UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it

Reasoning about Contextual Requirements for Mobile Information Systems: a Goal-based Approach

Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini

March 2010

Technical Report # DISI-10-029

Reasoning about Contextual Requirements for Mobile Information Systems: a Goal-based Approach

Raian Ali, Fabiano Dalpiaz, Paolo Giorgini

University of Trento - DISI, 38100, Povo, Trento, Italy.

Abstract

Mobile information systems are characterized by their ability to adapt to varying contexts. There is a strong relationship between the requirements of a mobile information system and its operational context. On the one hand, context can be considered to determine the requirements, the alternatives that can be adopted to satisfy these requirements, and qualities of the systems. On the other hand, the system itself may cause changes in the context during its operational activities. More fundamentally, context influences users' goals and the way they achieve these goals. In previous work, we introduced contextual goal modeling, a systematic way to analyze and specify contexts. The key idea was to associate variants for goal satisfaction with contexts. In this paper, we propose analysis techniques and automated reasoning mechanisms to verify properties of contextual goal models. We show how to detect inconsistent context specifications and how to analyze possible runtime conflicts originated by context changes. Also, we present an analysis process to construct and reason about contextual goal models. We illustrate and evaluate our framework through a case study of a mobile information system for supporting the life of people with dementia.

Keywords: Mobile Information Systems, Requirements Engineering, Contextual Requirements, Goal Modeling.

Email address: raian.ali, fabiano.dalpiaz, paolo.giorgini@disi.untin.it (Raian Ali, Fabiano Dalpiaz, Paolo Giorgini)

1. Introduction

The recent advances in computing and communication technologies such as sensor systems, positioning systems, mobile devices, and so on, have led to the emergence of mobile information systems (MobISs). Such systems weave computing with humans living environments in order to facilitate the achievement of users' needs. A core element of MobISs is the operational context of the system, that is monitored and used to adapt the system behavior. Contextual information, such as spatio-temporal, environmental, social, task, and personal information are main factors for deciding what a MobIS has to perform, how, and how well it can perform it [1].

Context has a strong influence on the requirements of a MobIS. It should be considered for deciding about requirements to meet, choosing among possible ways to satisfy these requirements, and assessing the qualities of these ways. On the other hand, the system itself may cause changes in the context as a consequence of its actions on the operational environment. In spite of such a mutual influence, context is either ignored or presumed uniform in most requirements engineering literature and is considered mainly during the later stages of software development (Architecture [2], Runtime Adaptation [3], HCI [4], Services [5]). Traditional requirements engineering approaches fall short of specific approaches for MobISs. Thus, modeling and analyzing requirements for information systems reflecting their context is currently a main research challenge [6, 1].

Information systems are means to reach users' and organizations' goals [7, 8, 9]. In the case of MobISs, goals are strongly influenced by the context. For example, in a health care institute for people with dementia, a caregiver may have the goal to *"involve the patient in social activities"* (G_1) whenever *"the patient is feeling bored and it has been long time since his last social activity"* (C_1). The caregiver can satisfy goal G_1 both by *"taking the patient for a trip in the city"* ($G_{1.1}$) or *"asking a relative or an old friend of the patient to come"* ($G_{1.2}$). Goal $G_{1.1}$ is adoptable only if *"the city is not crowded"* ($C_{1.1}$), since people with dementia usually get anxious in crowded places. Goal $G_{1.2}$ is adoptable only if *"the patient has relatives or friends that can come"* ($C_{1.1}$). The requirements model of a MobIS that supports people with dementia should reflect the caregiver goals G_1, $G_{1.1}$, and $G_{1.2}$, the rationale $G_{1.1} \vee G_{1.2} \rightarrow G_1$ and adaptation to contexts: (i) if $C_1 \wedge C_{1.1}$ then $G_{1.1}$, and (ii) if $C_1 \wedge C_{1.2}$ then $G_{1.2}$.

Goal models have been proposed in the requirements engineering literature (i^* [10], Tropos [11, 12], and KAOS [13]) to represent high level goals and possible variants (alternatives) for their satisfaction. Moreover, goal models have been used to represent the rationale of both humans and software systems [14], and they have been shown very useful for adaptive systems engineering in particular [15, 16]. In [17, 18, 19, 20], we have proposed contextual goal models to capture the relation between contexts and the space of goal model variants. A contextual goal model may incorporate a large number of context specifications and variants for goal satisfaction that may easily lead to modeling errors which, amongst other things, may make the model inconsistent. In this paper, we propose analysis techniques and automated reasoning mechanisms to verify properties of contextual goal models. We show how to detect inconsistent context specifications and how to analyze possible runtime conflicts originated by context changes. We present an analysis process for capturing and reasoning about contextual requirements. Finally, we evaluate our framework through a case study of a MobIS for supporting people with dementia. Our approach contributes to facilitate the analysis of MobISs and completes our earlier work providing systematic process and reasoning mechanisms to model and analyze MobISs requirements.

The paper is structured as follows. In Section 2, we describe contextual goal models. In Section 3, we propose reasoning techniques to detect inconsistent contexts. In Section 4, we introduce reasoning techniques to detect and provide information about conflicts. In Section 5, we describe an automated support tool implementing our reasoning techniques. In Section 6, we discuss our requirements analysis process. We evaluate our framework in Section 7, discuss related work in Section 8, and conclude in Section 9.

2. Background

In this section, we briefly discuss our previous work on contextual goal modeling. We first introduce a case study using the Tropos methodology, and then summarize a set of definitions and models proposed in [17, 18, 19, 20].

2.1. Case Study

We take a case study of a MobIS for supporting the life of people with dementia. The case study is a variant of the scenario described in [21] and used in the EU sponsored Serenity project[1]. The MobIS has to support some

[1] http://www.serenity-project.org/

daily tasks that the patient might forget to do, such as eating, circulating the air inside the patient home, taking medicines, and so on. It has also to facilitate rescue activities; in case of health emergencies the Medical Emergency Rescue Center (MERC) is notified and requested to send a rescue team to the house. Besides their memory impediments, patients with dementia suffer from anxiety attacks. The MobIS should manage such situations by making the patient aware of the anxiety attack, or by preventing him from getting out of the house in an unusual way. The MobIS then has to calm the patient down, and call the caregiver to come and give a treatment. The MobIS supports also some other general tasks, such as preventing a potential house robberies (e.g., it can give the illusion that the home is lived in when the patient is out for long time).

In Figure 1, we show a partial Tropos goal model for the MobIS of our case study. Tropos goal analysis projects the system as a set of interdependent actors, each having its own strategic interests (*goals*). Goals are analyzed iteratively and in a top-down way, to identify the more specific sub-goals needed for satisfying the upper-level goals. Goals can be ultimately satisfied by means of executable processes (*tasks*). The actor *"Patient Caregiving System"* has the top-level goal *"home is managed for safety of patient"*), which is iteratively decomposed into subgoals by AND-decomposition (all subgoals must be achieved to fulfil the top goal) and OR-decomposition (at least one subgoal must be achieved to fulfil the top goal). The sub-goal *"home is protected against robbery"* is AND-decomposed into the subgoals *"give illusion of being lived in"* and *"act against potential robbery"*; the subgoal *"enforce routine exit procedure"* is OR-decomposed into the subgoals *"patient is alerted"* and *"patient is prevented of exiting"*. Goals are finally satisfied by means of executable tasks; the goal *"fresh air inside home"* can be reached by one of the tasks *"open windows"* and *"turn air ventilator on"*.

A dependency indicates that an actor (*depender*) depends on another actor (*dependee*) to attain a goal or to execute a task: the actor *"Patient Caregiving System"* depends on the actor *"Neighbor Assistance System"* for the goal *"a neighbor comes"*. This last goal is an alternative to the goal *"police comes''* and they both are alternatives for achieving a higher level goal that is *"assistance comes to act against robbery"*. Softgoals (*"patient privacy"*) are qualitative objectives for whose satisfaction there is no clear cut criteria, and they can be contributed either positively or negatively by goals and tasks: *"open windows"* usually contributes negatively to *"patient privacy"*, while *"turn on air ventilator"* contributes positively to it.

