On the behavior of context-sensitive systems

Jéssyka Vilela, Jaelson Castro, João Pimentel, Paulo Lima

Universidade Federal de Pernambuco
Recife, Brazil

{jffv, jbc, jhcp, plc2}@cin.ufpe.br

Abstract. Software systems are being used in ever more diverse and dynamic environments where they have to routinely and efficiently adapt to changing environmental conditions. Therefore, they must detect variations in their operating context and adapt their behavior in response to such variations. However, specifying monitoring and adaptation can be difficult due to their dependence on the contextual elements, which need to be made explicit. The variable nature of these systems calls for new approaches to create systems that can adapt to context changes. This paper proposes the GOals to Statecharts (GO2S) process to systematically derive the behavior of context-sensitive systems from requirements models. This is an iterative process centered on the incremental refinement of a goal model, obtaining different views of the system (design, contextual, behavioral). We illustrate our proposal with the meeting scheduler exemplar and we conducted a controlled experiment in order to evaluate our process. The experiment results show that the structural complexity of the group that used our GO2S approach was lower and the mean of behavioral similarity and the time spent was higher than control group. Besides, the subjects agreed that the GO2S process is easy to use indicating that it is possible to reproduce the process and it is understandable.

Keywords: Behavior, Adaptation, Context-sensitive system, Goal Model, Monitoring.

1 Introduction

There is a growing body of research on the use of context-sensitivity as a technique for developing computing applications. The sensitivity to context illustrates the possibility to react, according to certain predefined rules or based on intelligent stimulus. Thus, context-sensitive systems are flexible and capable of acting autonomously on behalf of their users. Such systems need to dynamically adapt their behavior.

It is of paramount importance to specify and analyze the intended behavior of such systems before they are fully implemented. The behavioral specification is used as input to the analysis, which explores the range of possible orderings of interactions, opportunities for concurrency, and time-based interaction dependencies among system elements [1]. Many notations may be used to capture behavioral information of context-sensitive systems such as Labelled Transition Systems (LTS), Petri Nets, and Statecharts. In this paper, we rely on Statechart [2], a popular visual formalism to represent the behavioral view of a system.

It is important to note that Non-Functional Requirements (NFRs) affect both the structural and behavioral aspects of the system (architecture). Therefore, they need to be operationalized and refined. Besides, NFRs should be taken in consideration when deciding which variant is more appropriate in a given context. Therefore, NFRs are critical and must be elicited, analyzed, and properly handled.

Goal models have been used as an effective means to capture the interactions and information-related requirements of adaptive systems [3]. A possible reason is that they incorporate a space of alternatives of operations sets, i.e. variants, which gives more flexibility to meet stakeholders' goals in a dynamic environment [4].

It is well known that requirements (for example, described in terms of goal models) and behavior (expressed using statecharts) must be related. In fact, they are inter-twined [5].

In this work, we propose the GO2S process to obtain the behavior of context-sensitive systems (expressed as statecharts) from requirements (described as goal models) and we illustrate our approach with the meeting scheduler exemplar. A key contribution is the proposal of the operationalization of softgoals, the monitoring and adaptation tasks in the same behavioral contextual design goal model. Therefore, our approach does not need any additional notation for adaptation modelling; it uses the elements of a contextual goal model, allowing the software engineer to visualize their impact on the system's behavior.

The experiment results show that the structural complexity of the group that used our GO2S approach was lower and the mean of behavioral similarity and the time spent was higher than control group. Besides, the subjects agreed that the GO2S process is easy to use indicating that it is possible to reproduce the process and it is understandable.

The benefits of obtaining the behavior of context-sensitive are manifold [1]: the models can be used as a communication channel among stakeholders during system-development activities; it will improve the confidence that the context-sensitive system will be able to achieve its goals; and it will support the reasoning. Hence, it will be possible to analyze properties such as system's completeness, correctness as well as the satisfaction of some quality attribute. Besides, since the resulting models are statecharts without any extension, they are amenable to simulation and code generation using existing tools.

