A Goal Modeling Framework for Self-Contextualizable Software

Raian Ali, Fabiano Dalpiaz and Paolo Giorgini

University of Trento - DISI, 38100, Povo, Trento, Italy
{raian.ali, fabiano.dalpiaz, paolo.giorgini}@disi.unitn.it

Abstract. Self-contextualizability refers to the system ability to autonomously adapt its behaviour to context in order to maintain its objectives satisfied. In this paper, we propose a modeling framework to deal with self-contextualizability at the requirements level. We use Tropos goal models to express requirements; we provide constructs to analyse and represent context at each variation point of the goal model; and we exploit the goal and context analysis to define how the system satisfies its requirements in different contexts. Tropos goal analysis provides constructs to hierarchically analyse goals and discover alternative sets of *tasks* the system can *execute* to *satisfy* goals; our framework extends Tropos goal model by considering context at its variation points, and provides constructs to hierarchically analyse context and discover alternative sets of *facts* the system has to *monitor* to *verify* a context. A self-contextualizable promotion information system scenario is used to illustrate our approach.

Key words: GORE, Context Analysis, Self-Contextualization

1 Introduction

There is a continuous need for systems that are adaptive and have a degree of autonomy to take decisions by themselves with the minimum intervention of users or designers. As a baseline, we need to identify the parameters that stimulate the need for changing the system behavior, what choices the system has that reflect to each range of parameters, and how to select between choices when more than one are possible. Context, the reification of the environment in which the system is supposed to operate [1], has been considered as a main stimulus for system behavior changes, but still there is a lack of research that involves context with requirements. The relation between context and requirements is tight; context can influence the requirements set, the choices to satisfy a requirement, and the quality of each choice.

Goal analysis (*i** [2], Tropos [3], and KAOS [4]) provides a way to analyse high level goals and to discover and represent alternative sets of tasks that can be adopted to achieve such goals. Goal models – a mainstream technique in requirements engineering – are used to represent the rationale of both humans and software systems, and help for representing software design alternatives. These features are also important for self-contextualizable software that must allow for alternatives and have a rationale to reflect users and software adaptation to context for adopting one useful execution course [5].

From a goal-oriented perspective, a self-contextualizable software is assigned a set of goals and has to keep them satisfied in different contexts. Context is out of the control of the system, and we do not expect it to adapt to software; rather, we can build software that adapts to context. Moreover, context may influence not only the software behaviour, but also the behaviour and possible choices of users. Therefore, the software should reflect users adaptation to the variable context to effectively satisfy users expectations. For example, hotel reservation is a common goal for travelers, while reservation procedures can differ from one hotel to another, and the same hotel can have distinct procedures each applying to a specific type of customers.

In our previous work [6, 7, 8], a modeling and reasoning framework has been presented to tackle some research challenges concerning mobile information systems presented in [9]. The main idea was to associate location (environmental or contextual) properties to the variation points of the goal model, and then to extract a location model from such properties. In that work, location properties are defined without further analysis, i.e. specified in one step as one monolithic block. We believe that a hierarchical analysis and representation of location properties would help for having more understandable, modifiable, and reusable specifications. There is also a need for analysing the domain of discourse[1] of goals and location properties to express explicitly the elements each goal and location property concern. Moreover, modeling the adaptable task execution workflow according to location is still missing.

In this paper, we extend the modeling framework proposed in [6, 7, 8] trying to overcome its mentioned limitations. We use goal analysis in conjunction with our proposed context analysis to build self-contextualizable goal models[2]. We provide constructs to hierarchically analyse context so to identify the verifiable facts and the monitorable data (i.e., we specify the monitoring requirements). We also identify and illustrate how to create self-contextualizable execution workflow from the resulted goal model, and discuss the utilization of the overall framework.

The rest of this paper is structured as follows. In section 2, we overview Tropos goal modeling. In section 3, we explain our proposed framework, defining the goal model variation points, the context analysis constructs, the self-contextual workflow, and discussing the utilization of the overall framework. We discuss the related work in section 4, and we conclude and discuss our future work in section 5.

2 Tropos Goal Modeling: Overview

Goal analysis represents a paradigmatic shift with respect to object-oriented analysis. While object-oriented analysis fits well to the late stages of requirement analysis, the goal-oriented analysis is more natural for the earlier stages where the organizational goals are analysed to identify and justify software requirements and position them within the organizational system [10]. Tropos goal analysis projects the system as a

[1] Domain (or universe) of discourse refers to the part of the world under discussion.

[2] Although the term *"location"* was used as a synonym of *"context"*, we chose to use *"context"*, because we realized that it has more common and well-accepted definition that also fits to what we meant by *"location"*.

set of interdependent actors, each having its own strategic interests (*goals*). Goals are analysed, iteratively and in a top-down way, to identify the more specific sub-goals needed for satisfying that upper-level goals. Goals can be ultimately satisfied by means of specific executable processes (*tasks*).

