Software and Systems Modeling (SoSyM) manuscript No.
(will be inserted by the editor)

Requirements-Driven Deployment

Customizing the Requirements Model for the Host Environment

Raian Ali[1], **Fabiano Dalpiaz**[2], **Paolo Giorgini**[2]

[1] Bournemouth University, UK
[2] University of Trento, Italy

Received: date / Revised version: date

Abstract Deployment is a main development phase which configures a software to be ready for use in a certain environment. The ultimate goal of deployment is to enable users to achieve their requirements while using the deployed software. However, requirements are not uniform and differ between deployment environments. In one environment, certain requirements could be useless or redundant, thereby making some software functionalities superfluous. In another environment, instead, some requirements could be impossible to achieve and, thus, additional functionalities would be required. We advocate that ensuring fitness between requirements and the system environment is a basic and critical step to achieve a comprehensive deployment process. We propose a tool-supported modelling and analysis approach to tailor a requirements model to each environment in which the system is to be deployed. We study the case of contextual goal model, which is a requirements model that captures the relationship between the variability of requirements (goal variability space) and the varying states of a deployment environment (context variability space). Our analysis relies on sampling a deployment environment to discover its context variability space and use it to identify loci in the contextual goal model where a modification has to take place. Finally, we apply our approach in practice and report on the obtained results.

Key words Requirements Engineering, Contextual Requirements, Deployment, Context-sensitive Systems Modeling

1 Introduction

Deployment is a main development phase that, among other things, customizes a software to fit its operational environment. The goal of deployment is that software becomes ready for use in its environment and users find it a valid and efficient way to satisfy their requirements. Traditionally, software deployment concerns the technical configuration of an implemented system, so as to fit to an instalment environment which includes specific computing resources, operating systems, etc. Though essential, we argue that having a system correctly assembled and fitting its technical environment is only a part of the deployment process. Deployment has to ensure fitness between requirements and the system environment. Lack of such fitness could lead to an undesirable, and possibly harmful, mismatch between software and its development purpose. Let us take as an example a requirement like "get user input". This requirement is achievable via two alternative requirements: "by voice recognition" and "by click-based dialogue". If the deployment environment is often noisy then the alternative "by voice recognition" is inapplicable regardless a correct deployment of software functionalities, settings and equipments.

The software environment includes whatever provides a surrounding within which the software operates. The state of such environment is denoted by the notion of *context* [1]. Context is variable and requirements are highly affected by context variability. The system has to monitor context at runtime and decide upon which requirements to activate, which software alternatives are applicable and can satisfy the activated requirements, and the quality of each of these alternatives. Specifying and implementing the relationship between context and requirements is essential to obtain software systems which meet user requirements in a dynamic environment. However, the space of context variability differs from one deployment environment to another. Each environment exhibits a specific contextual variability space and, thus, the requirements have to be customized so as to maximize the fitness with the environment at hand.

Goal models (*i** [2], Tropos [3,4], KAOS [5], and GRL [6]) are an intentional ontology used at the early requirements analysis phase to explain the *why* of a software system. Goal models have been used to represent the rationale of both humans and software systems [7] and they provide useful constructs to analyse high level requirements (goals) and discover ways to satisfy them. Such features are essential for the analysis and design of a software system that reflects the stakeholders' rationale and their adaptation to a dynamic sys-

tem environment [8,9]. Moreover, goal models help to identify alternative software functionalities to satisfy goals. This feature is essential to adapt to context variability of a deployment environment so that a change in context would cause a switch to a suitable alternative to fulfil requirements. However, the context variability profile of a deployment environment makes some of these alternatives redundant and useless. Moreover, the space of alternatives may be unable to accommodate certain contexts occurring in the environment. The discovery of these cases is important to determine whether extra functionalities have to be developed and deployed.

In this paper, we advocate that software customization should be first studied at the requirements level. The reason is that requirements are not uniform and differ according to different factors. Hui et al. [10] describe how requirements can be customized to fit the actual skills of users. Liaskos et al. [11,12] discuss customizing the requirements to users' preferences expressed over non-functional and optional requirements. Baresi et al. [13] propose adaptation requirements, called adaptation goals, to customize a system at runtime and maintain a correct execution course. Pourshahid et al. [14] study the alignment between requirements, expressed as goals, and the business process of an organization based on monitoring a set of Key Performance Indicators (KPIs) and then decide whether an adaptation action should take place. The context variability of a system environment is also a very important factor which highly affects the customization of requirements and we focus on it in this paper.

In our previous work, we have proposed contextual goal models to capture the relation between requirements variability and the variability of context [15–17]. Context is specified at a set of variation points on the goal model and its truth value defines which requirements are applicable and active. We refine high-level descriptions of context into concrete and monitorable criteria to judge if the analyzed context holds. Moreover, we have developed reasoning mechanisms for (i) runtime derivation of software configurations to satisfy goals that are applicable in a monitored context and able to accommodate user priorities, and (ii) design time identification of the least expensive alternative which guarantees goals satisfaction in the considered space of contexts [18] and (iii) detecting modelling errors which cause inconsistent requirements models [19]. Contextual goal models capture the relationship between the variability spaces of both requirements and context. Each deployment environment exhibits a different context variability space and this makes some parts of the contextual goal model useless, redundant, or insufficient. In this work, we address the customization of requirements, expressed via contextual goal models, to the contextual variability which characterizes a deployment environment.

In this paper, we propose an engineering approach for customizing requirements models to fit their deployment environments as an essential step for a comprehensive and complete systems deployment process. We consider the case of contextual goal models, which is a requirements model that explicitly captures the relation between variability of both requirements, expressed as goals, and the state of the system surrounding environment (its context). We develop reasoning mechanisms to analyse a contextual goal model given a certain deployment environment and customize it by (i) removing redundancy and uselessness, and (ii) adding default backup alternatives when the model lacks of alternatives that accommodate certain context variations. We develop a CASE tool to automate our reasoning and propose a methodological process to support a systematic customization of requirements. We evaluate our approach in practice and discuss the results.

The paper is structured as follows. In Section 2 we review contextual goal models and present our research questions. In Section 3 we introduce our approach for requirements-driven deployment and explain our CASE tool. In Section 4 we propose a process model for requirements-based deployment. In Section 5 we apply our approach in practice and discuss the results. In Section 6 we discuss related work, and in Section 7 we conclude the paper and present our future work directions.

2 Background and Research Question

Each system is situated in a possibly dynamic environment. In our previous work [15,16], we have observed that the dynamism of a system environment affects whether a requirement needs to be achieved, restricts the space of adoptable alternatives to achieve it, and affects the quality of each of these alternatives. We have advocated the need to weave together context and requirements. We have proposed to specify context (we called it "location" in that work) as a precondition at certain variation points of a goal model. We have also proposed basic automated analysis to (i) derive the set of adoptable system's alternatives for a certain context and (ii) determine the context that supports a given alternative. In [17,20], we have proposed the context analysis model, which includes constructs (statement, fact, support, decomposition) to hierarchically refine context into a formula of monitorable facts. We also proposed a way to derive the contextual workflow of the tasks of a contextual goal model.

In [18], we have proposed an automated analysis to reason about contextual goal models and derive, at runtime, alternatives which best fit both a monitored context and the preferences of a user expressed as ranking over softgoals. In the same work, we also developed a design-time reasoning for deriving systems' alternatives with minimal development costs. In [19], we have developed automated analysis to check the consistency of context in a contextual goal model. We have also developed an analysis to check the conflicts arising from the parallel execution of the software actions (goal model tasks). A first version of a CASE tool was implemented to perform the above four types of reasoning.

Research question and contribution. In this work, we address a different and important research challenge concerning the analysis of contextual requirements to fit a certain deployment environment. We analyse the fitness between a contextual goal model, which captures the relation between requirements and a space of context variations, and the space

of contexts occurring in and characterizing a certain deployment environment. That is, we customize a contextual goal model to the characteristics of a specific deployment environment. We provide a systematic way to explore that environment and an automated analysis to customize the requirements model. The customization here means removing parts of the model which are unusable or redundant in a specific environment and, also, indicating loci in the model where an augmentation is needed to cover cases where the system is unable to meet its requirements in that environment. We also extend our CASE tool (RE-Context) to support the proposed deployment analysis (unusability, redundancy, and augmentation) and evaluate our approach on a system for booking meeting rooms in an academic environment. Table 1 highlights the relations between our previous work and this paper.

2.1 Weaving together context and goals

In this section, we describe contextual goal models [15–17], our framework to model contextual requirements. As a running example, we consider a mobile information system scenario for promoting products to customers in a shopping mall. The main goal of this system is to promote a set of products to the customers that are inside the shopping mall. Both customers and sales staff are provided with PDAs as communication and interaction devices. The system can satisfy its main goal "promoting a product to a customer" through different execution courses. The adopted execution course depends on the context that may include the state of customers, products, sales staff and other elements in a shopping mall.

In Fig. 1, we show an example of a Tropos contextual goal model representing a part of the promotion information system. Tropos goal analysis [3, 4] views the system as a set of interdependent actors, each having its own strategic interests (*goals*). Goals represent requirements at the intentional level and are analysed iteratively in a top-down way to identify the more specific subgoals needed for satisfying the higher-level goals. Goals can be ultimately satisfied by means of executable processes (*tasks*).

The actor standing for a customer's mobile information system ("Customer IS") has the top-level goal "promote product [p] to customer [c] in mall [m]". Goals are iteratively decomposed into subgoals by AND-Decomposition (all subgoals should be achieved to fulfil the top goal) and by OR-Decomposition (at least one subgoal should be achieved to fulfil the top goal). The goal "sales staff [ss] delivers a product [p] sample to customer [c]" is AND-Decomposed into "[ss] is notified" and "[ss] meets [c]". The goal "promote by giving free sample of product [p] to customer [c]" is OR-Decomposed into "[c] gets [p] sample of machine [mc]" and "deliver [p] sample to [c] by sales staff [ss]". Goals are ultimately satisfied by means of executable tasks. For example, the goal "[c] knows about [p] and confirm the sample offer" can be satisfied by one of the tasks "interact via voice dialogue and voice recognition" and "interact via visual dialogue and click-based interaction".

A dependency indicates that an actor (*depender*) relies on another (*dependee*) to attain a goal or to execute a task: the actor "Customer IS" depends on the actor "Sales Staff IS" for achieving goal "deliver [p] sample to [c] by sales staff [ss]". Softgoals are qualitative objectives for whose satisfaction there is no clear-cut criteria ("less disturbance" is rather a vague objective), and they can be contributed either positively or negatively by goals and tasks: "interact via visual dialogue and click-based interaction" contributes positively to "less disturbance", while "interact via voice dialogue and voice recognition" in some cases contributes negatively to it.

The goal model of Fig. 1 incorporates multiple alternatives the system can adopt to fulfil its main goal "promote product [p] to customer [c] in mall [m]". A goal model alternative is a root-to-leaf path in a goal model constructed by selecting one subnode in OR-Decompositions and Means-Ends, all subnodes in AND-Decompositions, and one dependency when multiple dependencies exist for the same node. The model explicitly represents the relation between the space of alternatives incorporated in it and context. Contexts, labelled by $C_1 \ldots C_{15}$ in Fig. 1 and explained in Table 2, can be specified at the following variation points:

1. **OR-Decomposition**. The adoptability of a subgoal (subtask) in an OR-Decomposition may require a specific context.
2. **Means-end**. Goals can be ultimately satisfied by means of specific executable processes (*tasks*). The adoptability of each task may require a specific context.
3. **Actors dependency**. A certain context may be required for an actor to establish a dependency on another actor for a goal to be achieved or a task to be executed.
4. **Root goal**. A root goal may be activated only in certain contexts.
5. **AND-Decomposition**. The satisfaction (execution) of a subgoal (subtask) in an AND-Decomposition could be required only in certain contexts. In other words, some subgoals (subtasks) are not always mandatory to fulfil the top-level goal (task).
6. **Contribution to softgoal**. Softgoals are qualitative objectives, i.e., there is no clear-cut criteria for their satisfaction. Softgoals can be contributed either positively or negatively by goals and tasks. The contributions to softgoals can also vary from one context to another.