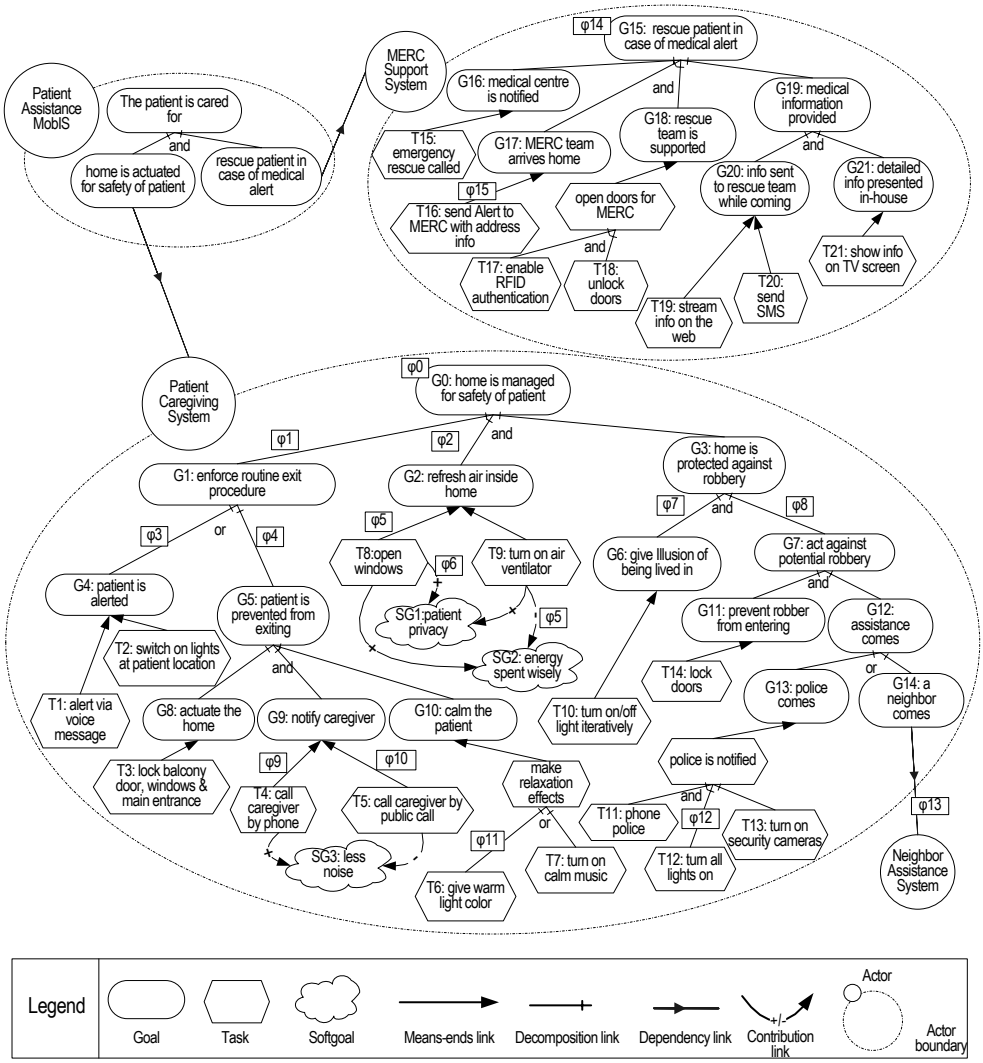

Figure 1: A partial Tropos goal model for the MobIS case study

2.2. Context and Contextual Goal Models

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, a caregiving system is an actor that may have the goal of acting against a potential robbery and keep the home protected. The caregiving system has the ability to decide when to activate this goal and what to do to reach it. The caregiving system may activate such a goal when there is a person who is trying to enter the home area in an unusual way. The caregiving system may reach such goal by calling the police or a neighbor and the decision between these two options is left to the caregiving system itself. The decision taken by an actor may depend on the state of a portion of the world such actor lives in. We call such a state *context*:

Definition 1 (Context). *A context is a partial state of the world that is relevant to an actor's goals.*

Contexts can be associated at the following variation points of a goal model (annotated by φ_i in the goal model of Figure 1 and described in Table 1):

1. *OR-decomposition*: the adoptability of a sub-goal (sub-task) in an OR-decomposition may require a specific context.
2. *Means-end*: the adoptability of a task in a means-end may require a specific context.
3. *Actor dependency*: a certain context may be required for an actor to attain a goal/get a task executed by delegating it to another actor.
4. *Root goals*: root goals may be activated only in certain contexts.
5. *AND-decomposition*: the satisfaction (execution) of a sub-goal (sub-task) in an AND-decomposition might be needed only in certain contexts; i.e., some sub-goals (sub-tasks) are not always mandatory to fulfil the top-level goal (task).
6. *Contribution to softgoal*: softgoals can be contributed either positively or negatively by goals and tasks. The contributions to softgoals can also vary from one context to another.

Similar to goals, context may need to be analyzed. On the one hand, goal analysis provides a systematic way to discover alternative set of tasks

	Description	Technology
$\varphi 0$	Home is lived in, and the patient is expected to have some dementia problem, and there is no awaken caregiver or healthy relative at home.	Database (info about home and patient), RFID tags (caregiver and relative)
$\varphi 1$	Patient is anxious and he is at home.	Smart-shirt or oxymeter, camera with motion recognition
$\varphi 2$	Humidity level in the house is too high, or home windows and doors haven't been opened for long time.	Humidity sensor, magnetic sensor (open-close), database
$\varphi 3$	The patient dementia disease is not in an advanced stage or he is moderately anxious.	Database (disease status), smart-shirt (anxiety)
$\varphi 4$	The patient suffers of advanced dementia, and he seems to be extremely anxious	Database, smart-shirt
$\varphi 5$	It is sunny and not very windy.	Barometer and wind sensor
$\varphi 6$	The patient is outside home.	GPS or RFID
$\varphi 7$	The patient is outside home since long time and it is night time.	GPS/RFID, database, digital clock
$\varphi 8$	A person is trying to get into the yard in a suspicious way (e.g., enter from a place different from the main gate).	Surveillance camera
$\varphi 9$	The phone is free and the caregiver is not using his phone for a call.	Information from telephony company, phone busy sensor
$\varphi 10$	It is not night time.	Digital clock
$\varphi 11$	The light level at patient location is too low or too high.	Light sensor
$\varphi 12$	It is too dark inside home.	Light sensor
$\varphi 13$	The neighbor is healthy, is at home, and can see or reach easily the patient's home.	Database (health status and house location), GPS/RFID (neighbor position)
$\varphi 14$	The patient health turns bad or he has fallen down.	Smart-shirt, oxymeter, camera with motion recognition
$\varphi 15$	The MERC is reachable and online.	Check connection
$\varphi 16$	The house has a device that can show medical information.	Database

Table 1: The description of Figure 1 contexts and the technology needed to monitor them

an actor may execute to reach a goal. On the other hand, context analysis should provide a systematic way to discover alternative sets of facts an actor may verify to judge if a context applies. We specify context as a formula of world predicates. The EBNF of this formula is as shown in Code 1.

Code 1 The EBNF of context world predicates formula

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

Definition 3 (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, namely it has the ability to capture the necessary data and compute the truth value of a fact. A statement can not be verified by an actor for different reasons, such as (i) lack of information to verify it; (ii) the abstract nature of the statement makes it hard to find an evaluation criteria. 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. However, a statement can be refined into a formula of facts and other statements. We call the relation between such a formula of word predicates and a refined statement *Support*, and we define it as following:

Definition 4 (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 might be refined to a formula of facts that supports it. In our contextual goal model, we allow only for **monitorable contexts**. A context is monitorable if it can be specified in terms of facts and/or statements that are supported by facts. 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. In Figure 2, we analyze the context φ_1. In this figure, *statements* are represented as shadowed rectangles and *facts* as parallelograms. The relation *support* is represented as curved filled-in arrow The *and*, *or*, *implication* logical operators are represented as black triangles, white triangles, filled-in arrows, respectively.

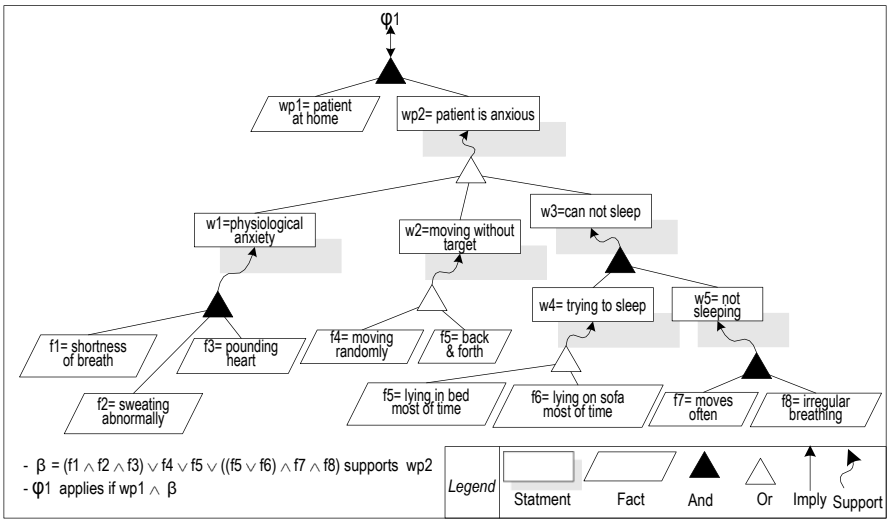

Figure 2: A context analysis for φ_1

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 of the world an actor can collect . For example, taking the facts of the context analysis shown of Figure 2, we could develop a data conceptual model, shown in Figure 3. Such model should be implemented and maintained by the MobIS in order to verify facts, judge if the analyzed contexts apply, and take decisions at the corresponding variation point of the goal model.

Based on its influence on goal model variants, we classify context in three kinds, each one is represented at a set of variation points of contextual goal models:

1. **Activation context**, when the context makes it necessary to achieve (execute) a set of goals (tasks). In our contextual goal model, activation

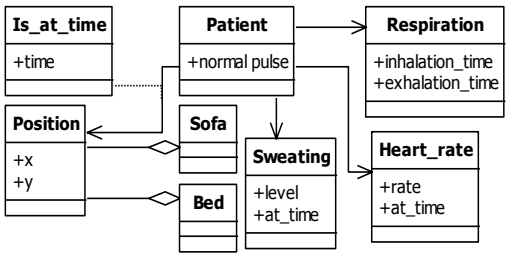

Figure 3: The data needed to verify the facts of context φ_1 shown in Figure 2

contexts are those contexts at the variation points (i) *root goal* and (ii) *And-decomposition*. They decide if a goal has to be reached or a task has to be executed. The activation context of a goal model variant is the conjunction of the contexts at the variation points of these two kinds.

2. **Required context**, when the context is necessary to adopt a certain way for achieving (executing) a set of activated goals (tasks). The contexts that are at the variation points (i) *Or-decomposition*, (ii) *Means-end*, and (iii) *Actors dependency* are required contexts. They are required to make applicable a variant of the goal model. The required context of a goal model variant is the conjunction of contexts at the variation points of these three kinds.
3. **Quality context**: when the context influences the quality of a variant of the goal model. Only the contexts at the variation point *Contribution to softgoals* are quality contexts. Contributions (links) to softgoals are, indeed, used in Tropos to capture the impact of a goal/task to a quality (i.e., softgoal).

Figure 4 shows two partial goal model variants (taken from Figure 1) and the contexts associated to them. The classification of context into these three categories allows us, amongst other things, to answer questions like: in a given context, does the system need to meet its requirements? what are the possible ways to meet them? and what is the quality of each of such ways?. In the rest of the paper, the term **context of a goal model variant** refers to the conjunction of the activation and required contexts of that variant.