In Fig.1, we show a partial Tropos goal model to clarify our goal analysis main concepts. Actors (*Customer IS* and *Mall Website*) have a set of top-level goals (*provide information to customer*), which are iteratively decomposed into subgoals by and-decomposition (all subgoals should be achieved to fulfil the top goal) and or-decomposition (at least one subgoal should be achieved to fulfil the top goal). The goal *provide information to customer* is and-decomposed into *establish network connection*, *get product identifier*, and *provide answer*; the goal *provide answer* is or-decomposed into *query mall database* and *get answer through website*. Goals are finally satisfied by means of executable tasks; the goal *"get product identifier"* can be reached by one of the tasks *"read RFID tag"*, *"read barcode"*, *"let customer type product ID"*.

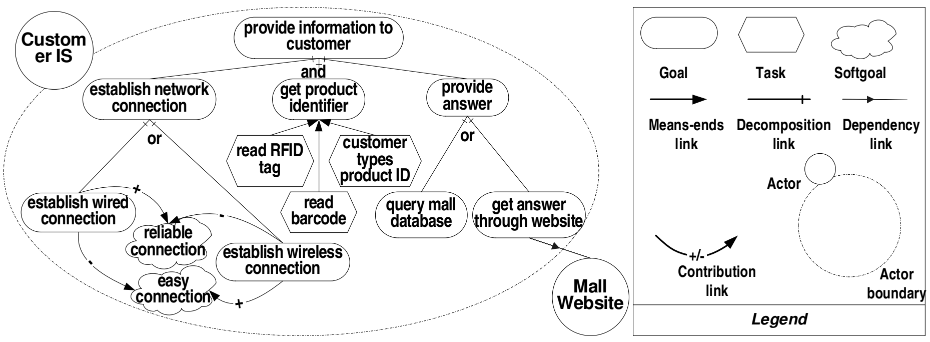

Fig. 1. Tropos goal model example.

A dependency indicates that an actor (*depender*) depends on another actor (*dependee*) to attain a goal or to execute a task: the actor *Customer IS* depends on the actor *Mall Website* for achieving the goal *get answer through website*. Soft-goals are qualitative objectives for whose satisfaction there is no clear cut criteria (*easy connection* is a rather vague objective), and they can be contributed either positively or negatively by goals and tasks: *establish wireless connection* contributes positively to *easy connection*, while *establish wired connection* contributes negatively to *easy connection*.

Goal analysis allows for different alternatives to satisfy a goal, but does not specify when each alternative can be adopted. Supporting alternative behaviours without specifying when to follow each of them rises the question *"why does the software support alternative behaviours and not just one?"*. On the other side, the consideration of different contexts the software has to adapt to, without supporting alternative behaviours rises the question *"what can the software do if context changes?"*. Analysing the different alternatives for satisfying a goal, and specifying the relation between each alternative and the corresponding context justify both alternatives and context, and help for having a self-contextualizable software.

3 Self-Contextualizable Software Modeling Framework

Fig. 2 represents a goal model for a promotion information system that is intended to interact with customers and sales staff, through their PDAs, in order to promote products in different ways. To make it self-contextualizable, we need to explicitly represent the relation between these alternatives and context. Contexts, labeled by $C1..C12$ in the figure, might be related to the following categories of the goal model variation points:

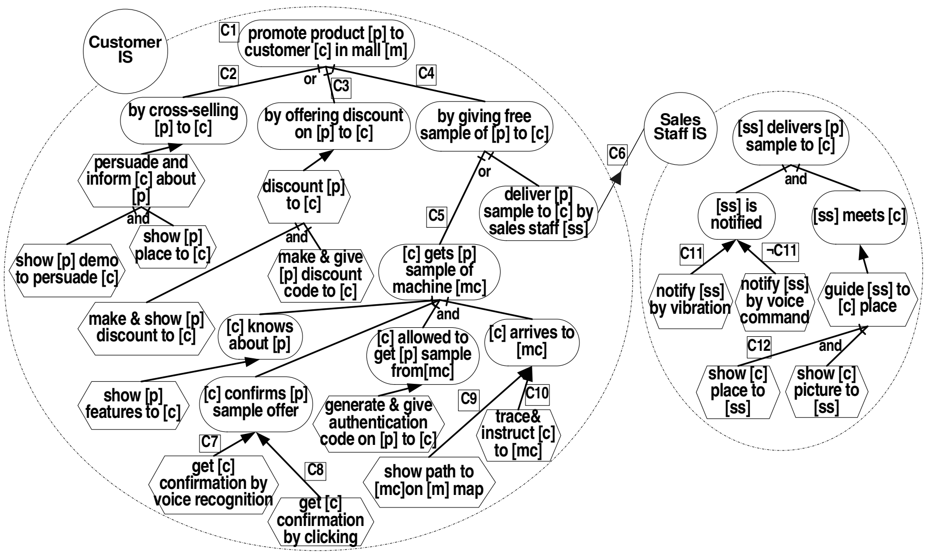
Customer IS
C1
promote product [p] to customer [c] in mall [m]
C2
or
C3
C4
by cross-selling [p] to [c]
by offering discount on [p] to [c]
by giving free sample of [p] to [c]
persuade and inform [c] about [p]
and
show [p] demo to persuade [c]
show [p] place to [c]
discount [p] to [c]
and
make & give [p] discount code to [c]
make & show [p] discount to [c]
or
C5
deliver [p] sample to [c] by sales staff [ss]
C6
[c] gets [p] sample of machine [mc]
and
[c] knows about [p]
[c] allowed to get [p] sample from[mc]
[c] arrives to [mc]
show [p] features to [c]
[c] confirms [p] sample offer
generate & give authentication code on [p] to [c]
C9
C10
trace& instruct [c] to [mc]
C7
C8
get [c] confirmation by voice recognition
get [c] confirmation by clicking
show path to [mc]on [m] map
Sales Staff IS
[ss] delivers [p] sample to [c]
and
[ss] is notified
[ss] meets [c]
C11
¬C11
notify [ss] by vibration
notify [ss] by voice command
guide [ss] to [c] place
C12
and
show [c] place to [ss]
show [c] picture to [ss]

Fig. 2. The parametrized goal model with the variation points annotation.

1. *Or-decomposition*: Or-decomposition is the basic variability construct, we still need to specify in which context each alternative in an Or-decomposition can be adopted. E.g. *"promoting the product by cross-selling"* can be adopted when the product can be used with another product the customer already has $(C2)$, while *"promoting by offering discount"* is adopted when product is discountable and interesting to the customer $(C3)$, and *"promoting by free sample"* can be adopted when product is free sampled and new to the customer $(C4)$. The alternative *"get free sample from a machine"* can be adopted when customer has experience with such machines and can reach the machine and start to use it in a little time $(C5)$.
2. *Actors dependency*: in some contexts, an actor might attain a goal / get a task executed by delegating it to another actor. E.g. the customer information system can satisfy the goal *"deliver a sample of the product to customer by sales staff"* by delegating it to the sales staff information system, when the corresponding sales staff is free, speaks a language common to the customer, has sufficient knowledge about the product, and is close enough to the customer $(C6)$.

3. *Goal/Task activation*: an actor, and depending on the context, might find necessary or possible triggering (or stopping) the desire of satisfying a goal/ executing a task. E.g. to initiate the goal *"promote product to customer in mall"*, there should be enough time to accomplish the promotion, the customer is not in a hurry or has to work, and the customer did not recently buy the product and does not have it in his/her shopping cart $(C1)$.
4. *And-decomposition*: a sub-goal / sub-task might (or might not) be needed in a certain context, that is some sub-goals / sub-tasks are not always mandatory to fulfil the top-level goal / task in And-decomposition. E.g. the sub-task *"show customer current place to sales staff"* is not needed if the customer stays around and can be seen directly by the sales staff $(C12)$.
5. *Means-end*: goals can be ultimately satisfied by means of specific executable processes (*tasks*). The adoption of each task might depend on the context. E.g. *"get customer confirmation by voice recognition"* can be adopted when the customer place is not noisy, and the system is trained enough on the customer voice $(C7)$, while the alternative *"get customer confirmation by clicking"* can be adopted when 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 $(C8)$. The task *"show path to sample machine on the mall e-map"* is adopted when customer can arrive easily to that machine $(C9)$, while *"trace and instruct customer to sample machine"* task is adopted when the path is complex $(C10)$. The task *"notify by vibration"* can be adopted when sales staff is using his PDA for calling $(C11)$, while *"notify by headphone voice command"* is adopted in the other case $(\neg C11)$.
6. *Contribution to soft-goals*[3]: the contributions to the softgoals can vary from one context to another. We need to specify the relation between the context and the value of the contribution. E.g. the goal *"establish wireless connection"* contributes differently to the softgoal *"reliable connection"* according to the distance between the customer's device and the wireless access point.

We need to analyse context to discover, represent, and agree on how it can be verified. Differently from the other research in context modeling (for a survey see [11]), we do not provide an ontology or a modeling language for representing context, but modeling constructs to hierarchically analyse context. Moreover, and in order to keep the link between the domain of discourse (i.e. the elements of the environment under discussion) between goal and context analysis, we propose parametrizing the goal and context models. Deciding the parameters is not straightforward and we might need several iterations to settle the final set of parameters.