Observation. Some contexts could precondition a goal or a task no matter where it appears in the goal model hierarchy. That is, certain contexts could inherently precondition the operability of a goal or a task. For example, let us take the task T = "print the map of the shopping mall" which belongs to the reception office system actor, say. For the operability of T, a context like C= "the receptionist's PC is connected to a printer" should hold no matter where this task appears. Let us call this an *inherent context*. However, some other contextual conditions apply on this same task only when it is part of a certain refinement and we call this a *refinement context*. If T is a means to satisfy a goal G_1= "familiarize emergency team with the mall", we should add to C the context C_a= "there

Work	Contribution
Weaving context with requirements [15,16]	- A theoretical ground for weaving together the variabilities of both context and requirements - Defining a set of variation points on goal models where context conditions can be specified - A basic formalization of the model and a set of basic analysis implemented in Datalog
Context analysis [17, 20]	- A systematic way to refine context and elicit its specification - A modelling language for context refinement - A systematic way to derive contextual workflows from contextual goal models
Automated analysis of contextual goal models [18,19]	- Derivation of the goal alternatives which fit a context monitored at runtime and a set of user preferences expressed over softgoals - Derivation of a set of alternatives with minimal costs and able to meet all goals - Checking the consistency of context specification - Checking the consistency of tasks and avoid conflicting actions - A first version of a CASE tool (RE-Context) which implements the above four reasoning mechanisms
This work	- Advocating the need to explore a deployment environment and customize the requirements model - Providing a systematic approach, based on contextual goal models, to explore a deployment environment - Proposing analysis mechanisms to customize a contextual goal model to fit the characteristics of an explored environment by removing uselessness and redundancy and suggesting augmentations - Extending our CASE tool (RE-Context) proposed in [18] to implement the proposed analysis

Table 1 Summary of our previous work and contribution of this paper

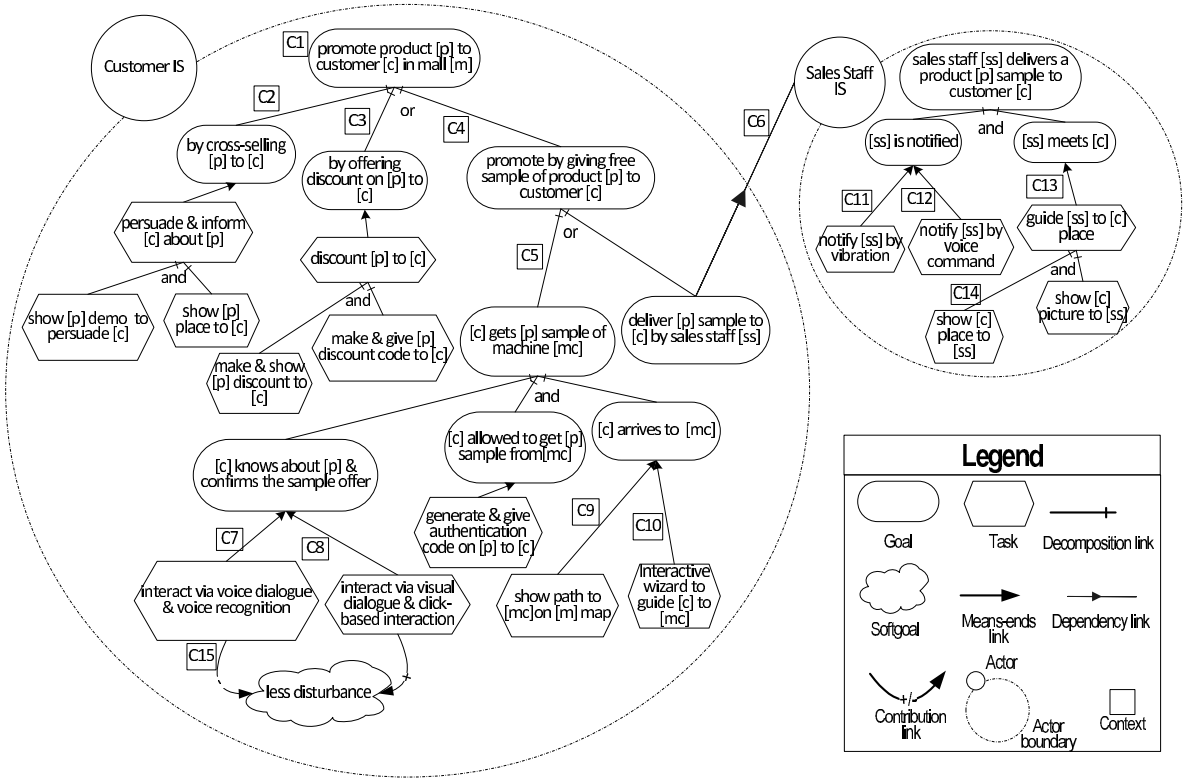

Fig. 1 A goal model annotated with contexts at its variation points

is an emergency situation and the emergency team is not familiar with the mall". If T is a means to achieve a goal G_2= "promote the mall" then we should add to C a context like C_b= "the customer is visiting the mall for the first time and he did not visit another branches with a similar structure". □

By attaching the context condition to the refinement and not to the task/gaol itself, we capture both cases: the drawback is that the annotation of the inherent contexts is repeated each time the corresponding tasks or goals appear in a refinement. On the other hand, associating context with a task or a goal itself regardless where it appears in the model leads to mistakes as the previous example shows. It was our design choice to tolerate this redundancy to improve simplicity and readability of the model. Alternatively, the analyst can specify inherent and refinement contexts separately. The analysis we propose in the rest of this paper apply to both ways as it processes the accumulated context of a whole goal model alternative rather than the individual contexts specified on it.

C_i	**Description**	**Variation point type**
C_1	the customer is in the product area for more than 5 minutes, has not got a previous promotion on it, and promotion is potentially realizable	Root goal
C_2	the product can be used with another product the customer already has	OR-Decomposition
C_3	the product is discountable and interesting to the customer	OR-Decomposition
C_4	the product is free sampled and the customer does not have the product and does not know about it	OR-Decomposition
C_5	the customer has experience with free samples machines and can reach one of the machines and start to use it in a short time	OR-Decomposition
C_6	a sales staff has the ability and time to explain sufficiently about the product to the customer	Actors dependency
C_7	the customer place is not noisy, and the system is trained enough on the customer voice	Means-end
C_8	the customer has a good level of expertise with regards to using technology and a good control on his fingers, and the used device has a touch screen	Means-end
C_9	there is a sample machine close to and in the same floor as the customer and he is familiar with digital maps technology	Means-end
C_{10}	the path between customer and the machine is complex and the customer knows how to interact and follow the guidance wizard	Means-end
C_{11}	the sales staff PDA vibration is activated and he is holding his PDA and not using it for a call	Means-end
C_{12}	the sales staff is putting his headphone on and he is not using his PDA for a call	Means-end
C_{13}	the customer is not moving quickly	Means-end
C_{14}	the customer stays around the sales staff and can be seen easily	AND-Decomposition
C_{15}	there are other people around the customer and the place is quiet	Contribution

Table 2 Description and variation point type for each context in Fig. 1

The context preconditions propagate in a top-down style. The context which preconditions a root goal G will also precondition all the nodes belonging to its subhierarchy when G is to be achieved. However, certain nodes in the subhierarchy might also belong to the hierarchies of other goals. Thus, the context preconditioning a node depends on the top nodes which were activated and selected. For example, consider a task T= "turn on the siren", which could be a means to fulfil two goals G_1= "protect mall from robbery" and G_2= "protect mall from fire". Each of these two goals is activated in different contexts: C_1= "a customer with unchecked item is leaving", and C_2= "smoke sensors indicate a fire", respectively. When task T is a means to satisfy G_2, it is also preconditioned by C_3 = "there is some staff or someone who can hear the siren and call the emergency". Thus, C_2 and C_3 are not always preconditions for T but only when it is meant to satisfy goal G_2. When T is part of the goal model alternative to fulfil G_1, context C_1 will be propagated to it, while C_2 and C_3 will not be considered.

2.2 Context analysis

Similarly to goals, contexts need to be analysed. On the one hand, goal analysis provides a systematic way to discover alternative sets of tasks an actor can execute to satisfy a goal. On the other hand, context analysis should provide a systematic way to discover alternative sets of facts an actor needs to verify to judge if a context applies. We specify context as a formula of world predicates. The EBNF of this formula is shown in Code 1. We classify world predicates, based on their observability by an actor, into two types: *facts* and *statements*:

Definition 1 (Fact) *A world predicate F is a fact for an actor A iff F is observable by A.*

Code 1 The EBNF of context world predicates formula

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

Definition 2 (Statement) *A world predicate S is a statement for an actor A iff S is not observable by A.*

An actor can observe a fact if it has the ability to capture the necessary data and compute the truth value of a fact. A statement cannot be observed by an actor for different reasons, such as (i) lack of information to verify it: (ii) its abstract nature 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 3 (Support) *A statement S is supported by a formula of world predicates φ iff φ provides enough evidence to the truth of S.*

The *support* relation is an approximation that analysts introduce to infer the validity of a statement from monitorable facts. In turn, this enables to verify high-level contexts (expressed in business terms) from monitorable facts (expressed in physical terms). In an iterative way, a statement is 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 Fig. 2, we analyse 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. The *and*, *or*, *implication* logical operators are represented as black triangles, white triangles, filled-in arrows, respectively.

Our context analysis technique helps the analyst to refine the high level descriptions of contexts in an iterative and systematic way to ultimately reach observable facts. In order to decide the observability of a world predicate, i.e., to decide whether it is a statement or a fact, the analyst looks first for a set of data the system can collect and judge upon the truth value of that world predicate. If such a set exists, then the world predicate is a fact. When reaching a fact, the analyst has also to depict explicitly those data required to verify it. For example, and taking the context analysis shown in Fig. 2, the truth values of the leaf facts can be determined based on the data conceptually modelled in Fig. 3. Each fact, however, concerns a fragment of the model and the figure shows the aggregation of those fragments in one model.

3 Requirements-Driven Deployment Framework

In this section, we discuss the customization of requirements, expressed via contextual goal models, to fit a system deployment environment. Such environment could exhibit certain characteristics that originate shortcomings, redundancies and uselessness in parts of the requirements model. Modifying the requirements model to fit its environment is an essential step for a holistic software deployment. While requirements customization could also consider decisions about different aspects, such as users preferences, costs, qualities, and organizational rules, we are here interested in modifying the model to suit the contextual variability profile of a deployment environment. For this reason, we propose to first sample the environment where the system is to be deployed (Section 3.1). Then, the collected samples are analyzed for two purposes: (i) to locate loci in the contextual goal model where augmentation with extra alternatives is required (Section 3.2) and (ii) to minimize development costs by removing alternatives which are inapplicable or redundant (Section 3.3).

3.1 Sampling a deployment environment

The context variations in a system environment can influence users requirements. Certain variations may activate certain requirements, or be required to adopt a specific alternative set of functionalities to satisfy the active requirements. For example, if a customer seems interested in a product (context) then a product promotion has to be established (requirement). Establishing the promotion can be done via different alternatives, such as giving a free sample, cross-selling, or discounting. The adoptability of each of these alternatives requires a certain context to hold. For example, promoting by giving a sample requires a context like “the product is new to the target customer“ to hold.

Sampling a deployment environment means monitoring what context variations, which influence the systems requirements, occur in that environment. The sampling process monitors the contexts that activate requirements. If one of these contexts holds, then the process also monitors the contexts required to adopt each of the alternatives that the model supports for fulfilling the activated requirement. The values of the monitored contexts will be stored as a *sample*. Sampling the system environment is an iterative activity which should be performed during a period of time at the deployment phase.

Contextual goal models can provide guidance and rationale for the environment sampling process as they explicitly capture the influence of context variations on requirements at the goal level. Our models represent the influence of context on goal activation, as well as on possible alternatives and their qualities. Accordingly, we have specialized context into three types, each associated with a different set of variation points in a contextual goal model:

1. **Activation context** makes it necessary to achieve (execute) a set of goals (tasks). In our contextual goal model, activation contexts reside at the variation points (i) *Root goals activation* and (ii) *AND-decomposition*. An activation context decides if a goal should be satisfied or a task should be executed. The activation context of a goal model alternative is the logical conjunction of the contexts at the variation points of these two types (see Fig. 4).
2. **Required context** is necessary to adopt a certain way for achieving (executing) a set of active goals (tasks). Required contexts are those associated to the contextual variation points (i) *OR-decomposition*, (ii) *Means-end*, and (iii) *Actors dependency*. These contexts are required to make a specific alternative of the goal model applicable. The required context of a goal model alternative is the logical conjunction of contexts at the variation points of these three types (see Fig. 4).
3. **Quality context** influences the quality of an alternative 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 measure (i.e., softgoal). In this paper, we do not address softgoals and quality contexts. We only focus on the operability of the system and we leave the quality dimension for future work.

Sampling the environment means monitoring the activation contexts of a contextual goal model alternatives until one context is true. Such event activates one or more alternatives in the goal model. whose required contexts are then monitored. At this point, we can face two situations:

– If there exists at least an alternative with a holding required context, then the sample is *positive* and the system has a way to satisfy the activated set of goals/tasks.