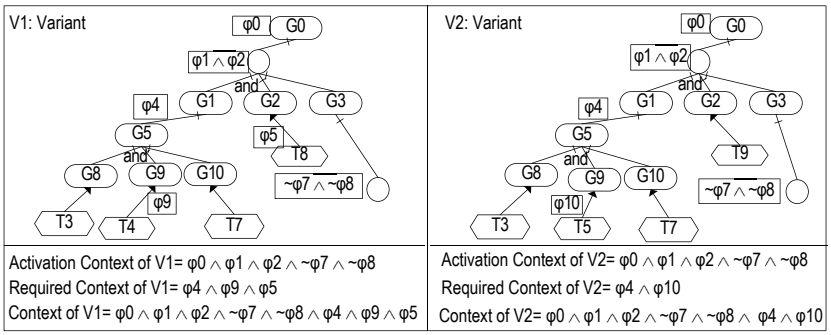

Figure 4: Two partial goal model variants and their contexts

3. Consistency Analysis

Context analysis allows us to refine contexts at the variation points of the goal model and discover formulae of facts that specify them (see Figure 2). We remind here that only monitorable contexts are allowed in our contextual goal models, i.e., the contexts that are refinable to formulae of facts. However, when deciding if a goal model variant is applicable, a conjunction of these formulae has to be verified. A formula expressing a context (or a conjunction of contexts) could be inconsistent. Inconsistencies could be modeling errors that should be fixed.

In order to check the consistency of a formula specifying a context, we need also to take into consideration all possible contradictions among the variables (world predicates) of that formula. For example, in Figure 1 we have $\varphi_7 = wp_{7.1} \wedge wp_{7.2}$ where $wp_{7.1}$ = *"patient is outside home for long time"* and $wp_{7.2}$= *"it is night time"*, and $\varphi_{10} = wp_{10.1}$ where $wp_{10.1}$ = *"it is not night time"*. In this example, $\varphi_7 \rightarrow \neg\varphi_{10}$ because $wp_{7.1} \rightarrow \neg wp_{10.1}$, so any goal model variant that whose context includes $\varphi_7 \wedge \varphi_{10}$ will be inapplicable. The logical relations between world predicates formulae (contexts) can be *absolute* or *dependent* on the characteristics of the system operational environment:

1. *Absolute* relations hold wherever the system operates. For example, given the three world predicates wp_1= *"caregiver [c] has never worked in another institute"*, wp_2= *"patient [p] is in the institute for the first day"* and wp_3= *"caregiver [c] was assigned to patient [p] some date before today"*, then $wp_1 \rightarrow \neg(wp_2 \wedge wp_3)$ holds in whatever institute the system operates in.

2. *Operational environment dependent* relations are true in a particular environment where the system operates without any guarantee that such relations hold in other operational environments. For example, lets us consider the two world predicates wp_1 = *"the temperature is less than 15 degrees at the patient's location"* and wp_2 = *"patient is at home"*. If in one institute, the heating system keeps temperature above 20 degrees then $wp_1 \rightarrow \neg wp_2$ holds always in that institute. Moreover, the operational environment itself may assure that some world predicates are always true or always false. Therefore, we have to consider a special kind of environment dependent relations: $Env \rightarrow world\_predicates\_formula$. For example, if the system operates in an institute for patients with severe dementia exclusively, then the implication $Env \rightarrow \neg wp_3$ where wp_3 = *"patient has basic dementia"* always holds.

We apply SAT-based techniques [22] to check if a formula, expressing a context, is consistent under a set of assumptions. Given a formula and a set of assumed logical relations between its variables[2], a SAT-solver checks if there exists a truth assignment for all variables that makes the conjunction of the formula and the logical relations formula satisfiable. The context specified by a formula is consistent iff such assignment exists. The pseudo-code of the algorithm (*CheckSAT*) is reported in Figure 5.

Input: context φ
Output: $\bot$ ($\top$) if φ is inconsistent/consistent
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**

Figure 5: Checking context consistency under assumptions (CheckSAT)

Obviously, the context at each variation point has to be consistent, otherwise it is a modeling error to fix. The accumulative contexts (activation, required, ...) for goal model variants could also be inconsistent. However,

[2]In this paper, we suppose that the relations between variables are manually provided.

the inconsistency of these accumulative contexts does not always indicate a modeling error and fixing or accepting such an inconsistency is an analyst's decision. The compact form of goal models integrates a large number of variants and may, as a side-effect, include variants that are not practically needed and their context inconsistency is acceptable. Moreover, the semantic of context inconsistency depends on the kind of accumulative context in which it happens. In what follows, we illustrate the above ideas via examples taken from the contextual goal model of Figure 1.

Example 1. *The inconsistency of the activation context of a goal model variant means that the variant is not needed. The variant shown in Figure 6 has an inconsistent activation context because of the contradiction between* $\varphi_1 = wp_{1.1} \wedge wp_{1.2}$*, where* $wp_{1.1}$*="patient is inside home",* $wp_{1.2}$*="patient feels anxious", and* $\varphi_7 = wp_{7.1} \wedge wp_{7.2}$*, where* $wp_{7.1}$*= "patient is outside the home area for long time" and* $wp_{7.2}$*= "it is night time". In this example, the variant is practically inapplicable and the context inconsistency is acceptable. Indeed, giving illusion of being lived in to protect home from robbery is needed when patient is outside, whereas treating his anxiety is needed when he is in the home area. However, given that these two requirements are not needed at the same time, the designers could accept the mentioned context inconsistency. In some other cases, inconsistency of activation contexts has to be fixed. Let us suppose that* φ_0 *is modified to* φ'_0 *that adds the fact "patient is at home". Therefore,* $\varphi'_0 \wedge \varphi_7$ *is inconsistent and* G_8*="give illusion of being lived in" will never be activated. In such case, the designers would decide to fix the inconsistency treating it as a modeling error.*

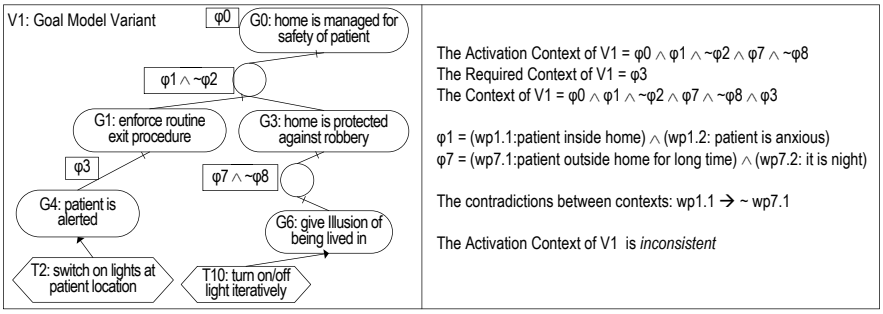

Figure 6: A partial goal model variant with an inconsistent activation context

Example 2. *The inconsistency of the required context of a goal model variant that has a consistent activation context means that the variant can be activated but it is unadoptable in any context. In other words, a set of requirements could be activated but a certain way (variant) to meet them is unadoptable. Figure 7 shows an example of inconsistent required context. In this example, the administration of the health care institute decides that calling caregiver through institute speakers requires that patient has extreme anxiety, while in the other cases caregiver could be called by phone. Therefore, φ_9 is modified into φ'_9 that adds the fact "patient anxiety is moderate" which make $\varphi'_9 \wedge \varphi_4$ inconsistent. In this new specification, the context required for calling caregiver by phone never holds and designers would decide to fix the inconsistency.*

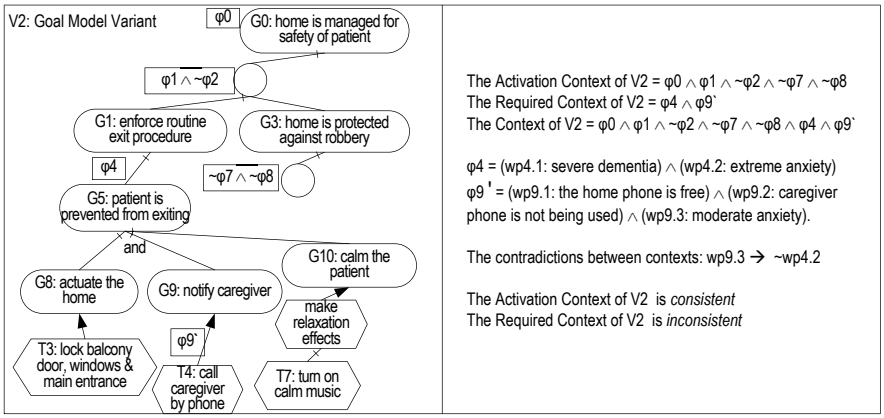

Figure 7: A partial goal model variant with an inconsistent required context

Example 3. *The inconsistency of the context of a goal model variant, when its activation and required contexts are consistent separately, means that the variant could be activated and adopted but never adopted in the context where it is activated. Figure 8 shows an example of a goal model variant with inconsistent context of this kind. In this example, the institute assigns a caregiver to each patient except for night time. This creates a contradiction between φ_0 and φ_{10} and make the context of the variant V_3 inconsistent. If T_5 does not appear in other goal model variants with a consistent context, one design decision could exclude it from the implemented system.*

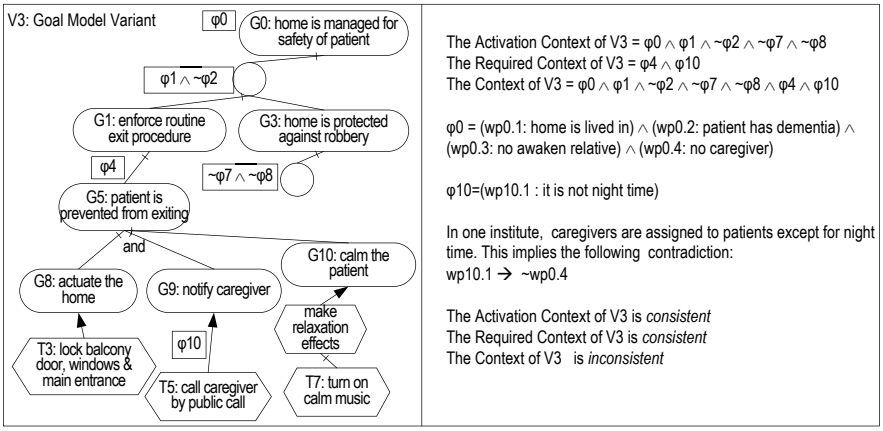

Figure 8: A partial goal model variant with an inconsistent context

Example 4. *The inconsistency in quality contexts happens when the conjunction of a context of one contribution to a softgoal and a context of a goal model variant in which this contribution exists is inconsistent. For example, the administration of some institutes could consider calling caregivers through the institute speakers has a negative impact on the softgoal "less noise" at the night hours while the impact is ignorable at the day hours. The negative contribution from T_5 to SG_3 will be preconditioned by the context φ="it is night time". Since T_5 requires day hours time then $\varphi_{10} \rightarrow \neg\varphi$ and, therefore, there will be no contribution between T_5 to SG_3 and the designers could just remove this contribution from the model.*

4. Conflict Analysis

Adaptability to context indicates a high degree of autonomy and flexibility that the system has for achieving users' goals in a variety of contexts. However, the system itself might lead to different changes over the context as a consequence of the tasks it executes to meet users' goals. These changes could be inconsistent and originate conflicts preventing the right achievement of user's goals. Understanding conflicts is preliminary for their resolution and requires to answer questions like:

- Why does a conflict occur?. In other words, what are the conflicting tasks and the goals behind them?