Taking the parametrized goal *"by offering discount on product [p] to customer [c] in mall [m]"*, the analysis of its context $(C3)$, and the data conceptual model that the analyst could elicit from the leaf facts are shown in Fig. 3. Each leaf of the context analysis hierarchy represents an atomic fact that is verifiable on a fragment of the data conceptual model that the monitoring system has to instantiate.

[3] In the rest of this paper, we do not consider softgoals and contextual contribution.

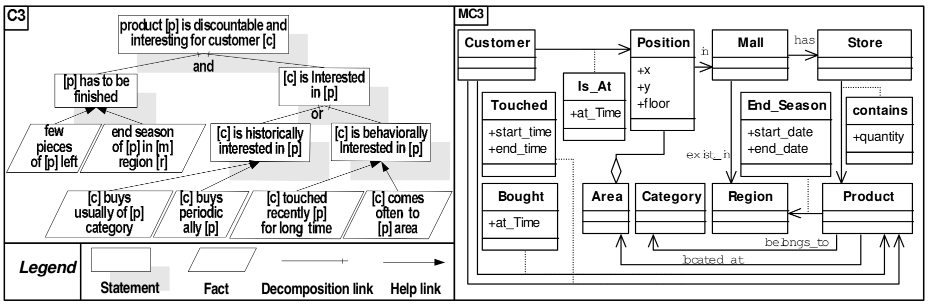

Fig. 3. The statement analysis for $C3$ and the correspondent data conceptual model $MC3$.

3.1 Context Analysis Constructs

We provide a set of constructs to analyse high-level contexts and elicit the atomic facts that are verifiable on monitorable data. Context, the reification of the environment surrounding a system, can be monitored but not controlled by the system itself [1]. Under this assumption, systems cannot change the context but should adapt to it for satisfying their objectives.

Definition 1 (Fact) *a boolean predicate specifying a current or a previous context, whose truth value can be computed objectively.*

The objective method to compute a fact truth value requires monitoring some characteristics and history of a set of relevant environment elements. Facts are graphically represented as parallelograms as in Fig.3. Examples of facts are the following:

– *"customer recently bought the product from the mall"*: to compute the truth value of this fact, the system can check the purchase history of the customer since a number x of days ago.
– *"two products are usually sold together"*: the system can check the sales record of all customers and check if the two products $p1$ and $p2$ are often sold together.
– *"product is not in the shopping cart of the customer"*: the system can use an RFID reader in the cart to check if the product (identified by its RFID tag) is in the cart of the customer.

Definition 2 (Statement) *a boolean predicate specifying a current or a previous context, whose truth value cannot be computed objectively.*

Statement verification could not be objectively done because the system is not able to monitor and get all the data needed to compute the truth value of a statement, or because there could be no consensus about the way of knowing the truth value of a statement. Anyhow, to handle such problem we adopt a relaxed confirmation relation between facts, which are objectively computable by definition, and statements, in order to assign truth values to statements. We call this relation *"help"* and define it as following:

Definition 3 (Help) *Let f be a fact, s be a statement.* $help(f, s) \iff f \rightarrow s$

The relation $help$ is strongly subjective, since different stakeholders could define different $help$ relations for the same statement, i.e. one stakeholder could say $help(f_1, s) \wedge help(f_2, s)$, whereas another one could say $help(f_2, s) \wedge help(f_3, s)$. Statements are graphically represented as shadowed rectangles, and the relation $help$ is graphically represented as a filled-end arrow between a fact and a related statement as in Fig.3. Examples of statements and help relations are the following:

– *"customer does not have the product"*: is a statement since the system cannot objectively compute its truth value. The system can get some evidence of this statement verifying two facts: *"customer did not buy the product from the mall recently"*, and *"the product is not in the cart of the customer"*, but these facts do not ensure that the customer does not have the product (e.g. the system cannot verify if the customer was given the product as a gift).
– *"customer is interested in the product"*: is a statement that different stakeholders would define differently how we can get an evidence about it. Moreover, to verify it, the stakeholder might state a variety of other conditions which are not necessarily computable due to the lack of some necessary data the system cannot monitor. However, we might relax this problem using the $help$ relation; a possible solution is to specify several facts that help the verification of the statement like following: *"customer buys the product periodically"*, *"customer buys usually from the product category"*, *"customer often comes to the product area"*, or *"customer holds recently the product for long time"*.

