show the number of nodes (NN) and variants (NV) in the goal model, whereas the next four columns show the time of executing our reasoning. *T_Der* is the time to derive all variants of goal model, *T_Inc* is the time required to get variants with unsatisfiable contexts, while *T_CGV* is the time the tool required to get the core groups of variants.

The graph on the right-hand side depicts the results shown in the table: the x-axis represents logarithmically the number of variants, the y-axis represents logarithmically the computation time needed. The collected data show that the time needed for computation is growing exponentially with the increase of the problem size. Anyhow, given that the reasoning is performed at design-time, the tool scales quite well in the test cases (it takes less than 16 minutes with 648.000 variants). We don't present here results for for deriving variants under given context and user prioritization over soft-goals, as the computational cost is negligible. Moreover, the algorithm we have shown in Fig. 11 led to negligible time for computing the tasks to develop with minimum costs. We remark that the exponential growth is due to the nature of the goal model that allows for a huge number of variants to be modeled compactly. Future work will involve the processing of goal model iteratively during the construction to reduce the final complexity. Moreover, applying divide-and-conquer techniques could potentially reduce the complexity of reasoning about very large contextual goal models.

7 Related Work

The research in context modeling, (e.g., [26]), concerns finding modeling constructs to represent software and user context, but there is still a gap between the context model and software behavior model, i.e. between context and its use. We tried to reduce such a gap at the goal level and allow for answering questions like: "*how do we decide the relevant context?*", "*why do we need context?*" and "*how does context influence software and user behavior adaptation?*". Salifu et al. [27] investigate the use of problem descriptions to represent and analyze variability in context-aware software. Their work recognizes the link between requirements and context as a basic step in designing context-aware systems.

Software variability modeling, mainly feature models [28,29], concerns modeling a variety of possible configurations of the software functionalities to allow for a systematic way of tailoring a product upon stakeholder choices, but there is still a gap between each functionality and the context where this functionality can or has to be adopted, the problem we tried to solve at the goal level. Furthermore, our work is in line, and has the potential to be integrated, with the work in [30] and the FARE method proposed in [31] that show possible ways to integrate features with domain goals and knowledge to help for eliciting and justifying features.

Requirements monitoring is about insertion of a code into a running system to gather information, mainly about the computational performance, and reason if the running system is always meeting its design objectives, and reconcile the system behavior to them if a deviation occurs [8]. The objective is to have more robust, maintainable, and self-evolving systems. In [32], a GORE (goal-oriented requirements engineer) framework KAOS [6] was integrated with an event-monitoring system (FLEA [33]) to provide an architecture that

Time		Goal Model Size						Iterations	NCV	TC	SC	CAS	CGV	MC
TD	TF	NA	NG	NT	NSG	NVP	V							
16 Hours	7.5 Hours	3	41	51	7	26	324000	40	192	2845	2045	800	84	525

Legend

TD : time to develop the graphical model
TF : time to formalize the model & fix inconsistencies
NA : number of actors.
NG : number of goals.
NT : number of tasks.
NSG : number of softgoals.
NVP : number of variation points.
V : number of variants.

Iterations : number of iterations to fix/accept all unsatisfiabilities.
NCV : number of non-core variants.
TC : the total cost of developing the tasks each aside.
SC : the shared cost between all tasks.
CAS : the cost of developing all tasks, i.e, all variants
CGV : the number of core groups of variants.
MC : the minimum cost set of tasks that implements, at least, one variant of each core group of variants.

Fig. 13 The results obtained by applying the developed tool on the museum-guide system

Size of goal model		T_Der	T_Inc	T_CGV
NN	NV			
10	15	108	18	153
20	60	238	43	1968
33	90	337	64	4200
47	2250	1081	272	7182
58	20250	4781	716	6912
91	324000	147388	91098	10662
100	648000	551543	381142	25151

Legend

NN : number of nodes
NV : number of variants in the model.
T_Der : time to derive all variants
T_Inc : time to get variants with inconsistent contexts.
T_CGV : time to get the core groups of variants

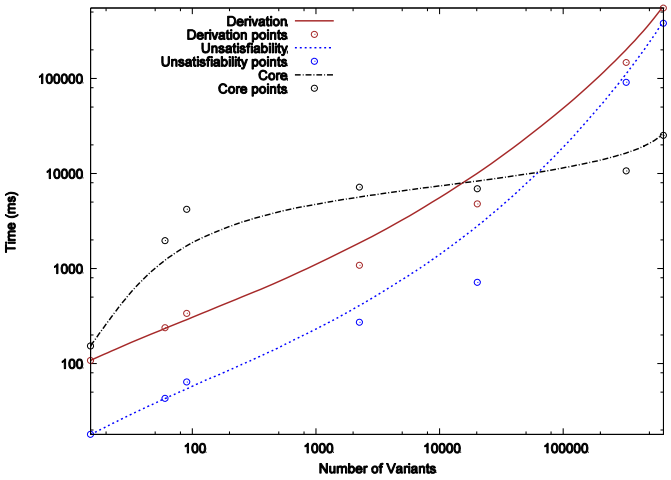


Fig. 14 Tabular and Graphical representation of the performance of the developed tool. Time is in milliseconds.

enables the runtime automated reconciliation between system goals and system behavior with respect to a priori anticipated or evolving changes of the system environment. Differently, we propose model-driven framework that concerns an earlier stage, i.e. requirements, with the focus on identifying requirements together with context, and eliciting the monitoring data.