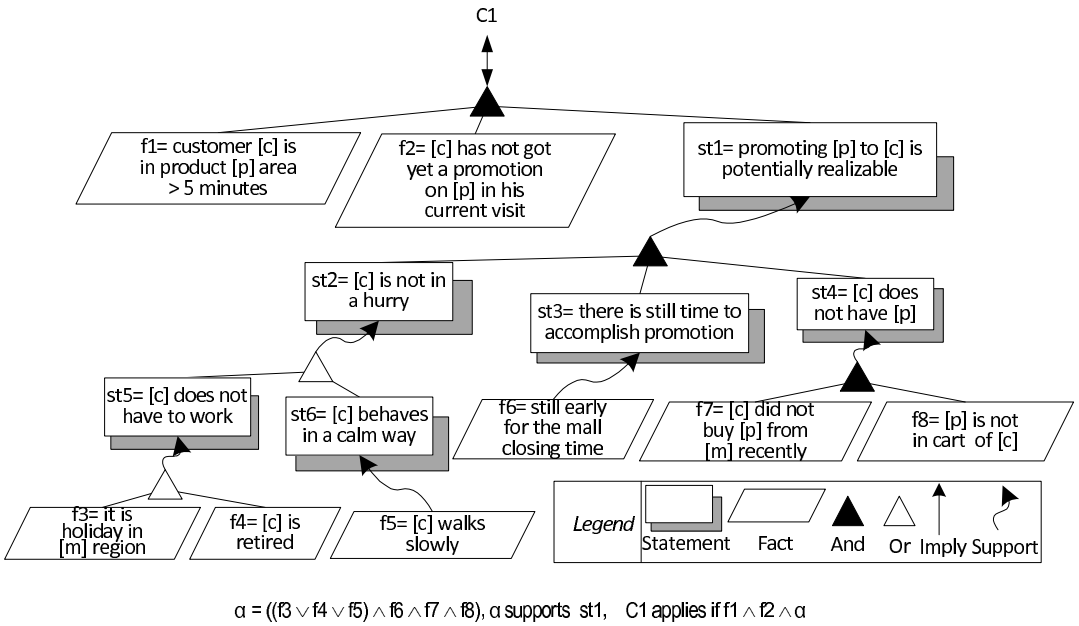

Fig. 2 Context analysis for C_1

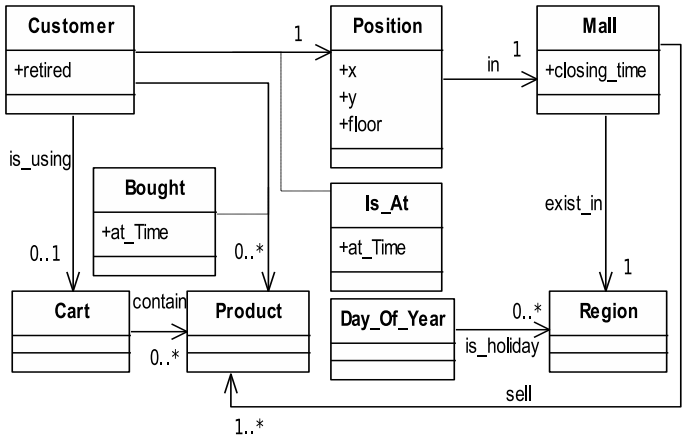

Fig. 3 The data needed to verify the facts of context C_1 shown in Fig. 2

– If there exists no such alternative, then there is no adoptable way to satisfy the activated goals/tasks and, thus, the sample is *negative*.

The sampling process stores the values of monitored contexts as environment samples. The sampling process will monitor the environment at deployment time and collect a set of environment samples to decide upon whether there is a need to augment the goal model with additional alternatives (Section. 3.2) and also if we need to clean it from redundant and useless ones (Section. 3.3).

3.2 Augmenting contextual goal models

A contextual goal model can be deployed in multiple environments. The space of alternatives that the goal model supports, each preconditioned by a specific context variation, could be insufficient for meeting users' goals in one environment for several reasons. The reason we address in this paper is the unsatisfiability of contexts that are required to adopt activated goal model alternatives. In other words, certain contexts of the deployment environment activate a set of goals and there is no alternative with a valid required context. Thus, some requirements are activated but unsatisfiable. We propose a solution to this problem based on augmenting the goal model alternatives space to enhance the operability of the system when contextualized alternatives are missing.

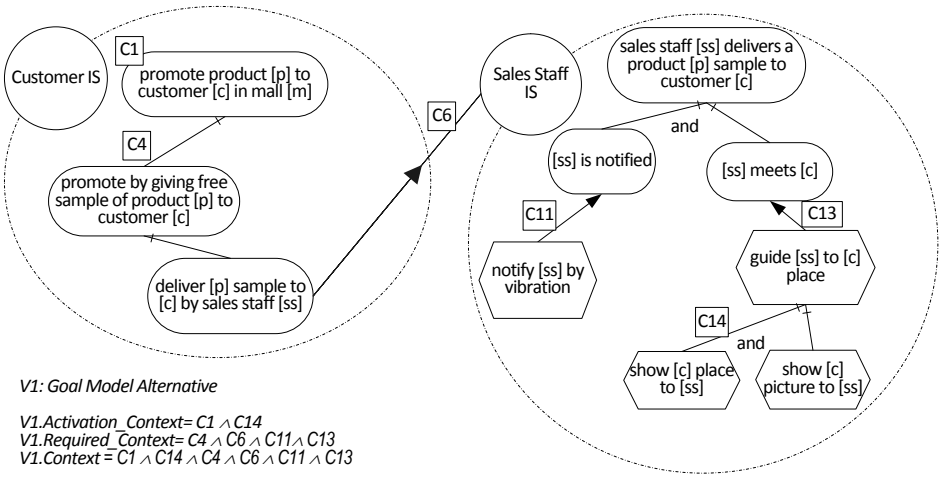

Fig. 4 An example of a goal model alternative and its accumulative contexts

We augment the goal model with default domain-independent solutions that can be adopted when no adoptable contextualized alternatives exist. Default solutions are backup options, and are typically not as good as contextualized ones. They are meant to allow operability of the system when it does not find an alternative that is expressly tailored to a current context.

Example 1 Consider the contextual goal model of Fig. 1. When C_1= "the customer is in the product area for more than 5 minutes, he has not already got a promotion on it, and the promotion is potentially realizable" is true, the root goal "promote product to customer in mall" is activated. If none of the contexts C_2= "the product can be used with another product the customer already has", C_3= "the product is discountable and interesting to the customer", and C_4= "the product is free sampled and the customer does not have the product and does not know about it" holds, then none of the alternatives to fulfil the activated root goal will be adoptable. Thus, we have an activated requirement and no alternative to fulfil it. When such a situation occurs, a default solution could be executed by sending an SMS to the customer introducing the basic features of the product.

The algorithm *Discover Global Solutions* shown in Fig. 5 takes as an input a contextual goal model and a set of environment samples obtained as described in Section 3.1. As an output, it returns the set of possible solutions (alternative augmentations sets) for each negative sample. To this end, the algorithm generates the set of goal model alternatives together with their contexts (Line 1). For each environment sample (Line 2), the algorithm checks if the system can find a way to satisfy the activated set of goals (Line 3). If so, the algorithm does not need to find any solution as the sample is positive (Lines 3-4). Otherwise (Line 5), the algorithm invokes another algorithm *Process Sample* (reported in Fig. 6 and described later) to specify loci in the goal model where an augmentation could be needed (Line 6). Then, the algorithm traverses the resulting annotated goal model to organize the partial augmentations suggested by the invoked algorithm *Process Sample* as sets of global solutions and associates these solutions with the processed sample (Line 7). The algorithm cleans the contextual goal model by deleting the annotated augmentations (Line 8), before processing the next sample.

Input: CGM: *a contextual goal model*
$Samples$: *Array [1..n] of environment samples*
Output: $Solutions$: *Array [1..n] of set of possible solutions*

1: $V := Generate\_Alternatives\_Set(CGM)$
2: **for** $i = 1$ *to* n **do**
3: **if** $\exists v \in V, Holds(v.Activation\_C, Samples[i]) \wedge Holds(v.Required\_C, Samples[i])$ **then**
4: $Solutions[i] := \emptyset$
5: **else**
6: $Process\_Sample(CGM.Root\_Goal, Samples[i], CGM)$
7: $Solutions[i] := Construct\_Global\_Solutions(CGM)$
8: $Delete\_Annotated\_Augmentations(CGM)$
9: **end if**
10: **end for**

Fig. 5 Algorithm *Discover Global Solutions*

The algorithm *Process Sample* is shown in Fig. 6. The algorithm is invoked for negative samples and its purpose is to identify loci where augmentations with default non-contextual alternatives are needed and to annotate the goal model accordingly. The algorithm starts processing the root goal node and traverses the model recursively. For each subnode of the node being processed (Line 1), the algorithm checks if another recursion is needed (Lines 2-4), i.e, if augmentations in the hierarchy rooted by that subnode could be needed. This happens when the context preconditioning this subnode is true in the processed negative sample (Line 2). If such a context is false then any augmentations in that subnode hierarchy will not be even satisfiable and are therefore useless. While traversing the model, the algorithm annotates a possible augmentation for each refinement of the types *OR-Decomposition, Means-end, Dependency* (Lines 5-6). The reason is that these types of refinements introduce alternatives and, therefore, a non-contextual alternative may be added there when contextualized alternatives are not adoptable.

Example 2 In Fig. 7, we show the results of processing the contextual goal model of Fig. 1 with respect to a negative

Input: N: *a contextual goal model node (goal/task)*
s: *an environment sample*
CGM: *contextual goal model*
Output: *CGM annotated by possible augmentations*

1: **for all** $\{N_i : IsSubnode(N_i, N)\}$ **do**
2: **if** $Holds(N_i.Context, s)$ **then**
3: $Process\_Sample(N_i, s, CGM)$
4: **end if**
5: **if** $N.Refinement\_Type \in \{OR, Means{-}end, Dependency\}$ **then**
6: $Annotate\_Augmentation(CGM, N.Refinement)$
7: **end if**
8: **end for**

Fig. 6 Algorithm *Process Sample*

environment sample. The activation context that holds here is $C_1 \wedge C_{14}$. In this sample, there is no alternative to satisfy the activated set of goals and tasks. In other words, there is no alternative with a true required context supported in the contextual goal model. Algorithm *Discover Global Solutions* will find this sample negative and then invoke the algorithm *Process Sample* that will traverse the model looking for loci where augmentations could convert the sample to a positive one. The algorithm *Process Sample* returns the goal model annotated with the augmentations $A_1 \ldots A_7$. Algorithm *Discover Global Solutions* traverses the goal model and organizes these partial augmentations as a set of global solutions $\{\{A_1, A_2\}, \{A_3, A_4\}, \{A_5\}, \{A_6\}, \{A_7\}\}$. The implementation of any of these sets will enable an alternative to fulfil goals when the sample under discussion occurs.

The algorithm *Discover Global Solutions* produces a set of possible solutions concerning each negative sample separately. Each solution includes one or more augmentations to the goal model (e.g., $\{A_1, A_2\}$ is a solution that includes 2 augmentations as shown in Fig. 7). However, a suggested augmentation is could not be implementable. Therefore, the analyst has to specify for each augmentation whether it is realizable and refine the model specifying how this can be done. This specification will decide which samples are convertible to positives and support the final decision about which augmentations to implement. The decision may rely upon, e.g., a cost-benefit analysis concerning the feasibility of the augmentations. We leave this topic for future work.

Example 3 Taking the augmentations suggested in Example 2 and illustrated in Fig. 7, the analyst could find A_4, A_5, A_6, and A_7 unrealizable by contrast to A_1, A_2, and A_3. The analyst has then to specify each realizable augmentation. The analyst may specify that A_1 can be realized via task *"inform via text message & get confirmation by OK button"*, A_2 via *"leaving a voice mail"*, and A_3 via *"sending SMS"*. All of these augmentations are less effective than the contextualized ones. For example, by *"inform via text & get confirmation by OK button"*, the system cannot interact with the customer and provide expressive information that leads to better management of the sample delivery process.

3.3 Reducing contextual goal models

The system environment may exhibit properties having static nature which help to customize the requirements model at the deployment stage. In one environment, some requirements are never activated or applicable and other requirements have always more than one alternative to satisfy. Contextual goal models may have unusable or redundant parts when the system operates in a specific environment. Thus, the deployment has to check a contextual goal model against its environment to detect whether it contains such parts and derive a reduced version of it that is expressly tailored to its deployment environment. Reducing contextual goal models helps to reduce the functionalities the deployed software has to support. Consequently, it reduces the time of the deployment process and avoids the costs of implementing functionalities that are not going to be used. We propose a process to reduce contextual goal models mainly by detecting and removing unusability (Section 3.3.1) and redundancy (Section 3.3.2).