Definition 4 (And-decomposition) *Let* $\{s, s_1, \ldots, s_n\},\ n \geq 2$ *be statements (facts).* $and\_decomposed(s, \{s_1, \ldots, s_n\}) \iff s_1 \wedge \ldots \wedge s_n \rightarrow s$

Definition 5 (Or-decomposition) *Let* $\{s, s_1, \ldots, s_n\},\ n \geq 2$ *be statements (facts).* $or\_decomposed(s, \{s_1, \ldots, s_n\}) \iff \forall i \in \{1, \ldots, n\}, s_i \rightarrow s$

Decomposition is graphically represented as a set of directed arrows from the sub-statements (sub-facts) to the decomposed statement (fact) and annotated by the label *And* or *Or*. Examples of decompositions are the following:

– *"customer is interested in the product"* is a statement verified if the sub-statements *"customer is historically interested in the product"* ***or*** *"customer is behaviourally interested in the product"* are verified.
– *"customer did not get the product from the mall recently"* is a fact that is verified if the sub-fact *"customer does not have the product in his/her cart"* ***and*** the sub-fact *"the customer did not buy the product from the mall recently"* is true.

As discussed in [1], context is a reification of the environment that is whatever in the world provides a surrounding in which the system is supposed to operate. Each single fact and statement is a context, and our proposed reification hierarchy relates different subcontexts into one more abstract. Moreover, by considering that context is the reification of the environment, our context analysis is motivated by the need for constructs to analyse context to discover by the end the relevant atomic data that represent that environment, i.e. the data the system has to monitor.

3.2 From Goals to Self-Contextualizable Workflow

Specifying the relation between context and goal alternatives is not enough to define how self-contextualizable software will execute tasks to achieve goals depending on the context. In order to handle this issue, two questions should be answered:

1. *How are goals / tasks sequentialized?*. For example, if the achievement of a goal g requires the execution of $t1 \wedge t2$, we have to specify if $t1$ is executed before or after or in parallel with $t2$.
2. *How does the system choose between alternatives when more than one are adoptable?*. For example, if a goal g can be reached through $g1 \vee g2$, we need to specify which one to follow. The intervention of stakeholders is required to prioritize alternatives along the goal hierarchy (for goals and tasks) to face cases where multiple options are possible in some contexts.

A possible self-contextualizable goal achievement workflow is shown in the activity diagram of Fig. 4. We have used activities to represent the tasks of the goal model of Fig. 2. The context of the alternative with the highest priority is evaluated first, and if it is confirmed that alternative is selected and carried out (even if other alternatives are also applicable). In our example, stakeholders stated that the priority for the alternative *"promotion by cross-selling"* is higher than the priority of the alternative *"promotion by free sample"*, whose priority is in turn higher than that of *"promotion by discounting"* alternative. Our depicted workflow reflects such prioritization – and also other prioritization on the other sets of sub-alternatives – by evaluating the contexts associated to the alternative with a higher priority first.

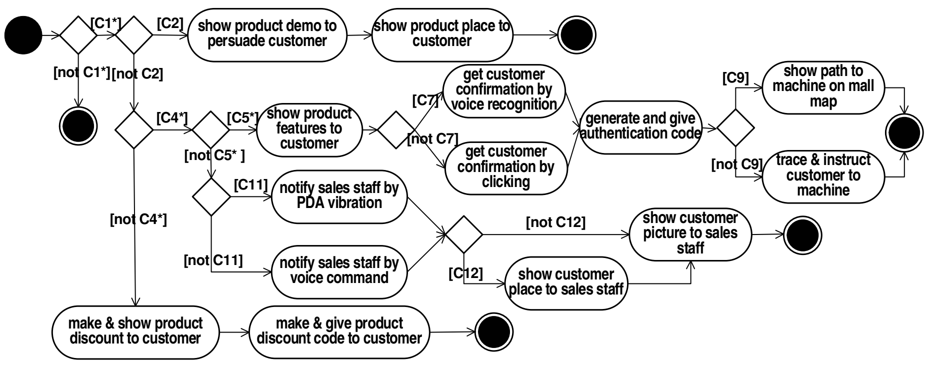
[C1*]
[C2]
show product demo to persuade customer
show product place to customer
[not C1*]
[not C2]
[C4*]
[C5*]
show product features to customer
[C7]
get customer confirmation by voice recognition
[not C7]
get customer confirmation by clicking
generate and give authentication code
[C9]
show path to machine on mall map
[not C9]
trace & instruct customer to machine
[not C5*]
[C11]
notify sales staff by PDA vibration
[not C11]
notify sales staff by voice command
[not C12]
show customer picture to sales staff
[C12]
show customer place to sales staff
[not C4*]
make & show product discount to customer
make & give product discount code to customer

Fig. 4. A possible self-contextualizable goal achievement workflow.