Customizing goal models to fit to user skills and preferences was studied in [34,35]. The selection between goal model variants is based on one dimension of context, i.e. user skills, related to the atomic goals (executable tasks) of the goal hierarchy, and on user preferences which are expressed over softgoals. In [36] Lapouchnian et al. propose techniques to design autonomic software based on an extended goal modeling framework, but the relation with the context is not focused on. Liaskos et al [37], study the variability modeling under the requirements engineering perspective and propose a classification of the intentional variability when Or-decomposing a goal. We focused on context variability, i.e. the unintentional variability, which influences the applicability and appropriateness of each goal model variant. Reasoning with Tropos goal model has been already studied in [38]; adding context to goal models creates the need to integrate between reasoning with context and that with the goal model.

8 Conclusions and Future Work

In this paper, we have developed a goal-oriented framework for modeling and analyzing requirements for varying contexts. We extended Tropos goal model to capture the relationship between each variant to goal satisfaction and context. In turn, context is defined through a hierarchial analysis. The context analysis represent a systematic way to identify facts the system needs to verify in order to confirm an analyzed context.

We have formalized the extended goal model and developed two reasoning techniques. The first technique

is for deriving the requirements variants with respect to context and user priorities. This reasoning technique is used at runtime to determine the variant to adopt from the pool of goal model variants supported by the system. The second technique is for deriving a minimum-cost set of tasks that has to be implemented to enable the system of meeting users' goals in all considered contexts. We have applied our framework on a a scenario of a mobile information system to assist visitors in museums and we have reported and discussed the results obtained.

Concerning future work directions, we aim to improve our automated support towards an automatic generation of the contextual goal models formalization, an optimization of the performance to deal efficiently with very large contextual goal models, besides reducing the manual input that the analyst has to provide (especially when specifying the relations between contexts). Besides improving the automated support, we will focus our research along two lines:

- **Managing viewpoints of context:** besides the potential inconsistency between different stakeholders' specifications of requirements, that is well studied in the literature (e.g., [39]), context specification itself might be debatable. We need to manage multiple perspectives (viewpoints) of context since different stakeholder might specify context differently or even in contradictory ways. Categorizing, detecting, and managing, such differences in context specifications are necessary to have well specified requirements. For example, in a museum-guide system, the context A="visitor is interested in watching a documentary film" is a high level context that can be differently specified by different stakeholders. One stakeholder can say ($A \leftarrow B \vee C$) where B="the film is related to the visitor's local culture" and B="the film concerns a city where the visitor has been once at least". Another stakeholder might say: A="a visitor is interested in the film if it conveys very new information to him". To some extent, these two descriptions are inconsistent.
- **Context and security requirements:** most of security requirements approaches (such as Secure Tropos [40]) deal with security requirements that are context-independent. In some cases, context can influence security requirements and we would need to do research in context-dependent security requirements. For example, in an emergency situation (such as fire), a visitor will accept rescue team to know his location and other data needed to guide him to a safe area, while in a normal situation a visitor would have more restricted security concerns.

Acknowledgements This work has been partially funded by EU Commission, through the SERENITY, and COMPAS projects, and by the PRIN program of MIUR under the MEnSA project. We also thank Jaelson Brelaz de Castro, Bashar Nuseibeh, Yijun Yu, Alberto Griggio, Anders Franzen, John Mylopoulos, and Amit Chopra for the helpful discussions that enriched the ideas in this paper.