3.3.1 Processing unusability The contexts specified at a contextual goal model may have static values in a certain environment. The environment samples may show that some of these contexts are always true or always false. Each case has different effect and requires a different reduction action.

The contexts having true truth values in all of the collected samples lead to useless monitoring functionalities and, thus, unjustified costs. Context monitoring requires collecting environment data which requires deploying equipments (databases, sensors, positioning systems, cameras, etc.) and processing collected data. If a context is always true, we do not need to deploy its monitoring functionalities. To detect and remove this type of contexts, we need to traverse the goal model and check whether each context (specified at each variation point) is always true in all the environment samples.

Example 4 If the shopping mall, where the system is to be deployed, consists of one floor, has plenty of sample distribution machines and it is for products usable by young people who are usually familiar with new technology, then C_9= *"there is a sample machine close to and in the same floor as the customer and he is familiar with digital maps technology"* would be always true and there is no need to monitor it. Such context has to be removed from the goal model deployed in that specific mall. Removing this context avoids installing a positioning system, preparing a map for the mall, and developing a database to store customer profiles (presuming that other system functionalities do not require that).

On the other hand, the contexts having false truth values in all collected samples lead to unjustified monitoring functionalities. Moreover, the existence of such contexts means that some goal model alternatives are never applicable. Both the contexts specified at the variation points and the accumulative contexts of goal model alternatives could have false values. As we mentioned earlier, the context of a goal model alternative is the conjunction of two accumulative contexts:

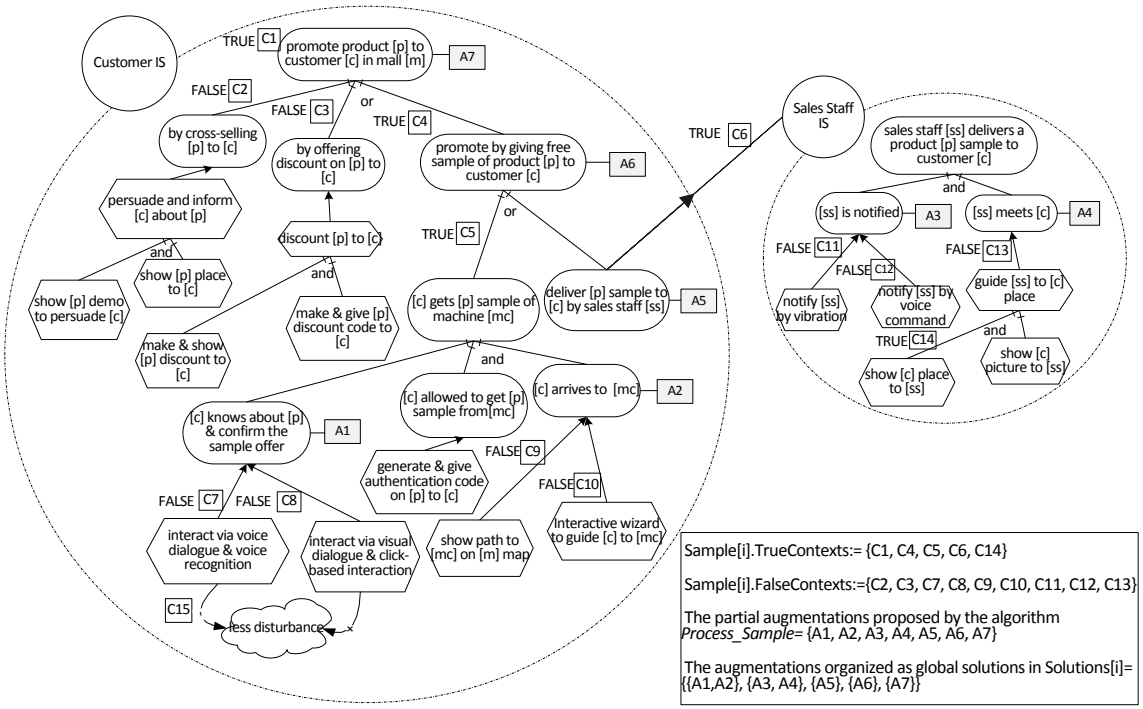

Fig. 7 An example of augmentations found by sample processing, and alternative solutions resulted by organizing augmentations

the activation context and the required context of that alternative (see Fig. 4). The alternatives including contexts (individual or accumulative) that are false in all the environment samples should be removed from the model. This is to avoid deploying functionalities meant to support inapplicable goal model alternatives. By removing these alternatives, we potentially reduce the costs of monitoring the contexts included in them. However, the removal of one alternative does not mean that the goals, tasks, and contexts included in it will be removed definitely from the final model, as they might also appear in other applicable alternatives.

The algorithm reported in Fig. 8 processes a contextual goal model against a set of environment samples and reduces it by removing alternatives with always-false contexts. The algorithm generates all the contextual goal model alternatives together with their contexts (Line 1). An alternative is adoptable with regards to one environment sample if both its activation and required contexts hold in that sample. The algorithm then checks all the contextual goal model alternatives (Line 3–7). If an alternative is adoptable in at least one of the environment samples (Line 4) then it is added to the set of the adoptable alternatives V' (Line 5). Finally, the algorithm gives as an output the set of adoptable alternatives as a reduced version of the contextual goal model (Line 8).

Example 5 In a noisy shopping mall, the context C_7= *"the customer's place is not noisy, and the system is trained enough on the customer voice"* will never hold and any alternative preconditioned by a context includes C_7 will be inapplicable. Such alternative should be removed from the final set of adoptable alternatives. Excluding alternatives means excluding some functionalities (tasks) from the deployed system. The task *"voice dialogue and voice recognition"* can be excluded from the requirements model, as it will never be adoptable. The exclusion of alternatives leads also to excluding contexts to minimize monitoring costs. Removing C_7 avoids us the costs of sensing and analyzing the noise level in the location of customer and tracking the system learning of the user voice.

Input: CGM: *a contextual goal model*
S: *a set of environment samples*
Output: CGM': *CGM cleaned from useless alternatives*

1: $V := Generate\_Alternatives\_Set(CGM)$
2: $V' := \emptyset$
3: **for all** $v \in V$ **do**
4: **if** $\exists s \in S, Holds(v.Context, s)$ **then**
5: $V' := V' \cup \{v\}$
6: **end if**
7: **end for**
8: $CGM' := \bigcup\{v \in V'\}$

Fig. 8 Algorithm *Process Unusability*

3.3.2 Processing redundancy Some functionalities might be redundant when the system operates in a certain environment. An early and main source for redundancy is the requirements model. The environment may make some requirements redundant and therefore the functionalities that are deployed to satisfy these requirements will be redundant as well. Accommodating a large space of functional alternatives, even though redundant, could be desirable for reasons such as supporting high flexibility, different users' preferences, and fault tolerance. However, for reasons such as costs and time constraints, detecting the functionalities that allow the system to fulfil its requirements without redundancy is often needed.

Contextual goal models represent explicitly the relation between two spaces: the space of requirements alternatives and the space of context variations. Analysing contextual goal models to discover which requirements alternatives are redundant leads to a model that incorporates the minimum but sufficient space of alternatives. We consider an alternative of the goal model v redundant if there is another alternative v' that is applicable whenever v is applicable. In other words, we deal with redundancy originated by the replaceability of some alternatives. As we mentioned earlier, supporting redundant alternatives in the deployed system could be justified to develop high-variability systems. We leave the selection between redundant alternatives for future work.

The processing of a contextual goal model to remove redundant alternatives and group the rest based on applicability equivalence is reported in Fig. 9. The algorithm takes as input a contextual goal model and a set of environmental samples and returns the redundant and equivalent groups of alternatives. First, the algorithm generates the set of goal model alternatives together with their contexts (Line 1) and initializes the set $Equivalent\_Gv$ to an empty set. This set will be used later to represent the sets of equivalent alternatives (Line 2). The algorithm then organizes the adoptable goal model alternatives in groups based on equivalence of applicability in the environment samples (Line 3–11). To this end, the algorithm picks randomly an alternative v (Line 4) and finds the group of alternatives $Gr.alternatives$ that are applicable exactly when v is applicable (Line 5). The samples set in which these alternatives are applicable is also recorded (Line 6). The group of equivalent alternatives is eliminated from the set of all alternatives (Line 7) and considered for further processing if there exists at least one sample in which v (and therefore the other alternatives in the group) holds[1]. If a group of equivalent alternatives is replaceable by another, i.e., if in all samples where the first is applicable so the second is, then the first group is marked replaceable and removed from the output (Line 11 – 12).

Input: CGM: *contextual goal model*
S: *a set of environment samples*
Output: CGM': *CGM without redundancy*

1: $V := Generate\_Alternatives\_Set(CGM)$
2: $Equivalent\_Gv := \emptyset$
3: **while** $V <> \emptyset$ **do**
4: $v := Pick\_alternative(V)$
5: $Gr.alternatives := \{v' \in V, \forall s \in S, Holds(v.context, s) \leftrightarrow Holds(v'.context, s)\}$
6: $Gr.samples := \{s \in S, Holds(v.context, s)\}$
7: $V := V \setminus Gr.alternatives$
8: **if** $Gr.samples <> \emptyset$ **then**
9: $Add\_element(Gr.alternatives, Equivalent\_Gv)$
10: **end if**
11: **end while**
12: $Replaceable\_Gv := \{G \in Equivalent\_Gv, \exists\ G' \in Equivalent\_Gv \text{ and } G.samples \subset G'.samples\}$
13: $CGM' := \bigcup\{v \in (Equivalent\_Gv \setminus Replaceable\_Gv)\}$

Fig. 9 Algorithm *Process Redundancy*

[1] This check is unneeded if unusability is already processed

Example 6 Fig. 10 shows an example of a replaceable alternative. In this example, we suppose that the assistance staff members have to put the headphone on when they are available for serving customers. If such a policy holds in the shopping mall where the system is to be deployed, then the implication $C_{11} \rightarrow C_{12}$, where C_{11} = *"the sales staff PDA vibration is activated and he is holding his PDA and not using it for a call"* and C_{12} = *"the sales staff is putting his headphone on and is not using his PDA for a call"*, holds in all of the environment samples collected from that mall. This means that V_2 is applicable in all samples where V_1 is applicable. Therefore, V_1 can be considered replaceable and can be removed from the model. Consequently, the task *"notify sales staff by vibration"* will not be deployed and the context C_{11} can be also removed from the set of contexts to monitor.

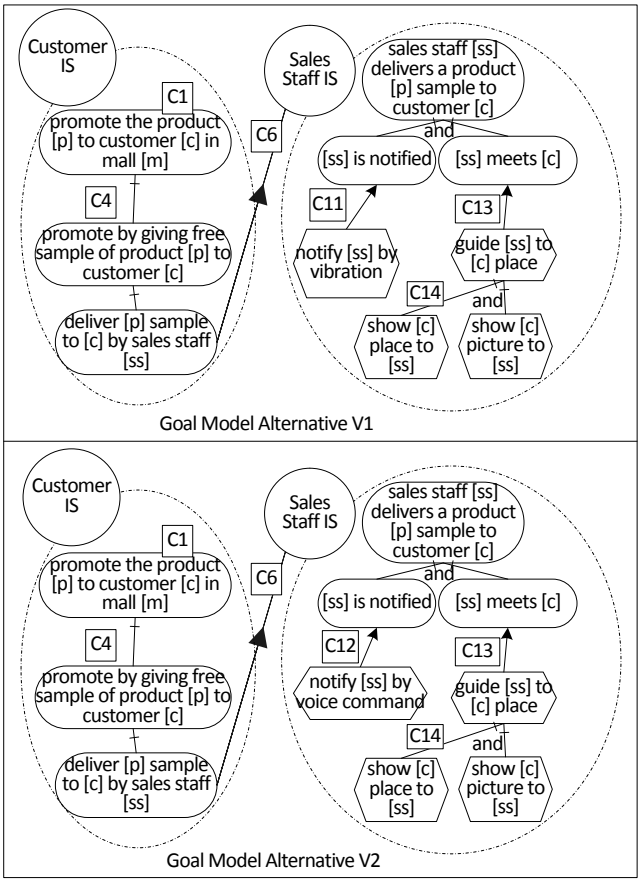

Fig. 10 V_1 is replaceable by V_2 in a mall where $C_{11} \rightarrow C_{12}$ in all samples

3.4 CASE tool: RE-context

We have developed a prototype automated tool (called RE-Context) to support the customization of a contextual goal model for a host environment. In our previous work [18, 19], we have developed earlier versions of this tool for different

kinds of analysis concerning the generation and processing of a contextual goal model alternatives and the verification of consistency and freedom of conflicts in a contextual goal models. In this paper, we implement a new component to support the activities of deploying a contextual goal model in a host environment: augmentation, unusability, and redundancy processing. The architecture of RE-Context is shown in Fig. 11 where we highlight the "deployment" component which is the one implementing the analysis proposed in this paper.