- What is the context in which a conflict occurs?
- Is there any alternative to avoid the conflict?
- What are the core conflicts that the system, at certain context, can not avoid? In other words, which conflicts are severe?

Most conflicts manifest themselves on a subject that is an object in the environment where the system operates [23]. In this paper, we focus on two kinds of conflicts:

- Conflicting changes: this conflict happens when two or more system executable processes (tasks in a goal model) try simultaneously to change an object in the system environment into different states. For example, the task T_8: *"open windows to circulate air"* and the task T_3: *"lock balcony door, windows, and main entrance to prevent patient of getting out"* aim to change an object, that is the windows, into two different states, *"closed"* and *"open"* respectively. If these two tasks execute in parallel, a conflicting change occurs.
- Exclusive possessing: this conflict happens when two or more executable processes need an exclusive possessing of an environment object. For example, both tasks T_{11}: *"phone police"* and T_4: *"call caregiver by phone"* need an exclusive possession of the landline phone in the patient's home. If these two tasks execute in parallel, an exclusive possessing conflict occurs.

4.1. Detecting conflicts

To analyze the two kinds of conflicts that we have mentioned, we need to enrich contextual goal models with two kinds of information:

- The effect of tasks execution on the system operational environment: we need to specify explicitly the influence of tasks execution on the objects in the system environment. For each object that the system interacts with, we need to define if the execution of a task changes the state of that object or requires an exclusive possession on it. In Figure 9, we show the influence of some of Figure 1 tasks on the patient's home objects.

Object		States
External Doors	Balcony	{open, closed, locked}
	Main Entrance	
Windows	Living room	{open, closed, locked}
	Bed room	
Lights	Living Room	{on, off, medium}
	Bed Room	
	Balcony	
Siren, Security camera, Ventilator		{on, off}

Task	Object	State	Exclusive
T1	Home speakers		true
T5	Institute speakers		true
T2	Lights	{on}	
T3	External doors	{locked}	
	Windows	{locked}	
T4	Landline		true
T11			true
T8	Windows	{open}	
T5	Institute network		false

Figure 9: Objects in the patient's home (a), and the tasks impact on them (b)

- The sequence/parallelism operators between tasks: we need to specify if two tasks, in each goal model variant, execute in parallel or in sequence. Specifying this information for each pair of tasks is obviously hard and time consuming activity. For this reason, we adopt the extension to goal model proposed in [24] where business process operators are introduced aiming at filling the gap between stakeholder goals and the business process to reach these goals. Out of these operators, we use the parallelism and sequencing operators to derive if two tasks may execute simultaneously. In Figure 10, we annotate the MobIS contextual goal model, shown in Figure 1, with these two kinds of operators.

The algorithm reported in Figure 11 processes a contextual goal model and enriches its variants with information concerning adoptability and conflicts. The algorithm extracts the goal model variants having consistent contexts (Line 2-3). The goal model variants with inconsistent contexts are excluded from further processing as they are unadoptable. Then each variant is checked for conflicts between its tasks (Line 6–14). The set of tasks belonging to each variant are extracted (Line 6) and partitioned based on the parallel execution (Line 10). Each partition of tasks is checked to know if it includes tasks changing an object in the system environment into different states (Line 11) or to exclusively posses it (Line 12). Each variant is enriched with information about conflicts happening between its tasks (Line 13). Consequently, by this reasoning we detect not only the conflicts between tasks but we also know the goals behind the tasks originating the conflicts and the context in which such conflicts happen.

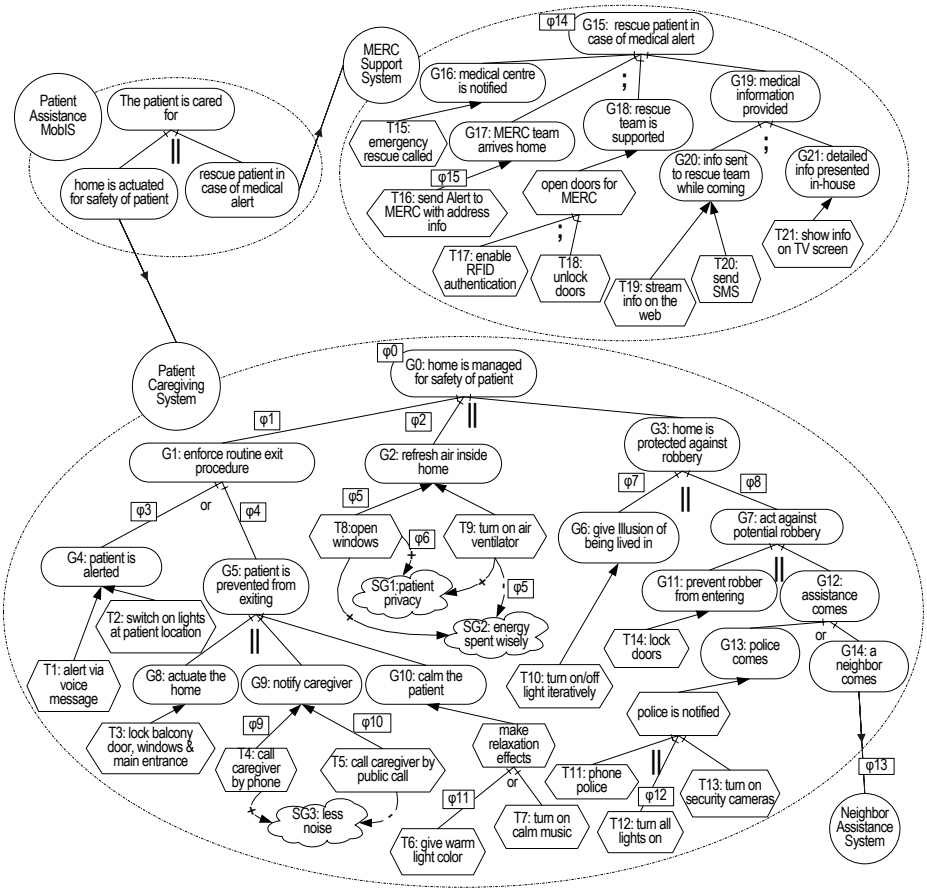

Figure 10: Goal model annotated with parallelism (||) and sequence (;) operators

Input: *S: the set of all goal model variants*
Output: *S enriched by adoptability and conflicts information*

1: **for all** $V \in S$ **do**
2: **if** $CheckSAT(V.context) = \bot$ **then**
3: $V.adoptability := \bot$
4: **else**
5: $V.adoptability := \top$
6: $T := V.set\_of\_tasks$
7: $V.conflict\_set := \emptyset$
8: **while** $|T| > 1$ **do**
9: $t_i := pop\_element\_of(T)$
10: $T_{t_i\|} := \{t_j : t_j \in T \wedge in\_parallel(t_i, t_j)\}$
11: $T_{t_i\|conflicting\_changes} := \{(t_i, t_j, o, t_i.o.state, t_j.o.state) : t_j \in T_{t_i\|} \wedge o \in Environment\ Objects \wedge t_i.o.state \neq t_j.o.state\}$
12: $T_{t_i\|exclusive\_possession} := \{(t_i, t_j, o, \text{“}exclusive\text{”}) : t_j \in T_{t_i\|} \wedge o \in Environment\ Objects \wedge t_i.o.exclusive \wedge t_j.o.exclusive\}$
13: $V.conflict\_set := V.conflict\_set \cup T_{t_i\|conflicting\_changes} \cup T_{t_i\|exclusive\_possession}$
14: **end while**
15: **end if**
16: **end for**
17: **return** S

Figure 11: Detecting conflicts in contextual goal models

4.1.1. Detecting core conflicts

Conflicts in one goal model variant can be resolved by adopting another variant that is conflict-free and applicable in all the contexts where the conflicting one is applicable. In some cases, there could be no such conflict-free variant and a resolution has to be crucially provided. In this section, we develop reasoning to discover when a conflict belongs to this kind, i.e. when it is core. We first give some basic definitions and then develop an algorithm processing a contextual goal model to detect core conflicts.

Definition 5 (Core variant). *A variant V_i with a context specified by a formula φ_i is core iff φ_i is consistent and $\nexists$ variant V_j with a context specified by a consistent formula φ_j: $(\varphi_i \rightarrow \varphi_j) \wedge \neg(\varphi_j \rightarrow \varphi_i)$.*

From this definition, any variant that is not core has a set of core variants applicable in all contexts where it is itself applicable, but not vise versa. A

reason for keeping non-core variants is that at certain context they might assure better quality[3]. The core variants are grouped on the base of the equivalence, either direct or under assumptions, of their contexts to construct core groups of variants.