Fig. 4 introduces an additional concern, that is the accumulation of context at each variation point. Looking at Fig 2, we highlight that the confirmation of $C1$ is not sufficient to assure the existence of a workflow for the achievement of the top-level goal. For example, if $C2$-$C3$-$C4$ are all false, no task execution workflow is possible. Thus, finding an alternative for the top-level goal *"promote product [p] to customer [c] in mall [m]"* needs checking $C1^*$, defined as follows:

$C1^* = C1 \wedge (C2 \vee C3 \vee C4^*)$
$C4^* = C4 \wedge (C5^* \vee C6^*)$
$C6^* = C6 \wedge (C11 \vee \neg C11) \wedge (C12 \vee \neg C12) = C6$
$C5^* = C5 \wedge (C7 \vee C8) \wedge (C9 \vee C10)$

In order to evaluate the applicability of the alternative goal *"customer [c] gets product [p] free sample from a dedicated machine [mc]"*, we have to check the accumulated context $C5^*$, evaluating both $C5$ and the contexts that are lower in the goal model hierarchy: ($C7$ or $C8$) and ($C9$ or $C10$). If $C5^*$ is false, this means that no satisfaction alternative for the considered goal can be adopted.

3.3 Framework Utilization

Contextualization: the decomposition of the system into the functional part captured by goal model and the monitoring part that is captured by context analysis, and the association between variation points of goal model and the analysed context allow for a systematic contextualization of the software at the goal level of abstraction. All the functionalities needed by the alternative execution courses to satisfy goals has to be developed and then the contextualization has to be done. Contextualization can be done at two different times:

– *contextualization at deployment time*: when deploying the software to one specific environment, and when we know a priori some contexts that never change in that environment, we can consequently exclude from the deployed software some alternative sets of functionality that are never applied at that environment, as such functionalities will be never used and redundant. E.g. if the software is going to be deployed in a mall where the noise level is always high due to the nature of that mall (for instance, the mall is located in an open area, or the mall sells products of a specific nature), the context $C7$ will never, or rarely, be satisfied, and therefore the deployed software for that mall can exclude the functionality of voice recognition as a way of interaction with customer.
– *contextualization at runtime*: some other contexts are highly variable and should be monitored at runtime to know what behaviour to adopt. Consequently, the software has to monitor context, instantiating the monitoring data conceptual model, validating facts and inferring statements assigned to the variation points, and then adopt the suitable software alternative course of execution. E.g. the distance between customer and the self-service machine is a context which has always different values, and whether the software has to guide the customer to the machine using the alternative functionality *"trace and instruct customer to machine"*, or *"show path to machine on the mall map"* depends on the actual value of this variable distance.

Capturing and justifying monitoring requirements: our framework uses goal analysis in conjunction with context analysis to reduce the gap between the variability of software, at the goal level, and the variability of context, and helps for identifying and justifying both the functional and the monitoring software requirements. While goal analysis helps to elicit and justify each software functionality and to position it within the set of the organizational goals, the context analysis we propose, helps to elicit and justify the data the system has to monitor. The system has to monitor data to validate leaf facts so to confirm top level facts or statements that are used to decide which alternative to adopt for satisfying some organizational goal.

Reusability and scalability: systems change continuously; managing evolution is a hard task that is highly expensive and error-prone. Structuring the software functional requirements using the hierarchy of goal model and the monitoring requirements using the hierarchy of context analysis makes it more feasible to modify, extend, and /or reuse the software for another evolution of the system to operate in a new context or/and for a different group of stakeholders. The same goal model, or parts of it – and hence the same software functionality – can be contextualized differently by different stakeholders. We might need only to change the statements at each variation point, which might influence the data to be monitored.

The hierarchical context analysis has the potential to make a context (i) more understandable for the stakeholders, (ii) easily modifiable as it is not given as one monolithic block, and (iii) more reusable as parts of the statement analysis hierarchy can be also used for other variation points or other stakeholders context specifications. Specifying for each fact the related fragments of the data conceptual model is useful for purpose of tracking. For example, if for some reason, a group of stakeholders decided to drop, to alter, or to reuse one alternative, statement, or fact, we still can track which fragments in the conceptual data model could be influenced.

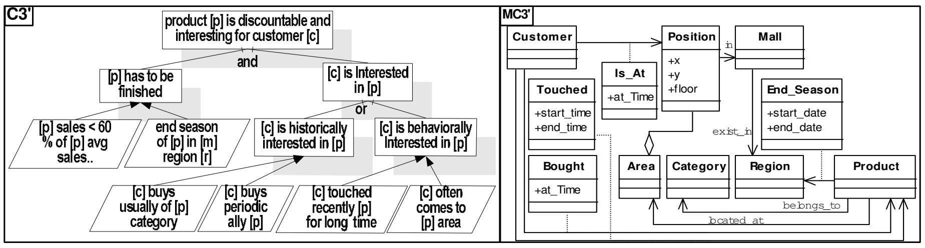
C3'
product [p] is discountable and interesting for customer [c]
and
[p] has to be finished
[c] is Interested in [p]
or
[p] sales < 60 % of [p] avg sales..
end season of [p] in [m] region [r]
[c] is historically interested in [p]
[c] is behaviorally Interested in [p]
[c] buys usually of [p] category
[c] buys periodic ally [p]
[c] touched recently [p] for long time
[c] often comes to [p] area
MC3'
Customer
Position
+x
+y
+floor
in
Mall
Is_At
+at_Time
Touched
+start_time
+end_time
End_Season
+start_date
+end_date
exist_in
Bought
+at_Time
Area
Category
Region
Product
belongs_to
located_at

Fig. 5. The modified context $C3'$ and the correspondent modified data model $MC3'$.