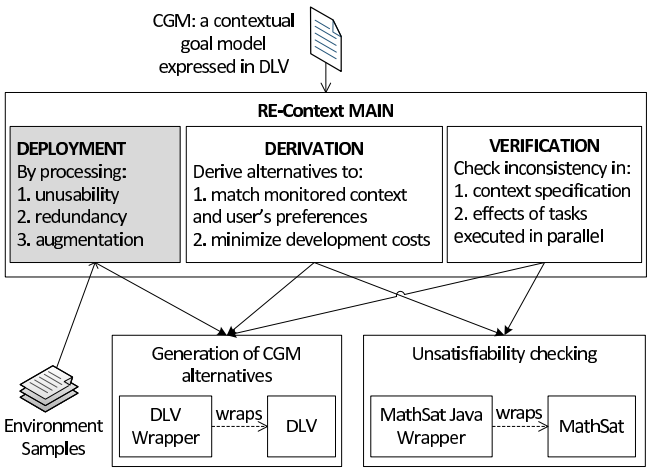

Fig. 11 The architecture of RE-Context CASE tool

As shown in Fig. 11, RE-Context is composed of three logical components. These components are developed in Java and are linked to external reasoners via wrappers. The deployment component is the component specifically developed for this paper. The derivation of the space of alternatives for a contextual goal model is a basic step for all types of analysis of RE-Context. RE-Context can derive alternatives included in a contextual goal model by running the DLV reasoner as a planner. RE-Context outputs all the valid models that satisfy the Datalog inference rules in the input file encoding the contextual goal model. Each alternative consists of a set of tasks to execute and a set of contexts associated with it. For more details about the generation of alternatives please see our work in [18]. In the following, we describe the main components of RE-Context:

– **Derivation**. RE-Context generates the space of alternatives included in a contextual goal model. Re-Context enables the choosing between alternatives in two styles. The first is a runtime derivation of the alternatives matching a given context and user's preferences expressed as ranking over softgoals. The second concerns the derivation of a set of tasks with minimal development costs which, if implemented, allows the system to cover and operate in the space of context variations captured in a contextual goal model. To this end, the analyst has to specify the resources needed for the development of each task in a goal model and their costs. Please see our work [18] for more details about this component.
– **Verification**. RE-Context generates the space of contextual goal model alternatives and checks context satisfiability for each alternative. This checking is done by relying on the state-of-the-art SMT solver MathSAT[2]. The alternatives with unsatisfiable contexts are excluded from the model since they are inapplicable. Furthermore, the tool checks the generated alternatives to detect those including tasks executing in parallel and leading to conflicts. RE-Context expects the analyst to specify the effect of each task on the environment, the parallel and sequence operators on a goal model [21], and the logical relations between contexts. More details on this component can be found in [19].
– **Deployment**. This component implements the analysis we have proposed in this paper. It takes as input the DLV input file which encode the contextual goal model, and the CSV file encoding the samples collected from a host environment. Both files are provided by the analyst. The DLV should encode an already verified contextual goal model (using the verification component) to avoid having unnecessary processing and incorrect results. For example, if we check redundancy before verification, the tool might return that an alternative a_1 is redundant given that there is always an alternative a_2 which is applicable whenever a_1 is applicable. However, the verification might show that a_2 has a conflict and, therefore, we cannot consider a_1 redundant, since a_2 is not functionally correct and cannot replace a_1. The process shown in Fig. 12 and discussed in Section 4 explains the way of using the automated analyses. The deployment component performs three types of analysis which are all implemented in Java:
  – **Unusability**. This analysis implements the algorithm described in Fig. 8, where the samples file is scanned to identify contexts which are always true or always false. Based on that, the analysis reduces the set of contextual goal model alternatives (as described in Section 3.3.1), excluding the alternatives which are inapplicable and replacing the contexts which were always true or false with their static value.
  – **Redundancy**. This analysis discards redundant alternatives, i.e., those operating in an environment where other alternatives can operate and achieve the same goals. The analysis enacts the algorithm reported in Fig. 9 to identify and remove redundant alternatives. This analysis becomes unnecessary if the designer decides to keep redundant alternatives for reasons such as flexibility and accommodating the preferences of different users.
  – **Augmentation**. The purpose of this analysis is to find loci in a contextual goal model where an augmentation with a default non-contextual alternative is necessary to deal with negative samples. This processing enacts the algorithm reported in Fig. 5. It returns a set

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

of possible global solutions after processing individual alternatives. We still did not provide automated analysis to qualify and evaluate solutions. The decision about this is still left for the analyst and we plan to provide automated support for that in our future work.

4 Systematic Process

In this section, we provide guidance for modelling contextual requirements by means of contextual goal models and customizing it to a certain host environment. The process is depicted in the activity diagram of Fig. 12. A number of macro-activities are identified: goal analysis, context analysis, verification, environment sampling, augmenting alternatives space, processing unusability, processing redundancy, and composing the final contextual goal model.

1. **Goal analysis**. Actors and high level goals are identified and analysed. Actors and goals can be iteratively discovered through the set of scenarios which describe the problem domain [22]. Moreover, an intentional variability taxonomy [23] can guide variability acquisition when refining a goal/task to discover alternative ways of fulfilling/executing it. Each refinement step is followed by a context analysis.
2. **Context analysis**. This activity is meant to link between the requirements and the context in which they are activated and adoptable. The context analysis activity is composed of two steps:
   (a) *Contextual variation points identification*. Each variation point in a goal model can be contextual. Context may affect either goal activation or the selection of alternatives to meet a goal. When a variation point is identified to be contextual, a high level description of the correspondent context has to be written down. As a result of this activity, the contextual variation points at the goal model should be annotated similarly to the model we have shown in Fig. 1. Moreover, the contexts associated with these variation points should be described similarly to the descriptions shown in Table 2.
   (b) *Context refinement*. The contexts at each contextual variation point should be analysed. The analysis goal is to identify ways through which the system can verify if a context holds. In other words, the context refinement has to define the environmental *facts* the system has to capture and the way these facts are composed to judge if an analysed context holds. An example of context refinement is shown in Fig. 2.
3. **Verification**. The requirements model has to be verified to ensure freedom of modelling errors which lead to inconsistencies in the context specification and conflicts between the executable tasks of a goal model. RE-Context supports this automated reasoning in our work described in [19]. Some alternatives of the goal model may be preconditioned by a set of contexts which never hold together. This make the alternative inapplicable and useless if implemented. Moreover, the tasks of a goal model could lead to contradicting changes on the system environment. This leads to conflicted actions that prevent a correct satisfaction of requirements. The verification of contextual goal models should be done as a preliminary step for the analysis that concerns the deployment.
4. **Environment sampling**. The deployment environment has to be explored through sampling. The samples are used to process the requirements model and ensure fitness between the model and the system environment. To do the sampling, the contexts specified at the contextual goal model need to be monitored iteratively for a period of time and their values should be recorded as samples. More specifically, the contexts that activate goal model alternatives should be monitored until one of them holds. When this happens, the sampling process records the values of the contexts specified at all of the variation points of the activated goal model alternatives. This will allow for further analysis to judge if the model supports or lacks ways to satisfy its activated requirements. Monitoring the environment could be established by different techniques such as surveying prospective users, being physically there and recording context variations, recording via cameras and analysing observations, etc. The sampling period has to be long enough in order to capture contexts variations that happen less frequently or periodically. However, the analyst would need to compromise between the coverage and quality of the sample set and the sampling period as we discuss in Section 5.2.
5. **Augmenting alternatives space**. In this activity, the contextual goal model has to be analysed using the collected environment samples to identify places where default non-contextual solutions are needed. To this end, RE-Context can be used to analyse the model with regards to each sample and record the results. Doing that, the tool will ask the analyst about the realizability of each suggested augmentation. The analyst can also assign costs to each augmentation and then use the tool to answer queries like calculating the minimum-cost solution that converts the largest number of negative samples to positive. The tool will record the augmentations approved by the analyst to be added later to the final goal model. These default solutions may be recorded to be suggested when deploying the contextual goal model to other environments that shares similar contextual profile.
6. **Processing unusability**. The goal of this activity is to remove unusable functionalities from the model. This activity is fully automated and executable in our automated RE-Context tool. The tool will scan the environment samples to elicit the set of contexts specified at the variation points of the goal model and having always-true values in all the samples. It will also check the accumulative context of each goal model alternative to see if it holds in at least one sample. After doing these two checks, the tool will remove the always-true contexts from the model

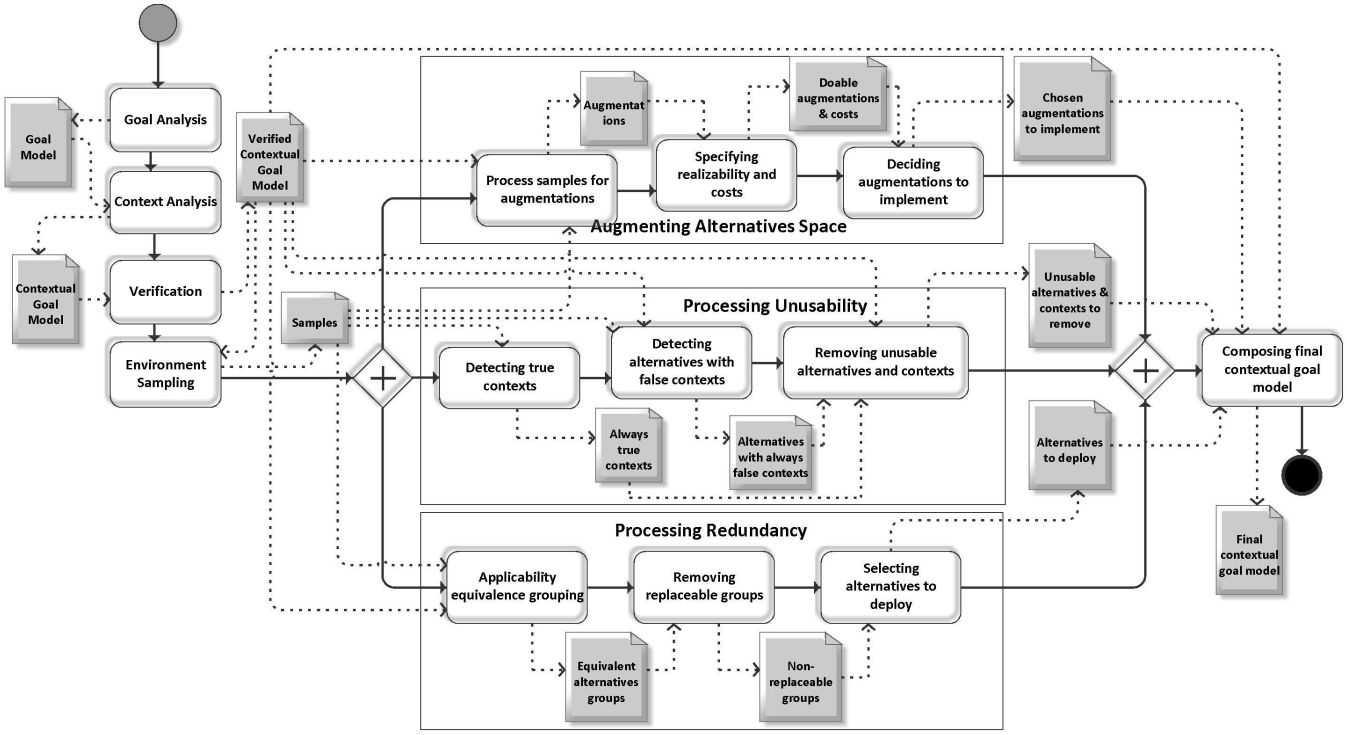

Fig. 12 The deployment process

since deploying the functionalities to monitor them is unjustified. Moreover, the tool will remove the goal model alternatives having always-false contexts to avoid deploying unusable functionalities. As we mentioned earlier, the removal of one alternative does not necessarily mean that all the goals, tasks and contexts included in it are removable as, indeed, they may appear in other alternatives that are applicable.