Definition 6 (Core groups set). *A core groups set is a set of core variants partitioned on the base of context equivalence.*

Definition 7 (Core group of variants). *A core group of variants is an element of a core groups set.*

Example 5. *In Figure 12, we show two partial goal model variants* $\{V_1, V_2\}$*. These two variants are two ways for satisfying the goal* G_5 *each in a specific context. The contexts of these two model variants are consistent and* $V_1.context \rightarrow V_2.context \wedge \neg(V_2.context \rightarrow V_1.context)$*. This means that* V_1 *is not core since there is the variant* V_2 *that can replace* V_1 *in all the contexts where* V_1 *is applicable.*

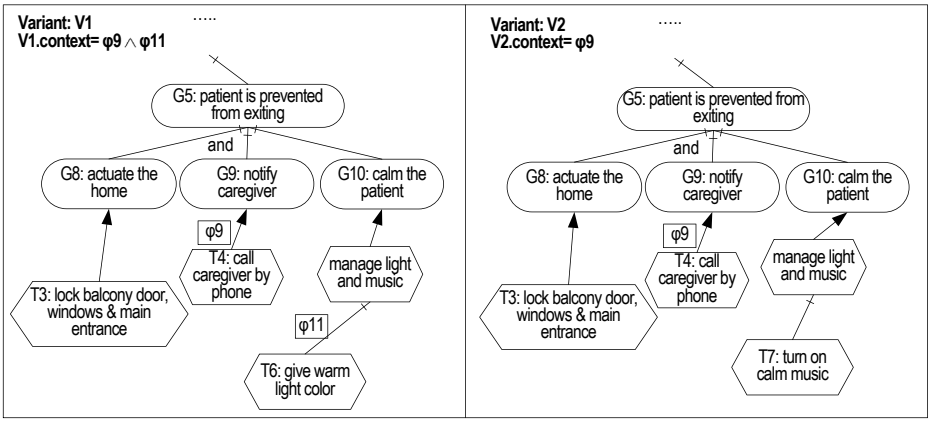

Figure 12: An example of a variant (V_1) that is not core

Having a conflict-free variant in a core group of variants means that any conflict in the other variants in the same group is not core. If all the variants in a core group of variants have conflicts, then we face a core conflict and

[3]The selection of non-core variants to keep is out of the scope of this paper.

a resolution has to be crucially provided for one, at least, of the variants in that group.

Definition 8 (Conflictual core group of variants). *A conflictual core group of variants is a core group of variants that does not include any conflict-free variant.*

Input: S: *all goal model variants set*
Output: S'':*the set of all core groups of variants with conflict*
1: $S' := Detect\_Conflict(S)$
2: $S' := S' \setminus \{V \in S : V.adoptability = \bot\}$
3: $S'' := \emptyset$
4: **while** $|S'| > 0$ **do**
5: $V := pop\_element(S')$
6: $temp := \{V\} \cup \{V' \in S' : CheckSAT(\neg(V.context \leftrightarrow V'.context)) = \bot\}$
{i.e. Check if V.context $\leftrightarrow$ V'.context}
7: $S' := S' \setminus temp$
8: **if** $\nexists\, V' \in S' : V.context \rightarrow V'.context$ **then**
9: $S'' := S'' \cup \{temp\}$
10: **end if**
11: **end while**
12: **for all** $U \in S''$ **do**
13: **if** $\exists V \in U : V.conflict\_set = \emptyset$ **then**
14: $S'' := S'' \setminus U$
15: **end if**
16: **end for**
17: **return** S''

Figure 13: Extracting the conflictual core groups of variants

The algorithm reported in Figure 13 extracts the core groups of variants in conflict from a contextual goal model. It calls the algorithm shown in Figure 11 to enrich each variant with information about adoptability and conflicts occurring in it (Line 1). The algorithm excludes the unadoptable variants, i.e., the variants with inconsistent contexts, as they are obviously not core (Line 2). The algorithm then extracts the core groups of variants (Line 4–11). To this end, the algorithm partitions the set of variants based on context equivalence (Line 6). The algorithm CheckSAT, shown in Figure 5, can be also used to check the equivalence between boolean 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 inconsistent under the assumptions ξ. Then the algorithm checks if each group is core (Line 8) and keeps it for further processing if it is like that (Line 9). The algorithm then checks each core group of variants to decide if it contains at least one conflict-free variant. If this occurred, then the group is not conflictual and it is excluded from the output set (Line 12–16).

Example 6. *As shown in Figure 14, the assumption is that the subgoals of the root goal "home is managed for patient safety" are not dependent on each other and may need to be reached in parallel when their corresponding contexts hold (notice the notation ||). The variant V_1 includes a conflict between the tasks T_3 and T_8 manifested on the environment object "windows". Each of these two tasks changes the state of this object differently as we specified in Figure 9. The variant V_2 can replace V_1 in all of its contexts since $V_1.context \rightarrow V_2.context$ which means that V_1 and its conflict are not core. An example of a core conflict is that occurring in V_3 because of the exclusive use of the environment object "phone" between the two tasks T_4 and T_{11} and the absence of variants that are adoptable whenever V_3 is adoptable and that are conflict free.*

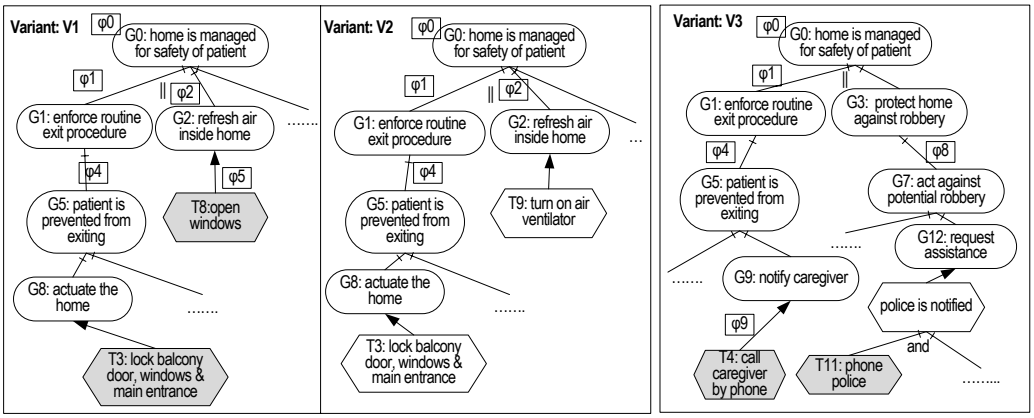

Figure 14: Non-core variant with conflict (V_1), its conflict-free alternative (V_2), and variant with core conflict (V_3)

5. Automated Support Tool

In order to support the reasoning techniques proposed in Section 3 and Section 4, we have developed a prototype automated tool called *RE-Context*. This tool takes as input a contextual goal model expressed as an input file for DLV[4], a disjunctive Datalog [25] implementation. The tool has been developed to show the usefulness of our reasoning techniques when applied in practice. Currently, we do not provide a graphical goal modeling editor and automated translation to the DLV input format.

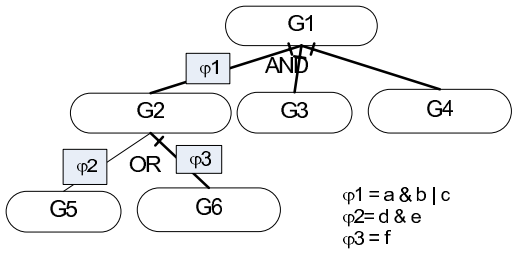

Figure 15: Sample contextual goal model

Code 2 Sample goal model of Fig 15 expressed as DLV input

```
1 ach(g2) v -phi(1) :- ach(g1).
2 phi(1) :- ach(g2).
3 todo(g3) :- ach(g1).
4 todo(g4) :- ach(g1).
5 todo(g5) v todo(g6) :- ach(g2).
6 phi(2) :- todo(g5).
7 phi(3) :- todo(g6).
8 ach(g1).
```

Code 2 shows the sample goal model of Fig 15 translated to the DLV input format. The top-level goal G_1 is and-decomposed to G_2, G_3 and G_4, but the decomposition to G_2 is subject to the context φ_1. The mapping is shown in the first four lines: whenever G_1 has to be achieved (the predicate *ach* is used for non-leaf-level goal achievement): (i) either G_2 should be achieved

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

or context φ_1 should not hold; (ii) if G_2 should be achieved, then φ_1 should hold; (iii) if G_1 should be achieved, then the leaf-level goals G_3 and G_4 has to be done (the predicate *todo* is used for leaf-level goals achievement). The or-decomposition from G_2 to G_5 and G_6 is shown in lines 5-7. If $g2$ has to be achieved, then either G_5 or G_6 should be done. If G_5 is chosen, then context φ_2 should hold; if G_6 is chosen, then context φ_3 should hold. The last line states that goal G_1 has to be achieved.

The first task of our tool is to derive all variants, and what RE-context does is to run 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, i.e. the goal model variants. In particular, we are interested in the set of tasks to be carried out (the *todo* predicates) and the contexts that should (not) hold (the *phi* predicates).

The next step is to check context consistency for each variant, which corresponds to run the *CheckSAT* algorithm described in Figure 5. To verify the consistency of a context, RE-Context uses an external tool (MathSAT[5]) that is based on MiniSat SAT solver[6]. In order to carry out this step, RE-Context loads the definition of contexts from a separate file (contexts.msat), which contains the representation of all contexts as boolean formulas expressed over a set of variables. The relations between contexts/varianles are defined in another MathSAT input file called relations.msat. In order to check an inconsistency, RE-Context takes the contexts that refer to the analyzed variants from contexts.msat and merges these formulas with the relations between contexts, then runs the SAT solver to determine the formulae consistency.

The inconsistency of individual contexts at variation points is, obviously, a modeling error to be fixed and not subject to design decisions. RE-Context checks each individual context for inconsistency and alerts the analyst when it is inconsistent. Fixing or accepting inconsistency of the accumulative contexts of goal model variants is a design decision as we have explained in Section 3. To minimize the interaction with the analysts, this activity propagates a design decision for one variant to the others when possible. Whenever an inconsistency in one accumulative context is discovered, RE-Context asks the analyst to fix or accept it showing the variant, the context, and the contradictions. If the inconsistency is accepted, RE-Context scans the rest of

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

[6]http://minisat.se/

the variants and marks the inconsistency of those containing the processed one (i.e. containment of tasks) as accepted. In order to further minimize the interaction with analysts, variants with less tasks are examined first.

The third phase consists of identifying conflicts in variants. In order to achieve this result, we need to express information concerning resources. Code 3 shows how we express the way a task changes a resource (lines 1-3) and the way a task requires a resource (lines 4-6).