This paper is structured as follows. In Section 2, we overview the research baseline for this work. In section 3, we explain our proposal, and discuss its use with a running example. Section 4 presents the contributions of this work and points out some open issues. Later, we present related works in Section 5, and we conclude and present our future works in Section 6.

2 Background

The GO2S process proposed in this paper consists of an incremental refinement of a goal model, towards a statechart, following the twin peaks concept [5]. The following sub-sections provide a brief overview of these concepts.

2.1 Contextual Goal Model

Goal-Oriented Requirements Engineering (GORE) is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements [6].

Goal models are a way to capture and refine stakeholder intentions to generate functional and non-functional requirements. The notation used in this paper is based on the one described by [3] which has goals, tasks, softgoals and contribution links (Make (++), Help (+), Hurt (-) or Break (--)).

A contextual goal model extends a goal model with context annotations in order to specify the variation points that are context-dependent. Context is defined as a partial state of the world that is relevant to an actor's goals [4]. The notation used in this paper to represent contexts in goal models is based on [4]. Thus, contexts in our work can be associated with the following variation points in a goal model:

- Or-refinement: the adoptability of a subgoal (subtask) may require a specific context to hold as a pre-condition for the applicability of the corresponding goal model variant;
- And-refinement: the satisfaction (execution) of a subgoal (subtask) in this refinement is needed only in certain contexts. Although this is syntactically equivalent to an or-refinement, the semantic is different [4]. A context on an and-refinement influence the need for the reaching or executing the corresponding subgoal/subtask, while a context on an or-refinement is itself needed to hold before adopting the corresponding subgoal/subtask. This semantic difference is essential to decide which requirements and alternatives will be active when a context change occurs at runtime;
- Contribution to softgoals: the contribution of softgoals can vary from one context to another.

Each context identified in a contextual goal model must be refined [4] to allow it to be checked. The contextual refinement has a tree-like structure (Fig. 1.) in which the root of this model is the context, and statements and facts are its nodes. Statements cannot be verified directly in a context, e.g. "*The meeting is not urgent*", and are subjective assertions that do not have clear criteria to be evaluated against. Facts, on the other hand, are predicates which truth values can be verified in a context, e.g. "*The meeting date is more two days away*". To obtain a verifiable context, all statements are refined into facts and sub-statements, until there are only facts left. Fig. 1. presents the refinement of the context related to *Collect timetables by email* task of our running example. It will be true if the number of participants is high, or the meeting is not urgent (which can be checked by the *The meeting date is more than 2 days away* fact), or the participants usually answer meeting requests by email (i.e. *The participants answered more than 50% of timetables requests*).

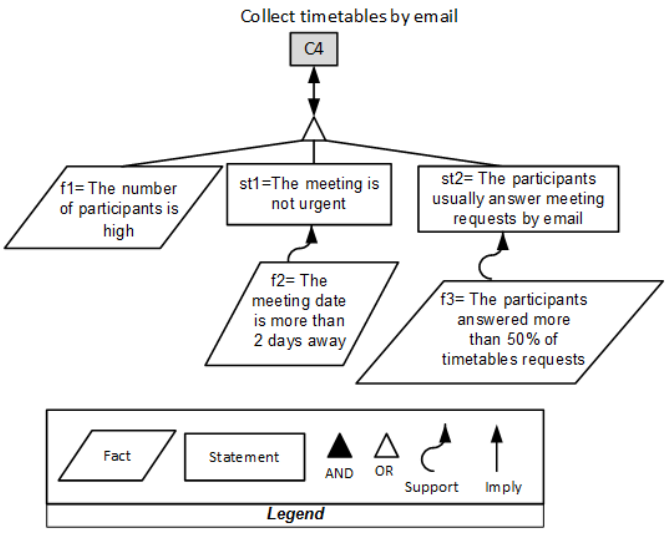

Collect timetables by email
C4
f1= The number of participants is high
st1=The meeting is not urgent
st2= The participants usually answer meeting requests by email
f2= The meeting date is more than 2 days away
f3= The participants answered more than 50% of timetables requests
Fact
Statement
AND
OR
Support
Imply
Legend

Fig. 1. Refinement of the context *Collect timetables by email*.