For example, a certain mall administration could decide that to promote by offering discount, it is not requested that *"few pieces of the product left"*, and it is, instead, requested that the fact *"[p] sales < 60 percent of the [p] historical sales average for the last 15 days"* is true. In this new context specification ($C3'$), one part of $C3$ is deleted, one is reused, and another is added as shown in Fig. 5. Removing the fact *"few pieces of product[p] remained"*, leads to remove the corresponding data conceptual model fragments (the class *store*, and the association class *contain*). To verify the new fact, the system needs the sales records that are already represented in the data model fragment $MC3$. Therefore, the new data conceptual model for $C3'$ will be like shown in Fig. 5.

4 Related Work

The research in context modeling, (e.g. [12]), concerns finding modeling constructs to represent software and user context, but there is still a gap between the context model and software behaviour model, i.e. between context and its use. We tried to reduce such gap and to allow for answering questions like: "*how do we decide the relevant context?*", "*why do we need context?*" and "*how does context influence software and user behaviour adaptation?*". Modeling context information should not be done as a standalone activity; context should be defined jointly with the analysis we do for discovering the alternative software behaviours. Salifu et al. [13] investigate the use of problem descriptions to represent and analyse variability in context-aware software; the work recognizes the link between software requirements and context information as a basic step to design context aware systems.

Software variability modeling, mainly feature models [14, 15], concerns modeling a variety of possible configurations of the software functionalities to allow for a systematic way of tailoring a product upon stakeholders choices, but there is still a gap between each functionality and the context where this functionality can or has to be adopted, the problem we tried to solve at the goal level. Furthermore, our work is in line, and has the potential to be integrated, with the work in [16] and the FARE method proposed in [17] that show possible ways to integrate features with domain goals and knowledge to help for eliciting and justifying features.

Requirements monitoring is about insertion of a code into a running system to gather information, mainly about the computational performance, and reason if the running system is always meeting its design objectives, and reconcile the system behaviour to them if a deviation occurs [5]. The objective is to have a more robust, maintainable, and self-evolving systems. In [18], a GORE (goal-oriented requirements engineering) framework KAOS [4] was integrated with an event-monitoring system (FLEA [19]) to provide an architecture that enables the runtime automated reconciliation between system goals and system behaviour with respect to a priori anticipated or evolving changes of the system environment. Differently, we propose model-driven framework that concerns an earlier stage, i.e. requirements, with the focus on identifying requirements together with context, and hierarchically analysing and representing context and eliciting the monitoring data.

Customizing goal models to fit to user skills and preferences was studied in [20, 21]. The selection between goal satisfaction alternatives is based on one dimension of context, i.e. user skills, related to the executable tasks of the goal hierarchy, and on user preferences which are expressed over softgoals. Lapouchnian et al. [22] propose techniques to design autonomic software based on an extended goal modeling framework, but the relation with the context is not focused on. Liaskos et al [23], study the variability modeling under the requirements engineering perspective and propose a classification of the intentional variability when Or-decomposing a goal. We focused on context variability, i.e. the unintentional variability, which influences the applicability and appropriateness of each goal satisfaction alternative.

5 Conclusions and Future Work

In this paper, we have proposed a goal-oriented framework for self-contextualizable software systems. We have used goal models to elicit alternative sets of executable tasks to satisfy a goal, and we have proposed the association between the alternative software executions and context. In turn, context is defined through statement analysis that elicits alternative sets of facts the system has to verify on monitorable data so to confirm the high level statements. Analysing facts will also lead to identify the data conceptual model the monitoring system has to instantiate to enable facts verification. Facts are verified upon the monitorable data to confirm statements that restrict the space of goal satisfaction alternatives. We have also shown how to construct self-contextualizable goal execution workflows that allow for the construction of an exact execution course of tasks to satisfy goals according to the context. Doing this, we specify the requirements of the monitoring system and the reasoning the system has to do on context to construct, autonomously, a contextualized goal execution course.

As future work, we will define models covering all development phases of self-contextualizable software and a process to facilitate the development of high quality self-contextualizable software. We also want to find a formalization and reasoning mechanisms that fit well to the modeling framework introduced in this paper. We will work on more complex case studies in order to better validate our approach. Similarly to features [24], contexts suffer from interaction problems; for instance, there could be contexts that contradict with others on one goal satisfaction alternative. Therefore, supporting tools and reasoning techniques should be proposed to assist the design and verification of models.