Code 3 Expressing the link between tasks and resources

```
1 resource(r1).
2 changes(g6,r1,value1).
3 changes(g4,r1,value2).
4 exclusiveUsage(r1).
5 requires(g4,r1).
6 requires(g3,r1).
```

Line 1 declares $r1$ as a resource; line 2-3 say that task G_6 (G_4) changes G_1 to $value1$ ($value2$). Line 4 states that $r1$ requires exclusive usage; lines 5-6 say that task G_4 (G_3) requires resource $r1$. In our example, a variant containing both G_6 and G_4 would imply a conflicted change, whereas a variant with both G_3 and G_4 would entail an exclusive usage conflict. Moreover, we also need to express information about the sequentiality of tasks and goals: conflicts concerning resources exist only in case of tasks executing in parallel. In our formalization, we make usage of the `parallel` and `sequence` predicates in the DLV input file. The sequentiality predicates are then propagated top-down in the goal trees, in order to identify which tasks should be executed in sequence and which have to be executed in parallel. If the code in Code 3 included the predicate `sequence(g4,g3)`, there would not be the exclusive usage conflict for resource $r1$. The last step our tool currently supports is the discovery of core conflicts. In order to carry out this analysis, we implemented in RE-Context the algorithm of Figure 13.

6. Analysis Process

In this section, we explain a process to follow for constructing a contextual goal model for a system operating in and reflecting multiple contexts. The overall picture is depicted in the activity diagram in Figure 16. Four macro-activities are identified: goal analysis, context analysis, reasoning about contextual goal models, and identifying monitoring requirements.

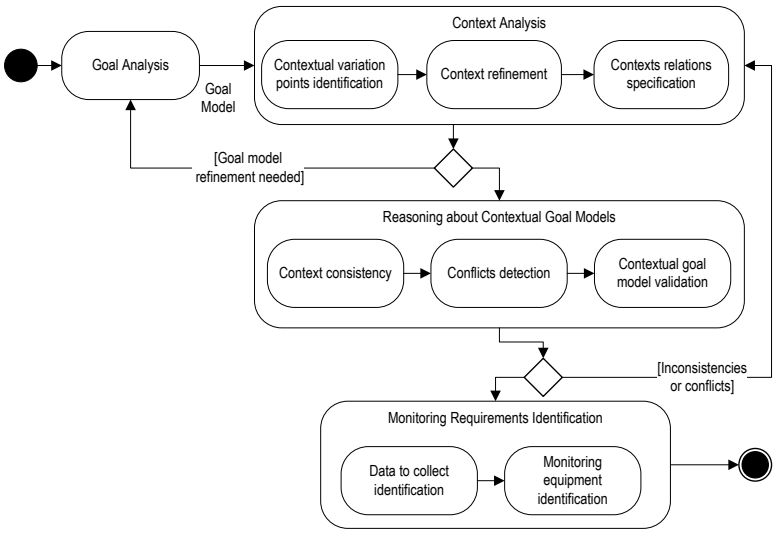

Figure 16: Analysis process for contextual goal modeling

1. **Goal analysis**: in this activity, high level goals are defined and analyzed. Goals can be iteratively identified through scenarios [26]. Moreover, an intentional variability taxonomy [27] can guide variability acquisition when refining a goal/task to discover alternative ways of reaching/executing it. Each refinement step is followed by a context analysis.
2. **Context analysis**: this activity weaves goal modeling with context aiming to link the requirements, at the goal level, to the context in which they are activated and adoptable. Context analysis activity is composed of:
   (a) *Contextual variation points identification*: for each variation point at the goal model, a decision has to be taken on whether context plays a role in the selection of variants at that point. In other words, the analyst has to decide if a variation point is contextual or context independent. When a contextual variation point is identified, a high level description of the correspondent context has to be made. As a result of this activity, the contextual variation points at the goal model are annotated as shown in Figure 1 and the contexts associated with them are described as shown in Table 1
   (b) *Context refinement*: in this activity, the contexts at each contextual variation point are analyzed. The analysis is for identifying

the way the contexts can be verified through. In other words, it is to define the facts of the environment the system has to capture and the way these facts are composed to judge if an analyzed context holds, as shown in the example of Figure 2. Moreover, in this activity, the analyst has to deal with different views of different stakeholders about the analyzed contexts. Stakeholder may define context differently, and even in contradictory ways. In case of inconsistency between stakeholder on context refinement, the analyst has to look for reconciliation among them through a consensus session.

(c) *Specifying logical relations between contexts*: after the refinement of each context, the logical relations (implications and contradictions) between it and the previously refined contexts need to be specified. These relations are essential for the forthcoming reasoning about contextual goal models. In some cases, defining these relations at the higher level of context analysis is possible, i.e. defining that the context C_1 at the variation point VP_1 is contradicted with C_2 at VP_2, as we could do in our case study. For larger contexts, we may need to specify these relations at a more fine-grained levels such as the facts level. We still do not provide an automated support for the specification of these relations and presume that this activity is done manually.

3. **Reasoning about contextual goal models**: this activity is supported by our developed automated reasoning tool (RE-Context). The tool allows reasoning about contextual goal models for different reasons. It analyzes a contextual goal model in order to detect inconsistency in contexts specified on it and potential conflicts among its executable processes (tasks). Moreover, the automated reasoning tool allows us to validate whether the model reflects stakeholders' requirements. To this end, this reasoning derives and shows to stakeholders the goal model variants that reflect a given context and user priorities. Here we detail the three kind of reasoning done in this activity:

   (a) *Reasoning about context consistency*: this reasoning is to check if a context can eventually hold. First, it has to be done for the contexts defined at each variation point. If a context of this kind is inconsistent, the analyst must either fix the inconsistency or remove the context and mark the variation point as context-independent.

The inconsistency of accumulative contexts of goal model variants are subject to design decisions as we have explained in Section 3. When an inconsistency in this kind of context is discovered, the tool asks the analyst to decide whether to fix the inconsistency or accept it and therefore exclude the correspondent goal model variant. When an inconsistency in one goal model variant is accepted, the tool (RE-Context) excludes the rest of variants that include the one being examined and mark their inconsistency as accepted as well.

(b) *Reasoning about conflicts*: in order to to enable the automated discovery of conflicts in a contextual goal model, the analyst has to enrich the model with further information. This information includes: (i) the objects in the system environment and the impact of the execution of goal model tasks on them, and (ii) the sequence/parallel operators between goals/tasks in And-decomposi tions. Adding this information, RE-Context is able to detect conflicts and classify them into two categories: Core and Non-Core. The analyst needs to resolve core conflicts crucially as this kind of conflicts leads to situations where there is no way to meet some requirements correctly.

(c) *Validating contextual goal models*: this reasoning is to ensure that the contextual goal model reflects stakeholder' expectation of the system in different contexts and in compliance with their priories. To this end, the the analyst can ask stakeholder to give a context and then show them the correspondent goal model variants. Alternatively, the analyst may ask stakeholder to derive the variant in a given context and compare it with the one obtained by our automated analysis. This test might be done for the whole goal model or parts of it. Moreover, user prioritization might be considered to select between goal model variants when more than one is adoptable in a given context. User prioritization can be specified over softgoals as proposed in [28, 29]. RE-Context currently supports the activity of deriving goal model variants for a given context and prioritization.

The reasoning about contextual goal model can be done iteratively to facilitate the correction of discovered errors. Whenever a new refinement of goal/context is done, we can start with the above reasoning techniques. With regards to the reasoning for conflicts, it may be hard

to decide the interaction between goal model variants and the objects in the system environment until we reach the tasks level. For this reason, we suggest to analyze parts of the goal model that have high probability of being in conflict first. We refine such parts until the tasks level and then we run the conflict analysis reasoning on them.

4. **Identifying monitoring requirements**: after context analysis and reasoning terminate, the analyst can identify the monitoring requirements. Monitoring requirements are fundamental to develop a contextual MobIS. We identify these requirements in terms of the data to collect from the system environment and the equipments needed to collect them.
   (a) *Identifying the data to collect*: by analyzing the facts obtained by context analysis, the analyst can identify the data needed to verify them as shown in Fig 3. We suggest to keep track of the relation between each facts and the fragment of the data conceptual model needed to verify it. This link is important to promote reusability and modifiability of the contextual goal model. In case a part of the context refinement is reused/modified, we will be able to identify which fragments of the data model are to be reused/modified. Moreover, we have used class diagrams to represent the data conceptual model in this paper. Recently, some other models have been proposed to represent the data to monitor in context-aware systems, e.g., the models proposed in [30, 31, 32]. The analyst may select one of these models instead of class diagrams if needed.
   (b) *Identifying the monitoring equipments*: for systems operating in and reflecting varying context it might be essential to specify the equipments to install and to use in order to enable data collection. This activity is for the specification of equipments needed to capture the data identified in the previous activity. For example, the analyst needs to specify the kinds of sensors to use, the topology of sensor distribution, the interval of data sensing, and so on. To achieve such specification, expertise in new technology is needed. In Table 1, we have given a brief description of the equipments needed for each of the contexts at the goal model. However, such specification of equipments becomes more precise after knowing the data to monitor in the environment, i.e, after the previous activity.

7. Evaluation

To evaluate our modeling and reasoning techniques—implemented in RE-Context—we have invited a group of five requirements engineers to participate in an evaluation lab session. All the participants have good experience in goal modeling and MobIS. We explained our framework including the conceptual model and the reasoning techniques. A domain expert with practical experience in a health-care system owned by ITEA[7] has explained the scenario to all participants and answered their questions during the development of the model.

The participants were asked to capture the MobIS requirements using our proposed contextual goal model with the purpose of evaluating the way the participants use the model and the results obtained and the performance of RE-Context. The models were iteratively checked to detect context inconsistency and take design decisions (i.e., fix or ignore inconsistency). The final model was processed by RE-Context to detect mainly the conflict and the core conflict information. The results we got by applying our reasoning on the resulted model are reported in Table 2.