7. **Processing redundancy**. The purpose of this activity is to discover the set of goal model alternative which are replaceable when operating in a certain host environment. To this end, RE-Context can be used to group the alternatives based on the equivalence of their applicability in all the environment samples. Obviously, the group of alternatives that are inapplicable in any of the deployment environment samples are not further processed as we described in the last activity (processing unusability). The tool will remove any group of alternatives for which there is another group applicable in all the samples the first one is applicable in. Such group will be redundant and inessential for the availability of the system. The analysts can then use the tool to select an alternative of each of the remaining groups according to a certain selection criteria. The criteria that is currently supported by RE-Context is the costs of resources needed for each task of the goal model. The tool is able to select a set of tasks of the goal model able to implement at least one alternative of the irreplaceable groups of variants with a minimum total cos. The selection algorithm is explained in our previous work in [18].
8. **Composing final contextual goal model**. In this activity, RE-Context can be used to merge the augmentations suggested by the activity of augmenting the alternatives space with the goal model alternatives that are usable and not redundant, i.e., the alternatives delivered by the activities of processing unusability and processing redundancy. This goal model will be the final outcome of our deployment process. It is a version of the original contextual goal model tailored to context variations happening in the system operational environment. This version removes the useless and redundant parts of the original model and adds default alternatives to be adopted when a contextual alternative one is missing. Thus, it leads to a version of software with maximized operability and reduced development costs with regards to a certain deployment environment. The tool delivers the final goal model as a DLV code. We need to extend our tool to draw a visual notation based on such code.

5 Evaluation

To evaluate our proposed approach, we have developed a contextual goal model for a room booking system used by academic research groups. The purpose of the system is to assist researchers in the room booking process. Room booking can be performed for different purposes: giving seminars, holding brainstorming sessions, making conference calls, or having a project meeting. Different rooms are available and differ in their equipment, availability status and times, etc. The three main requirements of the room booking system are to

assist researchers in booking a room, to announce the meeting/event, and to assist researchers to get and use of supplementary equipments needed to establish the meeting/event. To collect the environment samples, we monitored the room booking processes for a period of time of two research groups (RG_1 and RG_2) at the Department of Information Engineering and Computer Science (DISI)[3] at the University of Trento. Then we have used the samples to customize the contextual goal model as we will explain later. Here we give more details about the three main requirements of the room booking system and for more details, please see:
http://disi.unitn.it/∼ali/index_files/RoomBooking.pdf:

- *Booking a room*. This activity can be in charge of the department secretary or the research group secretary when the later is available and assigned to the task of rooms booking. The secretary has to be informed by the system and, then, she has to check if there is a room available and reply to the staff who requested to book a room. The communication with secretaries follows different alternatives in different contexts. Booking a room can be done by the staff who requested a booking himself if he has the permission to use the room booking calendar (we used a free calendar web service).
- *Announcing a seminar*. The meeting requester needs assistance to announce the meeting/event either to public or specific set of people. Public announcements can be done via the website of the research group of the requester, the website of the department, or the reception screens visible to people who enters the department. Private announcements can be done via designated mailing lists (all members of the research group) or via chosen contacts from the contact book of the research group.
- *Assistance about supplementary equipment*. The system has to inform a room booking requester if there is a need to get supplementary equipment such as projector, connection cables, remote control, and keys before starting the seminar. In case supplementary equipment is needed, the system has to provide the requester with information on how to get and use that equipment. For example, the requester has to be assisted on how to get the projector and use it, the place to get the keys from and the procedure to follow in order to do that, etc.

Our monitoring process lasted for 3 months and the samples set consists of 61 booking processes: 34 reservations were made by RG_1, and 27 by RG_2. To collect samples, we asked research groups members and department personnel involved in the booking process to fill in a form for the contexts values regarding each room booking process. Thus, and since we have not implemented an automated system for the monitoring of context, we did not do the context refinement as the main purpose of such analysis is to specify contexts as formulae of facts monitorable by an automated system. The form included questions to determine the truth values of the activation and required contexts in our developed contextual goal model for rooms booking. Group members were asked to specify the truth values of the dynamic contexts, i.e., whose which vary from one reservation to another. Some other contexts were static and their values were known by us in advance (e.g., the profile of each research group, the rooms each group is granted access to, and installed equipments). After the samples collection phase, we used our automated support tool RE-Context to process the collected samples according to the analysis proposed in Section 3 and obtained the results reported and discussed later in this section.

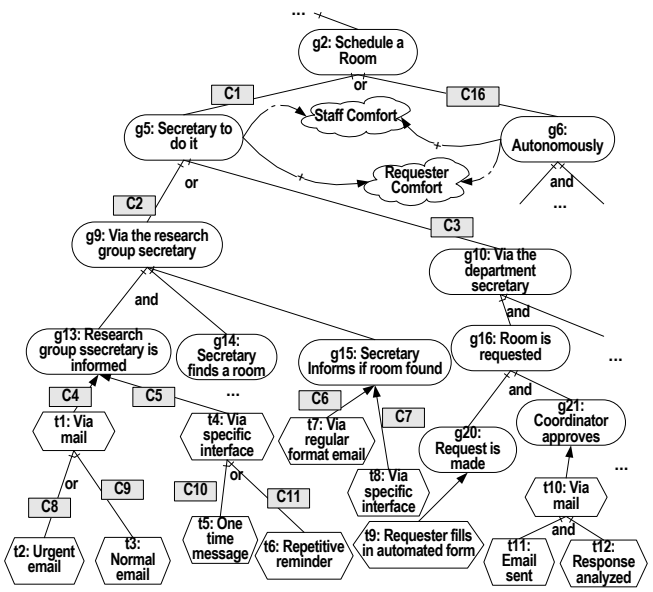

Fig. 13 Part of the contextual goal model for room booking

We now provide details concerning the contextual goal model we have developed. Part of this model is shown in Fig. 13. The main requirement of a researcher is to book a room for a certain scheduled activity. The reservation can be done by following various alternatives: the researcher can do it autonomously or can interact either with the research group secretary or with the department secretary. Each alternative is adoptable in a different set of contexts (e.g., C_3 states that the department secretary can be involved only if he is in charge of that specific group and if he is currently not occupied in higher-priority tasks) and the system is supposed to support and track the room booking accomplishment. For instance, the research group secretary can be notified either by sending an e-mail to him (task t_1) or via a specific web interface (task t_4), depending on the context. Another requirement for the system is to support the announcement of both public meetings and close meetings. Such requirement has to be supported via different alternatives: announcing via the web site of the department or the research group, circulating via mailing lists, displaying on electronic screens in the department, etc. Each alternative requires a specific context to hold. The last requirement of the room booking system is to remind meeting requesters about the supplementary equipment he would need such as projectors, web cams, network cables,

[3] http://disi.unitn.it/

and also to explain to a meeting requester how to get needed equipments and to use them. The contextual goal model we developed for the room booking system consisted of 5 actors, 33 goals, 52 tasks, 6 softgoals, and 34 contextual variation points. The space of alternatives to fulfil the main goal "having a room booked" included 2808 alternatives.

5.1 Samples processing results

In Fig. 14, we summarize the results obtained by applying the analysis described in Section 3 on the contextual goal model for the room booking system. In the remainder of this section we explain these results.

- **Samples positivity/negativity**. Sampling the room booking process for the first research group RG_1 led to 26 positive samples—where the top-level goals have been achieved—out of 34 samples, while the sampling for RG_2 led to 22 positive samples out of 27 samples. The main reason for failure in booking a room in RG_1 was that rooms are asked for booking short time before the event and this led to failure in completing the booking process on time. For example, the room requester did not have time to answer the emails concerning the supplementary equipments needed or to confirm the reservation after being made by the secretary. The main reason for failures in RG_2 was that the group did not have a secretary that assists in booking rooms. This group had to rely on the department secretary who often had other higher priority tasks to do.
- **Unusability processing**. This activity detects and removes static contexts that never vary and, also, inapplicable goal model alternatives. The developed system does not need to monitor static contexts (those having the same truth value in all collected samples). For example, the rooms assigned to RG_2 had all supplementary equipments and there was no need to monitor that. For RG_1, 3 contexts had false values in all the collected samples and 3 had true values. For RG_2, 3 contexts had false values while 2 contexts had true values in all the collected samples. The contexts that are always false in all the samples produced a large number of unadoptable alternatives (1512 for RG_1 and 2592 for RG_2). Consequently, the tasks that appear only in unadoptable alternatives are never executed (6 tasks for RG_1 and 20 tasks for RG_2). Also, it is useless to monitor the non-static contexts which appear in only unadoptable alternatives (3 contexts for RG_1 and 12 contexts for RG_2). For example, having a set of well-equipped rooms for RG_2 means the removal of all the goal model alternatives where researchers are reminded to bring supplementary equipments. This, in turn, led to unadoptability of the tasks and uselessness of monitoring the contexts needed to accomplish the assistance of the researcher to bring such equipments.
- **Redundancy processing**. A goal model alternative V is redundant if in every sample where V is applicable there exists another alternative V' which is applicable as well. As explained in Section 3, to discover redundant alternatives, we first remove inapplicable alternatives which have always-false contexts in all samples, i.e., unusable alternatives. Then, we partition the rest into groups on the basis of context value equivalence in all the samples, i.e., on the basis of applicability equivalence. The number of equivalent groups was 864 for RG_1 and 144 for RG_2. A group is redundant if its set of positive samples is included in the set of positive samples of another group. We got 792 redundant groups for RG_1 (containing 1152 alternatives) and 126 groups for RG_2 (containing 180 alternatives). The importance of this computation was the discovery of tasks that appear only in redundant alternatives (18 for RG_1 and 5 for RG_2). These tasks are redundant and, therefore, their deployment is optional. For example, our requirements model has two alternatives to notify a meeting requester: notification via email to be adopted when the person is often in office or has access to Internet, and notification via mobile phone (such as voice mail or SMS) that is adoptable when the staff is out of the department and has no access to Internet. In both research groups, researchers had always access to Internet even when researchers were out of the department. Thus, the option of communication via mobile phone becomes redundant and the system can exclude the deployment of tasks needed to establish it.
- **Augmentation processing**. Augmenting the requirements model with default non-contextual alternatives is the solution we proposed when the model lacks applicable contextual alternatives to meet goals, i.e., to convert negative samples to positive. However, it is not always possible to find a realizable default augmentation. For example, some of the collected samples of RG_2 were negative due to the fact that the group secretary and the department secretary were busy or having a day off. Finding a default non-contextual human-supported alternative was not possible and the sample could not be converted to positive. RE-Context performed the analysis we explained in Section 3, which locates a set of augmentations to the goal model and organizes these individual augmentations as alternative solutions in the form of sum-of-products formula. Then, analysts need to specify which augmentations are realizable. On the basis of this specification, a number of solutions will be marked realizable and, consequently, some negative samples will be convertible to positive. The total number of augmentations proposed to convert the negative samples to positive was 6 for RG_1 and 4 for RG_2. Out of these augmentations, the number of realizable ones was 2 in RG_1 and 2 in RG_2. This enabled us to convert 5 of the 8 negative samples in RG_1 to positive (3 of 5 in RG_2). For example, one sample of RG_1 was negative due to the fact that one overseas staff could not speak a language in common with the technician to understand the use of video-conferencing. The default solution was to develop a demo in multiple languages.

The Size of the Original Goal Model	Value	
Number of actors	5	
Number of goals	33	
Number of tasks	52	
Number of softgoals	6	
Number of contexts at variation points	34	
Number of goal model alternatives	2808	
Sampling	**Research Group 1**	**Research Group 2**
Time to collect the samples	3 Months	3 Months
Number of samples	34	27
Number of positive samples	26	22
Number of negative samples	8	5
Unusability processing		
Number of contexts true in all samples	3	3
Number of contexts false in all samples	3	2
Number of alternatives with always false contexts	1512	2592
Number of tasks that are never executed	6	20
Number of contexts removed with useless alternatives	3	12
Redundancy processing		
Number of equivalent groups of alternatives	864	144
Number of redundant groups of alternatives (RGV)	792	126
Number of alternatives included in RGV	1152	180
Number of tasks that appear only in RGV	18	5
Number of contexts that appear only in RGV	11	4
Augmentation processing		
Number of augmentations suggested for all negative samples	6	4
Number of realizable augmentations	2	2
Number of samples convertible from negative to positive	5	3
Size of the resulting goal model		
Number of actors	4	4
Number of goals	25	22
Number of tasks	28	27
Number of softgoals	6	6
Number of contexts at variation points	14	13
Number of goal model alternatives	144	36

Fig. 14 The results of analysing the room booking contextual goal model for two research groups

5.2 Discussion of the obtained results

The evaluation of our approach for customizing requirements to a deployment environment showed promising results (as reported in Fig. 15). A considerable percentage of contexts (58.8% for RG_1 and 61.8% for RG_2) were shown removable. Removable contexts include those having static—either true or false—value in all the collected samples, those removed because appearing only in useless alternatives, and those removed when redundant alternatives are dropped. Removing static contexts results in lower monitoring costs, as their values are known a priori. Moreover, a considerable percentage of tasks were shown removable (46.2% for RG_1 and 48.1% for RG_2). The tasks appearing only in alternatives that are useless, i.e., their contexts are false in all of the collected samples, should not be supported in the deployed system as they are not executable. The tasks that appear only in alternatives that are redundant are optional and thus can be removed to reduce costs. Moreover, our analysis has shown also a considerable quantity of negative samples where the contextual goal model does not include alternatives tailored to support certain contexts in its deployment environment (23.5% in RG_1 and 18.5% in RG_2). The analysis also suggested places in the requirements model where augmentations can convert the negative samples to positive. Some augmentations were realizable and this led to convert 62.5% of RG_1 and 60% of RG_2 negative samples to positive. Such transformation will maximize operability of the system in its prospective environment.