Acknowledgement

This work has been partially funded by EU Commission, through the SERENITY, and COMPAS projects, and by the PRIN program of MIUR under the MEnSA project. We would also like to thank Prof. Jaelson Brelaz de Castro for the valuable discussion we had about this work.

References

1. Finkelstein, A., and Savigni, A.: A framework for requirements engineering for context-aware services. Proc. 1st Int. Workshop on From Software Requirements to Architectures (STRAW), 2001
2. Yu, E.: Modelling strategic relationships for process reengineering. Ph.D. Thesis, University of Toronto (1995)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3) (2004) 203–236
4. Dardenne, A., Van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of computer programming 20(1-2) (1993) 3–50
5. Fickas, S., Feather, M.: Requirements monitoring in dynamic environments. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society Washington, DC, USA (1995) 140

6. Ali, R., Dalpiaz, F., Giorgini, P.: Location-based variability for mobile information systems. In Bellahsene, Z., L-eonard, M., eds.: CAiSE. Volume 5074 of Lecture Notes in Computer Science., Springer (2008) 575–578
7. Ali, R., Dalpiaz, F., Giorgini, P.: Modeling and analyzing variability for mobile information systems. In Gervasi, O.,Murgante, B., Lagan'a,A., Taniar,D., Mun,Y.,Gavrilova, M.L., eds.: ICCSA (2). Volume 5073 of Lecture Notes in Computer Science., Springer (2008) 291–306
8. Ali, R., Dalpiaz, F., Giorgini, P.: Location-based software modeling and analysis: Tropos-based approach. In Li, Q., Spaccapietra, S., Yu, E., Oliv-e, A., eds.: ER. Volume 5231 of Lecture Notes in Computer Science., Springer (2008) 169–182
9. Krogstie, J., Lyytinen, K., Opdahl, A., Pernici, B., Siau, K., Smolander, K.: Research areas and challenges for mobile information systems. International Journal of Mobile Communications, Inderscience Publishers Ltd, 2004, 2, 220–234
10. Mylopoulos, J., Chung, L., Yu, E. From object-oriented to goal-oriented requirements analysis Commun. ACM, ACM, 1999, 42, 31-37
11. Strang, T., Linnhoff-Popien, C. A context modeling survey. Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp, 2004
12. Henricksen, K., Indulska, J.: A software engineering framework for context-aware pervasive computing. In: Proc. Second IEEE Intl. Conference on Pervasive Computing and Communications (PerCom-04). (2004) 77
13. Salifu, M., Yu, Y., Nuseibeh, B. Specifying Monitoring and Switching Problems in Context Proc. 15th Intl. Conference on Requirements Engineering (RE'07), 2007, 211–220
14. Pohl, K., B-ockle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
15. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: Form: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. 5 (1998) 143–168
16. Yu, Y., do Prado Leite, J.C.S., Lapouchnian, A., Mylopoulos, J.: Configuring features with stakeholder goals. In: SAC -08: Proceedings of the 2008 ACM symposium on Applied computing, New York, NY, USA, ACM (2008) 645–649
17. Ramachandran, M., Allen, P.: Commonality and variability analysis in industrial practice for product line improvement. Software Process: Improvement and Practice 10(1) (2005) 31–40
18. Feather, M. S., Fickas, S., Lamsweerde, A. V., Ponsard, C. Reconciling System Requirements and Runtime Behavior. Proceedings of the 9th international workshop on Software specification and design IWSSD '98, IEEE Computer Society, 1998, 50
19. Cohen, D., Feather, M. S., Narayanaswamy, K., Fickas, S. S. Automatic monitoring of software requirements ICSE '97: Proceedings of the 19th international conference on Software engineering, ACM, 1997, 602–603
20. Hui, B., Liaskos, S., Mylopoulos, J.: Requirements analysis for customizable software goals-skills- preferences framework. In: RE, IEEE Computer Society (2003) 117–126
21. Liaskos, S., McIlraith, S., Mylopoulos, J.: 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.
22. Lapouchnian, A., Yu, Y., Liaskos, S., Mylopoulos, J.: Requirements-driven design of autonomic application software. In: Proc. 2006 conference of the Center for Advanced Studies on Collaborative research (CASCON -06), ACM (2006) 7
23. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., Mylopoulos, J.: On goal-based variability acquisition and analysis. In: Proc. 14th IEEE Intl. Requirements Engineering Conference (RE-06). (2006) 76–85
24. Cameron, E.J., Griffeth, N., Lin, Y.-J., Nilson, M.E., Schnure, W.K., Velthuijsen, H.: "A feature-interaction benchmark for IN and beyond," Communications Magazine, IEEE , vol.31, no.3, pp.64–69, Mar 1993