2.2 Statecharts

Statecharts were proposed by David Harel [2] and became a popular visual formalism for modelling reactive systems. It can be used to represent the behavioral view of a system. It is worth noting that they also support concurrency and hierarchy of states.

The main elements of statecharts are states, events, transitions, actions and regions. States are conditions during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event.

Transitions capture a change of state caused by the occurrence of some associated event. A transition may be guarded by some condition, represented by a condition name or an expression enclosed between brackets. A guard captures a necessary condition for transition firing. States are represented as boxes and transitions between states represented as arrows.

An action is an auxiliary operation associated with a state transition that is applied when the transition is activated. Statecharts also support the nesting of states. Concurrency is represented by dividing a composite state into regions that are shown separated by dotted lines.

3 GOals to Statecharts (GO2S) Process

We propose the GO2S process to systematically derive the behavioral view of context-sensitive systems (modeled as statecharts), from system's requirements (modeled as goal models). This process comprises six activities (see Fig. 2): *Construction of design goal model; Specification of contextual variation points; Specification of monitoring and adaptation; Specification of flow expressions, Statechart derivation and refinement* and *Prioritization of variants*. This is an iterative process that produces progressively more detailed requirements and design specifications. The detailed explanation of the activities proposed in our process is presented in the following sub-sections.

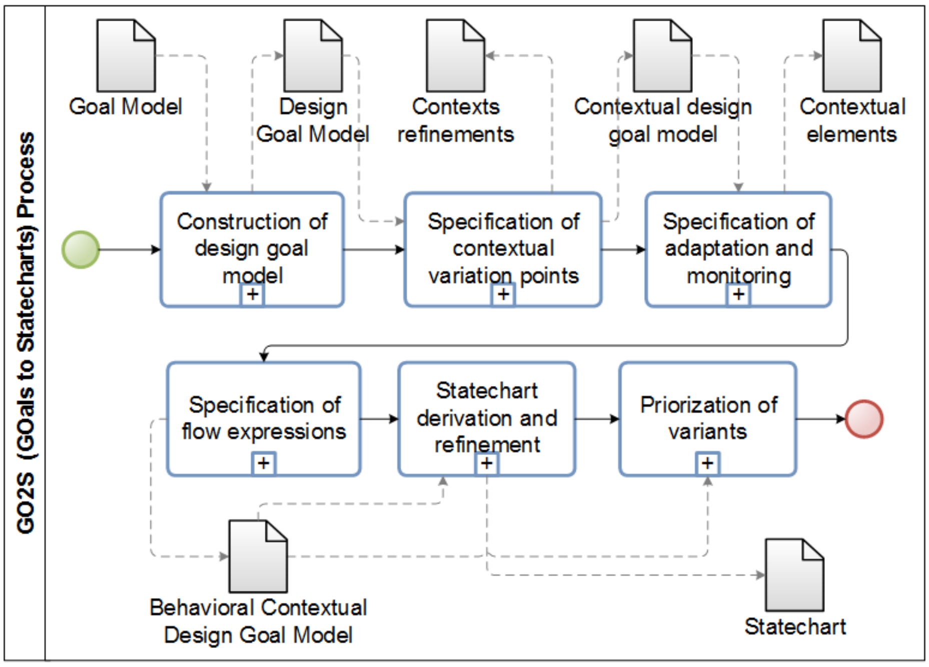

GO2S (GOals to Statecharts) Process
Goal Model
Design Goal Model
Contexts refinements
Contextual design goal model
Contextual elements
Construction of design goal model
Specification of contextual variation points
Specification of adaptation and monitoring
Specification of flow expressions
Statechart derivation and refinement
Priorization of variants
Behavioral Contextual Design Goal Model
Statechart

Fig. 2. GO2S process for deriving statecharts from goal models.