Though our experiment included a considerable percentage of removable contexts and tasks, the costs saving is not proportional to this percentage. The reason is the overlap between the resources needed for remaining and removed con-

Factor	Research Group 1	Research Group 2
Removable contexts	**20/34 = 58.8 %**	**21/34 = 61.8 %**
True in all samples	3/34 = 8.8 %	3/34 = 8.8 %
False in all samples	3/34 = 8.8 %	2/34 = 5.9 %
Appear only in useless alternatives	3/34 = 8.8 %	12/34 = 35.3 %
Appear only in redundant alternatives	11/34 = 32.4 %	4/34 = 11.8 %
Removable tasks	**24/52 = 46.2 %**	**25/52 = 48.1 %**
Appear only in useless alternatives	6/52 = 11.5	20/52 = 38.5 %
Appear only in redundant alternatives	18/52 = 34.6	5/52 = 9.6 %
Negative Samples	**8/34 = 23.5 %**	**5/27 = 18.5%**
Samples convertible to positive	**5/8 = 62.5 %**	**3/5 = 60 %**

Fig. 15 Comparing the analysis outcome to the original goal model and collected samples

texts and tasks. For example, both tasks "send the room details to secretary via email" and "send room details to staff via mail" require gathering, storing, and updating the rooms data (they need the same monitoring effort). Thus, the removal of one of these tasks leads only to a partial saving of costs. To precisely compute costs saving, we consider the resources needed for task execution and context monitoring, similarly to what we proposed in [18]. The explicit specification of these resources allows for identifying overlaps between tasks and contexts and, hence, for accurate calculation of costs and savings. This enables better decision making about keeping or removing redundant alternatives.

Our approach reveals shortcomings in a designed requirements model and, consequently, in the system to-be for accommodating certain contextual variations in a deployment environment. Our analysis proposes loci in the requirements model where augmentation with other alternatives is desirable to maximize system ability in a certain environment. We have discussed augmentation with default non-contextualized alternatives that the system can support. However, this is just one possibility to solve the problem. In many cases, the discovery of these loci motivates changes in the organizational environment. For example, we encountered samples where there was no technician speaking a language in common with an overseas researcher and explain to him about the use of the equipments in the seminar rooms. In this case, the deployed system can be augmented to support a default alternative, such as showing a demo. However, the solution could be also at the organizational level by guaranteeing the presence of at least one technical staff who speaks English. Our analysis motivates also the organization where the system is to be deployed to make changes that maximize the requirements satisfaction likelihood.

5.3 Approach limitations and threats to validity

We discuss here the limitations of our approach and the threats to internal and external validity of our evaluation on the room booking case study. Table 3 summarizes the main threats to internal and external validity.

Threats to internal validity
Th1. Experiment time period influence
Th2. Limited number of samples
Th3. Sampling was not transparent to participants
Threats to external validity
Th4. Other factors to support the analysts decisions are missing
Th5. Sampling is not always possible for privacy and security concerns, law, monitoring costs, etc.
Th6. Experiments performed only in one domain

Table 3 Threats to internal and external validity

The definition of the sample collection period is a sensitive decision to take. Indeed, the contextual variations that may happen in one deployment environment could vary from one period to another. This will influence the comprehensiveness of the samples collected and, therefore, the analysis results. This is an internal threat (**Th1**) to validity in our case study, as we have studied the room booking scenarios between April and June, which is an active period in academia. Probably, collecting samples in the vacation months, such as August, would provide different results. However, for practical reasons we need to find a compromise between keeping the samples collection time short and the accuracy of the analysis results. This leads to another internal threat (**Th2**), i.e., the relatively limited number of samples that we used which probably does not cover all possible contextual variations in the deployment environment.

Moreover, the sampling process itself may need an automated system which means the development of another system. For example, contextual information such as the location changes of a person may need positioning systems, which cannot be collected through survey forms. Interviewing domain experts and analysing user samples sets are possible ways to overcome these issues. We leave the development of these techniques for future work. In our case study, the involvement of users (researchers, secretary, technicians, etc.) in the sampling process is a threat to internal validity (**Th3**). The sampling was not transparent to the system users. Indeed, they had to fill in our designated forms and answer to our interviews when sampling the environment.

Further analysis is needed to support the analyst's decisions about the augmentations to support and the redundancy to remove. Our analysis outputs alternative augmentations to support default solutions when the model lacks contextualized alternatives, and leaves the choice between them to the analyst. It also discovers the redundant alternatives and leaves the choice of selecting which redundant alternatives to keep and remove to the analyst. These decisions are not always easy as they could be subject to different non-functional factors and quality measures such as maximizing fault tolerance, accommodating user preferences, reducing development time and operating costs, and so on. This is a threat to external validity (**Th4**). Future work includes providing better support to the analyst in taking the right decisions. We aim to incorporate softgoals in the decision about which augmentations to enact and what redundant elements to remove or keep.

Obtaining context information is not always straightforward due to several reasons. People usually have privacy concerns that prevent the system from monitoring their behaviours, location, activity, and so on. Some laws may impose constraints on monitoring contextual information without people's clear consent. Compliance with law might be hard to judge and might differ from one location to another. Moreover, although obtaining more contextual information can maximize certainty while taking a context-based decision, there could be a trade-off between the certainty and the required costs to establish this monitoring. In our case study, and because we are members in the academic organization in which the experiment was done, we have not faced such obstacles: this is another threat to external validity (**Th5**). Our approach still needs to deal with different issues related to monitoring context (privacy concerns, law, costs, ..) and we leave this for future work. An additional threat to external validity (**Th6**) is that we draw our conclusions with respect to a single domain (seminar rooms booking in academia). Perhaps, applying our framework in other domains might reveal limitations in both of our modelling foundations and deployment process.

6 Related Work

The research in context modelling, such as [24–27], is about finding modelling constructs to represent software and user context as well as reasoning about context. There is, however, a gap between the context model and software behaviour model, i.e., between context and its use. Our work reduces such a gap at the requirements level and allows for answering questions like: "*how do we decide the relevant context?*", "*why do we need context?*" and "*how does context influence software and user behaviour adaptation?*". Moreover, and rather than representing context data directly and presuming a prior knowledge of them, we proposed context analysis to enable a systematic hierarchical refinement to analyse context and reveal the facts and the data that define it.

Recently, several papers studied the relationship between requirements and contextual variability. The comparison of this paper to our previous work can be found in Table 1. Salifu et al. [28, 29] investigate the use of problem descriptions to represent and analyse variability in context-aware software. However, the nature of problem frames does not help for representing high-variability. Goal models, instead, incorporate a large space of alternatives in one compact hierarchy. Moreover, quality measures, softgoals in goal models, are not a first-class requirement in problem frames and, thus, goal models offer the basis of representing the influence of contexts on these measures and prioritizing requirements alternatives accordingly [18, 10, 12]. Lapouchnian et al. [30] share with us similar vision of integrating context with intentional variability captured by goal models. We have investigated more in this direction and defined a set of variation points where context affects goal models with clear semantic and a systematic way to analyse context. Such effort allows us to develop a formal framework that natively supports automated reasoning.

Cheng et al. [31] deal with environmental uncertainty using KAOS goal models and conceptual domain models. Their core contribution is to introduce tactics to mitigate uncertainty. However, unlike us, they do not rely on sampling and reflecting the actual contextual variability which characterizes a certain operational environment. Thus, their mitigation strategies are based merely on the analyst's expertise and knowledge. Amyot et al. [32] model the impact of context on softgoals, using GRL models, in the context of service engineering. By not relying on collected contextual data from the operational environment where the system is to be deployed, their approach heavily relies upon a domain expert to perform the customization of the requirements model. Our approach, instead, is more systematic as it models and analyses explicitly the reasons and the method to follow when a customization is to take place.

Software variability modelling—mainly feature models [33, 34]—concerns modelling a variety of possible configurations of the software functionalities to allow for a systematic way of tailoring a product upon stakeholder choices. These approaches do not bridge the 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. Recently, several works recognized the role of context in the software products derivation [35–37]. Our work, though proposed in the area of goal-oriented requirements engineering, can be adapted to complement contextual software product lines, providing a clear semantic of the relation between contexts and features, an automated contextualized derivation process, a systematic way to identify context itself, and a process to customize a product during the deployment. Furthermore, our work is in line, and has the potential to be integrated, with the work in [38] and the FARE method proposed in [39], that show possible ways to integrate features with domain goals and knowledge to help for eliciting and justifying features. Recent work [40] focuses on the selection of feature model configurations that take into account soft constraints that represent desired quality attributes. Unlike us, such approach does not explicitly consider contextual factors, which

are fundamental to discriminate between possible and inapplicable alternatives.

Requirements monitoring adds specific code to 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 behaviour to them if a deviation occurs [8]. The objective is to have more robust, maintainable, and self-evolving systems. In [41], the GORE (Goal-Oriented Requirements Engineering) framework KAOS [5] was integrated with an event-monitoring system (FLEA [42]) to provide an architecture that enables runtime automated reconciliation between system goals and system behaviour with respect to a priori known or evolving changes of the system environment. A similar effort along the same line is Wang's requirements-driven diagnosis [43], which diagnoses software errors and their influence at the requirements level. Our work is also focused on monitoring. Unlike such approaches, however, we provide an explicit notion of context as a main driver to derive the requirements to meet and the adoptable alternatives to meet them. The deployment process we proposed can be adapted for runtime adaptation and reconciliation, as relying on correct models is fundamental to ensure runtime adaptation is in response to threats to requirements and eventually leads to requirements satisfaction.

Customizing goal models to fit to user skills and preferences was studied in [10, 12]. Such work can be used at deployment time where specific user skills and preference will rule out some alternatives and, thus, the deployed system will not need to support them. We envisage that a joint usage with our approach would enrich the set of aspects the deployment process can deal with. Liaskos et al. [23] study the variability modelling under the requirements engineering perspective and propose a classification of the intentional variability when OR-decomposing a goal. We focused on contextual variability, i.e., the unintentional variability, which influences the applicability and appropriateness of each goal model alternative. Reasoning with goal models has been studied in [44]. Adding context to goal models creates the need to integrate contextual reasoning and goal reasoning. Our previous and current work provides concrete answers to such need.

Pourshahid et al. [14] address a very important aspect concerning the customization and the improvement of a system, which is the alignment between business goals and the business process. It extends the User Requirements Notation (URN) [6, 45] with Key Performance Indicators (KPIs) [46], whose monitoring guides the alignment between goals and a business process. KPIs concerns the actual operation of the overall system and how the system performs. Context, as proposed in our work, concerns the characteristics of the environment surrounding the system. Each setting of such environment could activate certain requirements and make adpotable/inadoptable some of the system's alternatives to fulfil the activated requirements. Pourshahid et al. [47] introduce *aspects* as a mechanism to model the redesign patterns and choose the best applicable patterns based on their impact on different views of the system (process, goal, validation, and performance). Such approach could be combined with ours to allow for a comprehensive customization and adaptation since we would be able to monitor both the characteristics of the system environment and the operation of the system itself in practice and decide what modification to do both at design time and runtime.

The idea behind requirements testing is that testing has to start with requirements, as code correctness is unachievable in case requirements are incorrect [48, 49]. Our work is supportive to this argument especially when we design requirements for adaptive systems where the requirements model represents a high-level abstraction of the behavior of the program itself. Our work provides models and reasoning mechanisms to test and customize a requirements model to its deployment environment. We also plan to develop requirements-driven runtime adaptation, in line with the vision of requirements reflection [50], that maximizes this fitness continuously and allows the system to learn from its experience in its operational environment.