The first two columns show the time required to develop (TD) and formalize/fix (TF&F) the goal model. Then, the goal model size is described in terms of the number of actors (NA), goals (NG), tasks (NT), soft-goals (NSG), variation points (NVP) and variants (NV). After this input information, the figure contains data collected by running the tool. First, the tool deals with inconsistency: how many times the designers have been asked to fix or accept an inconsistency (Iterations), how many times they fixed an inconsistency (Fixing), and how many variants with inconsistent contexts left (VwAccInc). After inconsistency checking, the tool processes the variants with consistent contexts. We present the number of conflicts (NC), the number of variants with conflicts (VwC), the number of core groups of variants (CGV) and the number of conflictual core groups (CCGV).

During the experiment, the attitude of participants was observed and then they have been interviewed to evaluate our engineering framework from the perspective of practitioners acceptance. The participants are experts in goal modeling and the focus of the experiment has been on the extension we proposed that includes mainly the variation points and context analysis constructs. For simple contexts, the participants did not need the context

[7]ITEA is the Social Housing Agency of the Province of Trento. www.itea.tn.it

Time		Goal Model Size						Context Inconsistency			Conflicts				
TD	TF&R	NA	NG	NT	NSG	NVP	NV	Iterations	Fixing	VwAccInc	NonCV	NC	VwC	CGV	CCGV
14 Hours	6 Hours	5	35	50	5	25	25560	27	3	11556	1908	29	13789	192	184

Legend	
TD: time to capture the requirements and draw the graphical model. TF&F: time needed to formalize the model+ time to fix/accept inconsistency in context specification. NA: number of actors. NG: number of goals. NT: number of tasks. NSG: number of softgoals. NVP: number of variation points. NV: number of variants.	Iterations: number of iterations to fix/accept inconsistency. I.e., how many times the analyst has been interacted. Fixing: number of times the analyst fixed an inconsistency. VwAccInc: number of variants with accepted inconsistency. NonCV: number of non-core variants. NC: number of conflicts. VwC: number of variants with conflicts. CGV: number of core groups of variants. CCGV: number of conflictual core groups of variants.

Table 2: The results obtained by applying our reasoning mechanisms on the studied MobIS

analysis constructs, and it was possible to specify those contexts at the level of facts directly. Most of the relations between contexts (implications and contradictions) were applicable directly between contexts specified at the variation points, and we rarely needed to specify them at the fact or intermediate levels. In spite of that, the experiment showed a need for more automated support of this activity, i.e., the activity of specifying the relations between contexts. For complex contexts, there has been disagreement (viewpoints) between the participants for the refinement of context. In some other cases, it was debatable if a certain world predicate is a statement or a fact. Therefore, new policies to manage such situations are still required. Temporal relations are still missed in our context analysis. Moreover, the analysis could be more expressive if weighted/probabilistic *Support* relations between formulae and refined statements are introduced.

In order to assess the performance of RE-Context, we installed it on a machine with two CPUs AMD Athlon(tm) 64 X2 Dual Core Processor 5000+ and 4 GB of RAM. Figure 17 reports the results of the performance analysis with respect to the time needed (in milliseconds) to perform reasoning. The first two columns represent the size of the goal model as number of nodes (goals plus tasks) and number of variants; then, the table reports the time needed to derive all variants (T_Deriv), to identify inconsistency (T_Inc), to get the core groups of variants (T_CGV). The time to compute the core groups of variant with conflicts is negligible in comparison to T_CGV. RE-Context scales well for medium-size goal models, e.g. it took us 30 minutes to process a goal model of the case study that contained 90 nodes and 25,560 variants.

To get goal model of large size, we adopted an approach similar to [33]: we cloned the original goal model that we have developed for the MobIS for patient with dementia. RE-Context needed 40 minutes for a goal model of 100,000 variants. As shown in Figure 17, the derivation of goal model variants and the inconsistency check scale quite well, whereas the identification of core groups of variants has scaling limitations for large-scale goal models. The number of nodes is not a critical factor for scalability, whereas the number of variants and the relations between contexts are crucial. Since our proposed reasoning is at design time, time is not a critical problem for medium size goal models. However, we still need optimization of the algorithms used to minimize the complexity when dealing with very large goal models.

Size of goal model		T_Derive	T_Inc	T_CGV
NN	NV			
18	3	62	3	5
30	12	79	18	10
42	108	273	53	288
49	540	582	195	3826
64	2565	1224	1351	23076
79	4275	2484	2009	59221
90	15300	7553	3926	100339
90	25560	10424	12006	1819126
150	104976	21861	63868	2348941

Legend

NN : the number of nodes in the processed model.
NV : the number of variants in the processed model.
T_Derive: time to derive all variants (in ms).
T_Inc: time to get all variants with inconsistent context.
T_CGV : time to get the core groups of variants.

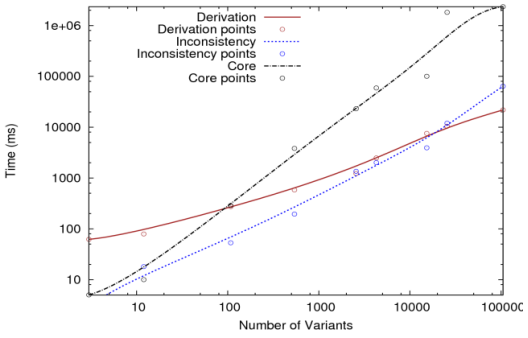

Figure 17: Tabular and Graphical representation of the performance of RE-Context

To deal with scalability problems for very large goal models, we might benefit of two techniques. The first is the iterative check of the model during construction. We can reason about consistency and conflicts while constructing the goal model instead of treating the entire final goal model at once. In such a way, problems are identified as soon as they arise and can fixed immediately. The second is by using divide and conquer techniques. Computing the core groups of variants could be complex due to the high number of invoking SAT solver. A way to reduce this complexity, is by dividing the model into parts, reasoning about each part separately, and then combining the results. For example, for an And-decomposed goal, we can compute the core groups of variants of each subgoal and then combine the results by cartesian product.

8. Related Work

The research in context modeling, (e.g., [32]), concerns finding modeling constructs to represent software and user's context, but there is still a gap between the context model and the requirements model, i.e. between context and one of its usages. We tried to reduce such 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 requirements?*".

Software variability modeling, mainly feature models [34, 35], concerns capturing a variety of possible configurations of software functionalities. This would 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 such functionality can or has to be adopted. We have tried to solve this problem at the goal level. Furthermore, our work is in line, and has the potential to be integrated, with the work in [36] and the FARE method proposed in [37] that show possible ways to integrate features with domain goals and knowledge to help for eliciting and justifying features.

Feature interaction research (for a survey see [38]) concerns predicting scenarios in which an interaction between system features occurs, and judging if an interactions is harmful and providing resolution mechanisms if this is the case. Recently, Nhlabatsi et al. [23] observed, following a large survey of the literature, that the feature interaction problem is essentially a problem on shared context; i.e. there is no interaction without a subject that is an object in the system environment. In line with this observation, our model supports an explicit notion of context and captures how context influences, and how it is influenced by, the requirements at the goal level. We develop reasoning mechanisms that use the model to detect and provide essential information about the conflicts occurring between system executable processes (tasks). This information includes the goals for which and the context in which a conflict happens, the alternatives the system has to avoid conflicts and so on.

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 [15]. The objective is to have more robust, maintainable, and self-evolving systems. In [39], the GORE (Goal-Oriented Requirements Engineering) framework KAOS [13] was integrated with an event-monitoring system (FLEA [40]) to provide an

architecture that enables a runtime automated reconciliation. The developed reconciliation is between the system goals and the system behavior with respect to a priori anticipated or evolving changes in the system environment that is mostly technical. In our work, we start earlier and capture the influence of context on users goals. Such context concerns the environment, that is not necessarily technical, of the user and the system. Moreover, we provide a conceptual modeling language that supports an explicit notion of context and a systematic way to analyzed it in conjunction with goal model. Our work could be integrated with FLEA towards more holistic reconciliation between system and user goals from one side and system and user contexts from the other.

Customizing goal models to fit to user skills and preferences was studied in [29, 41]. 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 expressed over softgoals. Lapouchnian et al. [42] propose techniques to design autonomic software based on an extended goal modeling framework, but the relation with context is not focused on. Liaskos et al. [27] study variability modeling under the requirements engineering perspective and propose a classification of the intentional variability that originate goal satisfaction alternatives. We focused on context variability, i.e. the unintentional variability, that highly influences the applicability and quality of each goal satisfaction alternative. Reasoning about Tropos goal model has been already studied in [43]. We need to revise the proposed reasoning to take the dimension on context in goal modeling into account.

Salifu et al. [44] apply Problem Frames approach to analyze different specifications, that can satisfy the core requirements, under different contexts. The relationship between contexts, requirements, and the specification (machine) are represented by a problem description. Alternative problem descriptions corresponding to different contexts are elicited to identify variant problems. Variant problems are variations of the original problem adapted for a particular context. Hartmann et al. [45] suggest studying the relation between context and features to support the engineering of software supply chains. Their approach allows for more systematic derivation of a product that fits to the environment in which it operates. In our position paper [46], we suggested the integration of our work with the above works that considered adaptability to context and we showed, theoretically, how such integration could help for holistic software production.

9. Conclusions and Future Work

In this paper, we have proposed a requirements engineering framework for modeling and reasoning about MobISs requirements. These systems operate in dynamic contexts and their requirements depend upon the current context. Our baseline is Tropos goal modeling, which captures alternatives to goal satisfaction and their qualities. We extended the Tropos goal model to weave together these alternatives and the contexts in which the system could operate. Context is defined through a hierarchical analysis. This analysis helps for a systematic way in identifying what information the system has to collect of its environment to judge if a context holds.

We have proposed analysis techniques and automated reasoning mechanisms to verify properties of contextual goal models. We have developed a reasoning technique to detect inconsistency between contexts specified on a goal model and discussed the semantics of different kinds of detected inconsistencies. We have also developed a reasoning mechanism to detect and assess the severity of conflicts originating from the context changes. The framework has been evaluated on a case study of a MobIS designed for people with dementia. We have evaluated our framework both qualitatively and quantitatively via stakeholder involvement and scalability analysis, respectively.