References

1. Mark Weiser. The computer for the twenty-first century. *Scientific American*, 265(3):94–104, 1991.
2. A. Finkelstein and A. Savigni. A framework for requirements engineering for context-aware services. In *Proceedings of STRAW 01*, 2001.
3. E. Yu and J. Mylopoulos. Why goal-oriented requirements engineering. In *Proceedings of REFSQ'98*, pages 15–22, 1998.
4. E.S.K. Yu. Modelling strategic relationships for process reengineering. *Ph.D. Thesis, University of Toronto*, 1995.
5. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
6. Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.
7. John Mylopoulos, Lawrence Chung, and Eric Yu. From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1):31–37, 1999.
8. S. Fickas and M.S. Feather. Requirements monitoring in dynamic environments. In *Proceedings of RE 1995*, page 140. IEEE Computer Society Washington, DC, USA, 1995.
9. D. Sykes, W. Heaven, J. Magee, and J. Kramer. From goals to components: a combined approach to self-management. In *Proceedings of SEAMS'08*, pages 1–8. ACM New York, NY, USA, 2008.
10. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Location-based variability for mobile information systems. In Zohra Bellahsene and Michel Léonard, editors, *Proceedings of CAiSE'08*, volume 5074 of *LNCS*, pages 575–578. Springer, 2008.
11. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Location-based software modeling and analysis: Tropos-based approach. In Qing Li, Stefano Spaccapietra, Eric S. K. Yu, and Antoni Olivé, editors, *Proceedings of ER 2008*, volume 5231 of *LNCS*, pages 169–182. Springer, 2008.
12. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal modeling framework for self-contextualizable software. In *Proceedings of EMMSAD 2009*, volume 29 of *LNBIP*, pages 326–338. Springer, 2009.
13. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Goal-based self-contextualization. In *CAiSE'09 - Forum*, volume 453, pages 37–42. CEUR-WS, 2009.
14. P. Brezillon. Context in artificial intelligence: I. a survey of the literature. *Computers and artificial intelligence*, 18:321–340, 1999.
15. A.K. Dey. Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7, 2001.
16. J. Krogstie, K. Lyytinen, A.L. Opdahl, B. Pernici, K. Siau, and K. Smolander. Research areas and challenges for mobile information systems. *International Journal of Mobile Communications*, 2(3):220–234, 2004.
17. X. Shen, B. Tan, and C.X. Zhai. Context-sensitive information retrieval using implicit feedback. In *Proceedings of SIGIR 2005*, pages 43–50. ACM, 2005.

18. M. Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.
19. Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE 2001*, pages 249–262. IEEE Computer Society, 2001.
20. I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. In *RE 2008*, pages 71–80. IEEE Computer Society, 2008.
21. Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J.C.S.P. Leite. From goals to high-variability software design. In *Proceedings of ISMIS'08*, volume 4994 of *LNCS*, pages 1–16. Springer, 2008.
22. F. Dalpiaz, P. Giorgini, and J. Mylopoulos. An architecture for requirements-driven self-reconfiguration. In *Proceedings of CAiSE'09*, volume 5565 of *LNCS*, pages 246–260. Springer, 2009.
23. A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009.
24. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
25. Y. Wang, S.A. McIlraith, Y. Yu, and J. Mylopoulos. An automated approach to monitoring and diagnosing requirements. In *Proceedings of ASE 2007*, pages 293–302. ACM New York, NY, USA, 2007.
26. Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *Proceedings of PerCom'04*, pages 77–86. IEEE Computer Society, 2004.
27. M. Salifu, Y. Yu, and B. Nuseibeh. Specifying monitoring and switching problems in context. In *Proceedings of RE 2007*, pages 211–220. IEEE Computer Society, 2007.
28. K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.
29. Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.
30. Y. Yu, J.C.S. do Prado Leite, A. Lapouchnian, and J. Mylopoulos. Configuring features with stakeholder goals. In *Proceedings of SAC 2008*, pages 645–649. ACM New York, NY, USA, 2008.
31. M. Ramachandran and P. Allen. Commonality and variability analysis in industrial practice for product line improvement. *Software Process: Improvement and Practice*, 10(1), 2005.
32. MS Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *IWSSD'98*. Association for Computing Machinery, Inc, One Astor Plaza, 1515 Broadway, New York, NY, 10036-5701, USA,, 1998.
33. D. Cohen, M.S. Feather, K. Narayanaswamy, and S.S. Fickas. Automatic monitoring of software requirements. In *Proceedings of ICSE 1997*, pages 602–603. ACM New York, NY, USA, 1997.
34. B. Hui, S. Liaskos, and J. Mylopoulos. Requirements analysis for customizable software: A goals-skills-preferences framework. In *Proceedings of RE 2003*, pages 117–126. IEEE Computer Society, 2003.
35. S. Liaskos, S. McIlraith, and J. Mylopoulos. Representing and reasoning with preference requirements using goals. Technical report, Dept. of Computer Science, University of Toronto, 2006. ftp://ftp.cs.toronto.edu/pub/reports/csrg/542.
36. Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, and John Mylopoulos. Requirements-driven design of autonomic application software. In *Proceedings of CASCON '06*. ACM, 2006.
37. Sotirios Liaskos, Alexei Lapouchnian, Yijun Yu, Eric Yu, and John Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of RE 2006*, pages 76–85. IEEE Computer Society, 2006.
38. Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Reasoning with goal models. In *Proceedings of ER 2002*, volume 2503 of *LNCS*, pages 167–181. Springer, 2002.
39. B. Nuseibeh, J. Kramer, and A. Finkelstein. Expressing the relationships between multiple views in requirements specification. In *Proceedings of ICSE 1993*, pages 187–196. IEEE Computer Society, 1993.
40. H. Mouratidis and P. Giorgini. Secure tropos: A security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2):285–309, 2007.