7 Conclusions and Future Work

In this paper, we have discussed deployment of requirements models as an essential activity of software deployment. Our argumentation is that requirements are not uniform and may differ from one environment to another. We provided a conceptual model, the contextual goal model, that explicitly captures the relation between context variations and requirements. We showed how this model is customized to fit each particular environment in which the system is to be deployed. Our process is based on exploring the system prospective environment and collecting samples reflecting the truth values of the contexts specified in the requirements models and then processing the model against the samples set. The processing aims to discover useless and redundant parts of the model. This, in turn, reduces the costs of the deployed system and avoids deploying unnecessary functionalities. Moreover, our proposed processing discovers loci in the model where extra solutions are needed to handle contexts for which the model lacks alternatives. This maximizes the operability of the system in its host environment. We developed a formal framework and an automated reasoning tool to support our approach. We also outlined a methodological process that guides the use of our framework to gather and deploy requirements.

We have presented a case study of deploying a requirements model for room reservation in an academic research environment. The case study included 2 research groups and their room reservations for meetings during 3 months. We processed the requirements model that we developed for room booking against the reported room reservations experience. The processing aimed at customizing the requirements model to its deployment environments. Our processing showed a significant amount of detected unusabilities and redundancies in the model. This is important to save the costs of deploying functionalities aimed to satisfy unnecessary requirements.

On the other hand, we discovered several cases in which the model lacks solutions for fulfilling the main requirements: having a room booked. Discovering these cases and augmenting the model with additional alternatives will maximize the system ability to meet users requirements in the deployment environment. The application of our framework in practice (Section 5) gave us insights on several challenges to address in a future work:

- Certain characteristics of the environment represent constraints on the system behaviour which are hard to capture by our contextual goal model. For example, the security and privacy policies in an organization might have a strong impact on the applicability of certain alternatives and the ability and permission to monitor certain contexts. By considering these organizational characteristics, we would be able to customize more holistically a requirements model.
- We assume that a high percentage of context variations happens within a period of time that is decidable by the analyst. However, this assumption may not be true since some context variations could be infrequent or periodic. Thus, the environment sampling period might need to be longer than desirable. Devising techniques to maximize the comprehensiveness of the collected samples in representing the deployment environment and at the same time reducing the sampling period is part of our future work.
- Our approach enables detection of unusability, redundancies, and shortcomings in the requirements model with respect to its deployment environment. While the analyst's decisions about removing unusable parts of the requirements model are often easy to take, decisions about what redundancies to accept and what augmentations to implement are often complex decisions. The reason for this complexity is that multiple factors play a role in that decision such as increasing flexibility and fault tolerance and accommodating different users preferences and so on. Developing a multi-criteria decision making techniques specialized for this problem is another research problem we aim to solve.
- Our approach deals with systems that are in the deployment stage. However, the system runtime operation might also reveal lack of alternatives, redundancy, uselessness, static contexts and so on. Maximizing the system awareness of its own behaviour and reflection to its current status autonomously or semi-autonomously is another research question to address in our future work. This deals with more challenging problems related to modifying already deployed and operating system at runtime to take benefit from the history of operation as a continuous process.

Acknowledgements

This work has been partially funded by the EU Commission, through the ANIKETOS and FastFix projects and by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie). We also thank Vitor E. Silva Souza for discussions that enriched the ideas of this paper.

References

1. Anthony Finkelstein and Andrea Savigni. A Framework for Requirements Engineering for Context-Aware Services. In *Proc. of the 1st International Workshop From Software Requirements to Architectures (STRAW 01)*, 2001.
2. Eric Yu. Modelling Strategic Relationships for Process Reengineering. *Ph.D. Thesis, University of Toronto*, 1995.
3. Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
4. Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards requirements-driven information systems engineering: The tropos project. *Information Systems*, 27(6):365–389, 2002.
5. Anne Dardenne, Axel van Lamsweerde, and Steve Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.
6. International Telecommunication Union (ITU-T). User Requirements Notation (URN) - Language Definition. Recommendation Z.151 (11/08). Geneva, Switzerland, 2008.
7. John Mylopoulos, Lawrence Chung, and Eric Yu. From Object-oriented to Goal-oriented Requirements Analysis. *Communications of the ACM*, 42(1):31–37, 1999.
8. Steve Fickas and Martin Feather. Requirements Monitoring in Dynamic Environments. In *Proc. of the 2nd IEEE International Symposium on Requirements Engineering*, pages 140–147, 1995.
9. Daniel Sykes, William Heaven, Jeff Magee, and Jeff Kramer. From Goals to Components: a Combined Approach to Self-management. In *Proc. of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, pages 1–8, 2008.
10. Bowen Hui, Sotirios Liaskos, and John Mylopoulos. Requirements Analysis for Customizable Software: A Goals-skills-preferences Framework. In *Proc. of the 11th IEEE International Conference on Requirements Engineering (RE'03)*, pages 117–126, 2003.
11. Sotirios Liaskos, Sheila A. McIlraith, Shirin Sohrabi, and John Mylopoulos. Representing and reasoning about preferences in requirements engineering. *Requirements Engineering*, 16:227–249, 2011. 10.1007/s00766-011-0129-9.
12. Sotirios Liaskos, Sheila McIlraith, and John 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.
13. L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy goals for requirements-driven adaptation. In *The 18th IEEE International Requirements Engineering Conference (RE)*, pages 125 –134, 27 2010-oct. 1 2010.
14. Alireza Pourshahid, Daniel Amyot, Liam Peyton, Sepideh Ghanavati, Pengfei Chen, Michael Weiss, and Alan Forster. Business process management with the user requirements notation. *Electronic Commerce Research*, 9:269–316, 2009. 10.1007/s10660-009-9039-z.
15. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Location-based Variability for Mobile Information Systems. In *Proc.*

of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08), volume 5074 of *LNCS*, pages 575–578. Springer, 2008.
16. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Location-based Software Modeling and Analysis: Tropos-based Approach. In *Proc. of the 27th International Conference on Conceptual Modeling (ER 2008)*, volume 5231 of *LNCS*, pages 169–182. Springer, 2008.
17. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A Goal Modeling Framework for Self-Contextualizable Software. In *Proc. of the 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2009)*, volume 29 of *LNBIP*, pages 326–338. Springer, 2009.
18. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A Goal-based Framework for Contextual Requirements Modeling and Analysis. *Requirements Engineering*, 15(4):439–458, 2010.
19. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Reasoning about Contextual Requirements for Mobile Information Systems: a Goal-based Approach. Technical Report DISI-10-029, University of Trento, April 2010.
20. Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. Goal-based self-contextualization. In *In the Forum of the 21st International Conference on Advanced Information Systems (CAiSE 09 - Forum)*, volume Vol-453, pages 37–42. CEUR-WS, 2009.
21. A. Lapouchnian, Y. Yu, and 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, pages 246–261. Springer-Verlag, 2007.
22. Colette Rolland, Carine Souveyet, and Camille Ben Achour. Guiding Goal Modeling using Scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, 1998.
23. Sotirios Liaskos, Alexei Lapouchnian, Yijun Yu, Eric Yu, and John Mylopoulos. On Goal-based Variability Acquisition and Analysis. In *Proc. of the 14th IEEE International Conference on Requirements Engineering (RE'06)*, pages 76–85, 2006.
24. Karen Henricksen and Jadwiga Indulska. A Software Engineering Framework for Context-Aware Pervasive Computing. In *Proc. of the 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, pages 77–86, 2004.
25. Stephen S. Yau and Junwei Liu. Hierarchical Situation Modeling and Reasoning for Pervasive Computing. In *Proc. of the 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'06)*, pages 5–10, 2006.
26. Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology Based Context Modeling and Reasoning using OWL. In *Proc. of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 18–22, 2004.
27. Tarak Chaari, Dejene Ejigu, Frédérique Laforest, and Vasile-Marian Scuturici. A Comprehensive Approach to Model and Use Context for Adapting Applications in Pervasive Environments. *Journal of Systems and Software*, 80(12):1973 – 1992, 2007.
28. Mohammed Salifu, Yijin Yu, and Bashar Nuseibeh. Specifying Monitoring and Switching Problems in Context. In *Proc. of the 15th International Conference on Requirements Engineering (RE'07)*, pages 211–220, 2007.
29. Mohammed Salifu, Bashar Nuseibeh, Lucia Rapanotti, and Thein Than Tun. Using Problem Descriptions to Represent Variabilities for Context-aware Applications. In *Proc. of the 1st International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2007)*, pages 149–156, 2007.
30. Alexei Lapouchnian and John Mylopoulos. Modeling Domain Variability in Requirements Engineering with Contexts. In *Proc. of the 28th International Conference on Conceptual Modeling (ER 2009)*, volume 5829 of *LNCS*, pages 115–130. Springer, 2009.
31. Betty Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *Model Driven Engineering Languages and Systems*, volume 5795 of *LNCS*, pages 468–483. Springer, 2009.
32. Daniel Amyot, Hanane Becha, Rolv Braek, and Judith E. Y. Rossebo. Next Generation Service Engineering. In *Proc. of the 1st ITU-T Kaleidoscope Academic Conference on Innovations in NGN: Future Network and Services (K-INGN 2008)*, pages 195–202, 2008.
33. Klaus Pohl, Günther Böckle, and Frank van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.
34. 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. *Annals of Software Engineering*, 5:143–168, 1998.
35. Herman Hartmann and Tim Trew. Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. In *Proc. of the 12th International Software Product Line Conference (SPLC'08)*, pages 12–21. IEEE Computer Society, 2008.
36. Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, and Jean-Paul Rigault. Modeling Context and Dynamic Adaptations with Feature Models. In *4th International Workshop Models@run.time (MRT'09)*, 2009.
37. Carlos Parra, Xavier Blanc, and Laurence Duchien. Context Awareness for Dynamic Service-oriented Product Lines. In *Proc. of the 13th International Software Product Line Conference (SPLC'09)*, pages 131–140, 2009.
38. Yijin Yu, Julio C. S. do Prado Leite, Alexei Lapouchnian, and John Mylopoulos. Configuring Features with Stakeholder Goals. In *Proc. of the 2008 ACM symposium on Applied computing (SAC'08)*, pages 645–649, 2008.
39. Muthu Ramachandran and Pat Allen. Commonality and Variability Analysis in Industrial Practice for Product Line Improvement. *Software Process: Improvement and Practice*, 10(1):31–40, 2005.
40. Ebrahim Bagheri, Tommaso Di Noia, Azzurra Ragone, and Dragan Gasevic. Configuring Software Product Line Feature Models Based on Stakeholders' Soft and Hard Requirements. In *Software Product Lines: Going Beyond*, volume 6287 of *LNCS*, pages 16–31. Springer, 2010.
41. Martin Feather, Steve Fickas, Axel Van Lamsweerde, and Cristophe Ponsard. Reconciling System Requirements and Runtime Behavior. In *Proc. of the 9th international workshop on Software specification and design (IWSSD'98)*, 1998.
42. Don Cohen, Martin S. Feather, K. Narayanaswamy, and Stephen S. Fickas. Automatic Monitoring of Software Requirements. In *Proc. of the 19th International Conference on Software Engineering (ICSE'97)*, pages 602–603, 1997.
43. Yiqao Wang, Sheila McIlraith, Yijin Yu, and John Mylopoulos. An Automated Approach to Monitoring and Diagnosing Requirements. In *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*, pages 293–302, 2007.

44. Paolo Giorgini, John Mylopoulos, Eleonora Nicchiarelli, and Roberto Sebastiani. Reasoning with Goal Models. In *Proc. of the 21st International Conference on Conceptual Modeling (ER'02)*, pages 167–181, 2002.
45. Michael Weiss and Daniel Amyot. Business process modeling with urn. *International Journal of E-Business Research*, 1(3):63–90, 2005.
46. Andreas Kronz. Managing of process key performance indicators as part of the aris methodology. *Corporate performance management*, pages 31–44, 2006.
47. Alireza Pourshahid, Gunter Mussbacher, Daniel Amyot, and Michael Weiss. Toward an aspect-oriented framework for business process improvement. *International Journal of Electronic Business*, 8(3):233–259, 2010.
48. Suzanne Robertson and James Robertson. Reliable Requirements through the Quality Gateway. In *Proc. of the 10th International Workshop on Database & Expert Systems Applications (DEXA'99)*, pages 357–363, 1999.
49. Suzanne Robertson. An Early Start to Testing: How to Test Requirements. In *Proc. of the EuroSTAR conference*, 1996.
50. Nelly Bencomo, Jon Whittle, Peter Sawyer, Anthony Finkelstein, and Emmanuel Letier. Requirements Reflection: Requirements as Runtime Entities. In *Proc. of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, pages 199–202, 2010.