In future work, we plan to address various problems concerning contextual-requirements engineering, such as:

- **Reasoning about monitoring requirements:** the system at runtime needs to collect environmental data to judge if a certain context holds and adapt to it by adopting a suitable behavior. This raises new category of requirements called Monitoring Requirements, i.e., what data the system has to capture from its environment and the way these data are logically composed to judge if a certain context holds. Monitoring requirements need specific analysis. An example of such analysis is the optimization of monitoring requirements, i.e., finding the less expensive set of data to capture required to verify if a given context holds [47]. For example, a context specified as $(B \wedge C)$ where $B =$ "patient is inside home" and $C =$ "it is cold at the patient's location" can be reduced into (B) if the healthcare institute regulates temperature inside homes in way that prevents going lower than a certain level. A positioning system would suffice to verify if context A holds, for the system can verify B whereas C is not needed.

- **Lifelong contextualization**: a context-aware MobIS has to monitor context at runtime and adapt to it. It would be even better if the system could evolve over time and enhance the way it satisfies user's needs in different contexts. Indeed, not all decisions can be fully specified at design time. A running system might collect data and learn which requirements and which alternatives (species) fit better to particular environments. For example, patients could prefer remote communication with caregiver in one context and in person in some other context. This knowledge is typically unavailable at design time, for considered patients might change their habits and unknown patients might use the system. The system can therefore evolve choosing the best way of communication benefiting of the history of the patient's behavior in different contexts.

- **Automated support of modeling activities**: a CASE tool is needed for supporting all the modeling activities proposed in this paper. For example, we still transform the graphical representation of contextual goal models into Datalog manually. Moreover, we presume that the logical relations between contexts, i.e., contradictions and implications, are also manually specified by the designers. Defining these relations for small–medium size systems could be doable manually. For larger systems, manual specification is error-prone and time consuming.

Acknowledgement

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

References

[1] J. Krogstie, K. Lyytinen, A. Opdahl, B. Pernici, K. Siau, K. Smolander, Research areas and challenges for mobile information systems, International Journal of Mobile Communications 2 (2004) 220–234.

[2] J. Kramer, J. Magee, Self-managed systems: an architectural challenge, Future of Software Engineering, 2007. FOSE'07 (2007) 259–268.

[3] P. Oreizy, N. Medvidovic, R. N. Taylor, Runtime software adaptation: Framework, approaches, and styles, in: Companion of the 30th international Conference on Software Engineering (ICSE Companion '08), ACM, New York, NY, 2008, pp. 899–910.

[4] A. Schmidt, Implicit human computer interaction through context, Personal and Ubiquitous Computing 4 (2000) 191–199.

[5] S. Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, Y. Berbers, Easy: Efficient semantic service discovery in pervasive computing environments with qos and context support, The Journal of Systems & Software 81 (2008) 785–808.

[6] J. Krogstie, Requirements engineering for mobile information systems, in Proceedings of the Seventh International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'01) (2001).

[7] M. Jackson, Problem Frames: Analyzing and structuring software development problems, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.

[8] A. Van Lamsweerde, Goal-oriented requirements engineering: A guided tour, in: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, pp. 249–263.

[9] E. Yu, J. Mylopoulos, Why goal-oriented requirements engineering, in: Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, pp. 15–22.

[10] E. Yu, Modelling strategic relationships for process reengineering, Ph.D. Thesis, University of Toronto (1995).

[11] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An agent-oriented software development methodology, Autonomous Agents and Multi-Agent Systems 8 (2004) 203–236.

[12] J. Castro, M. Kolp, J. Mylopoulos, Towards requirements-driven information systems engineering: The tropos project., Information Systems 27 (2002) 365–389.

[13] A. Dardenne, A. van Lamsweerde, S. Fickas, Goal-directed requirements acquisition, Sci. Comput. Program. 20 (1993) 3–50.

[14] J. Mylopoulos, L. Chung, E. Yu, From object-oriented to goal-oriented requirements analysis, Commun. ACM 42 (1999) 31–37.

[15] S. Fickas, M. Feather, Requirements monitoring in dynamic environments, in: Proceedings of the Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society Washington, DC, USA, p. 140.

[16] D. Sykes, W. Heaven, J. Magee, J. Kramer, From goals to components: a combined approach to self-management, in: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, ACM New York, NY, USA, pp. 1–8.

[17] R. Ali, F. Dalpiaz, P. Giorgini, Location-based software modeling and analysis: Tropos-based approach, in: Q. Li, S. Spaccapietra, E. S. K. Yu, A. Olivé (Eds.), ER, volume 5231 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 169–182.

[18] R. Ali, F. Dalpiaz, P. Giorgini, Location-based variability for mobile information systems, in: Z. Bellahsene, M. Léonard (Eds.), CAiSE, volume 5074 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 575–578.

[19] R. Ali, F. Dalpiaz, P. Giorgini., A goal modeling framework for self-contextualizable software., in: In the Proceedings of the 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD09), LNBIP 29-0326, Springer, 2009, pp. 326–338.

[20] R. Ali, F. Dalpiaz, P. Giorgini, Contextual Goal Models, Technical Report DISI-10-020, DISI, University of Trento, Italy., 2010.

[21] S. Campadello, L. Compagna, D. Gidoin, S. Holtmanns, V. Meduri, J.-C. Pazzaglia, M. Seguran, R. Thomas, Serenity Deliverable A7.D1.1. Scenario selection and definition, Technical Report, 2006.

[22] A. Biere, M. J. H. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009.

[23] A. Nhlabatsi, R. Laney, B. Nuseibeh, Feature interaction: the security threat from within software systems., Progress in Informatics (2008) 75–89.

[24] A. Lapouchnian, Y. Yu, J. Mylopoulos, Requirements-driven design and configuration management of business processes., in: Proceedings of the 5th International Conference on Business Process Management (BPM 2007), volume LNCS Vol. 4714, Springer-Verlag, 2007, pp. 246–261.

[25] T. Eiter, G. Gottlob, H. Mannila, Disjunctive datalog, ACM Transactions on Database Systems 22 (1997) 364–418.

[26] C. Rolland, C. Souveyet, C. Achour, Guiding goal modeling using scenarios, IEEE Transactions on Software Engineering 24 (1998) 1055–1071.

[27] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, J. Mylopoulos, On goal-based variability acquisition and analysis, in: Proc. 14th IEEE Intl. Requirements Engineering Conference (RE'06), pp. 76–85.

[28] F. Dalpiaz, P. Giorgini, J. Mylopoulos, An architecture for requirements-driven self-reconfiguration, in: P. van Eck, J. Gordijn, R. Wieringa (Eds.), CAiSE, volume 5565 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 246–260.

[29] B. Hui, S. Liaskos, J. Mylopoulos, Requirements analysis for customizable software: A goals-skills-preferences framework, in: Proceedings of the 11th IEEE International Conference on Requirements Engineering, IEEE Computer Society Washington, DC, USA, pp. 117–126.

[30] X. H. Wang, D. Q. Zhang, T. Gu, H. K. Pung, Ontology based context modeling and reasoning using owl, in: Proc. Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp. 18–22.

[31] S. S. Yau, J. Liu, Hierarchical situation modeling and reasoning for pervasive computing, in: Proc. Fourth IEEE Workshop on Software

Technologies for Future Embedded and Ubiquitous Systems (SEUS '06), pp. 5–10.

[32] K. Henricksen, J. Indulska, A software engineering framework for context-aware pervasive computing, in: Proc. Second IEEE Intl. Conference on Pervasive Computing and Communications (PerCom'04), pp. 77–86.

[33] Y. Wang, S. McIlraith, Y. Yu, J. Mylopoulos, An automated approach to monitoring and diagnosing requirements, in: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ACM New York, NY, USA, pp. 293–302.

[34] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005.

[35] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh, Form: A feature-oriented reuse method with domain-specific reference architectures, Ann. Softw. Eng. 5 (1998) 143–168.

[36] Y. Yu, J. do Prado Leite, A. Lapouchnian, J. Mylopoulos, Configuring features with stakeholder goals, in: Proceedings of the 2008 ACM symposium on Applied computing, ACM New York, NY, USA, pp. 645–649.

[37] M. Ramachandran, P. Allen, Commonality and variability analysis in industrial practice for product line improvement, Software Process: Improvement and Practice 10 (2005) 31–40.

[38] M. Calder, M. Kolberg, E. Magill, S. Reiff-Marganiec, Feature interaction: a critical review and considered forecast, Computer Networks 41 (2003) 115–141.

[39] M. Feather, S. Fickas, A. Van Lamsweerde, C. Ponsard, Reconciling system requirements and runtime behavior, in: Proceedings of the 9 th international workshop on Software specification and design (IWSSD'98), ACM, New York, USA.

[40] D. Cohen, M. Feather, K. Narayanaswamy, S. Fickas, Automatic monitoring of software requirements, in: Proceedings of the 19th international conference on Software engineering, ACM New York, NY, USA, pp. 602–603.

[41] S. Liaskos, S. McIlraith, 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.

[42] A. Lapouchnian, Y. Yu, S. Liaskos, J. Mylopoulos, Requirements-driven design of autonomic application software, in: Proc. 2006 conference of the Center for Advanced Studies on Collaborative research (CASCON '06), 7, ACM, 2006.

[43] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani, Reasoning with goal models, in: Proc. 21st Intl. Conf. on Conceptual Modeling (ER'02), pp. 167–181.

[44] M. Salifu, Y. Yu, B. Nuseibeh, Specifying monitoring and switching problems in context, in: Proc. 15th Intl. Conference on Requirements Engineering (RE'07), pp. 211–220.

[45] H. Hartmann, T. Trew, Using feature diagrams with context variability to model multiple product lines for software supply chains, in: 12th International Software Product Line Conference, IEEE Computer Society, 2008, pp. 12–21.

[46] R. Ali, Y. Yu, R. Chitchyan, A. Nhlabatsi, P. Giorgini, Towards a unified framework for contextual variability in requirements, in: Proceedings of the 3rd International Workshop on Software Product Management (IWSPM09).

[47] R. Ali, A. Franzen, A. Griggio, P. Giorgini., Modeling and Analyzing Contextual Requirements, Technical Report, Technical Report DISI-09-019, Ingegneria e Scienza dell'Informazione, University of Trento., 2009.