3.1 Construction of design goal model

In the GO2S process, we are concerned with the definition of the behavior of context-sensitive systems. We assume that requirements elicitation and analysis activities were previously performed and a goal model was generated. Hence, the first step (see Fig. 2) consists of the construction of design goal model and receives a goal model as input.

In order to illustrate our process, we consider the popular meeting scheduler system, which is based on the requirements described in [6]. Its purpose is to support the organization of meetings, that is, to determine, for each meeting request, a meeting date and location so that most of the intended participants can effectively participate. Its design goal model, which encompasses both requirements and design elements, is shown in Fig. 3.

Note that besides the traditional goal models elements, the design goal model includes design tasks and design constraints [7], represented through dashed borders in a goal model. This differentiation is used to emphasize the phase of the software development they appear – while requirements elements describe the stakeholders' needs, design ones express a possible way to fulfill those needs. By including these elements in a goal model, it is possible to make use of the existing goal reasoning infrastructure when designing systems with specific needs like context-sensitive.

NFRs are one source of design tasks and constraints. Therefore, it is important to consider their impact in the system, since they change or complement both the structural and behavioral aspects of the system architecture [8]. Therefore, in this first activity, the software engineer should check if there are any relevant NFRs in the system. For example, we can establish the relationship to the goal model using techniques for NFR analysis such as Softgoal Interdependency Graphic (SIG). If a NFR needs to be operationalized, a design task must be included in the goal model. Further, design constraints may also need to be included.

In the meeting scheduler system, we have the usability, performance and security NFRs. To satisfy the security NFR, it was decided to perform access management, so a new functionality should be added to satisfy this requirement. This is expressed by the *Manage Access* design task (see Fig. 3).

Moreover, the design goal model also allows the definition of assignments for its different elements [7]. In our running example, the *Contact Participants* design task (see task t10 in Figure 3) may be performed either by the meeting organizer or by a secretary, which is expressed through an annotation in that task. All the other tasks are either performed or assisted by the system-to-be. The output of this activity is the design goal model which could have the operationalization of NFRs. Next, we need to consider the variability of the model.

3.2 Specification of contextual variation points

In this activity, the contextual variation points are annotated in the design goal model to visually specify the effects of context in the system's behavior. Accordingly, the software engineer has to define the points of the design goal model that are context-dependent. In this work, we considered that contexts can be associated with the following links in a goal model: or/and refinements and contribution to softgoals (see Section 2.1).

When a contextual variation point is identified, the variants at the design goal model are labelled with C1…Cn and annotated in the model, as shown in Fig. 3. We follow the notation of [4] that considers that each context specified in the contextual design goal model must be refined through a set of statements and facts. As an example of context refinement, see the refinement of context C4 related to *Collect timetables by email* task in Fig. 1. The contextual refinements are required in order to allow the system be able to check the validity of context at runtime. Hence, if a context is true, the variant is enabled.

The outputs of this activity are the contextual design goal model and the context refinements. Next, we need to consider how the monitoring and adaptation will be performed.

3.3 Specification of monitoring and adaptation

Context-sensitive systems must monitor the context at runtime in order to decide which variant will be adopted. This can be done by the monitoring of Contextual Elements (CE). These elements can be defined as data or information in the domain whose instantiated values influence the truth values of facts. Each CE can be identified as regards its frequency or periodicity and classified as static or dynamic. Static CE indicates information that is, in general, fixed or does not change very often (e.g. user's personal data - date of birth, number of rooms in the meeting scheduler system). Dynamic CE changes almost instantly, hence it needs to be constantly monitored and updated (e.g. physical location of a person, participants' agenda, number of date conflicts in a meeting request). The dynamic elements are important to specification of context monitoring.

The context will be activated when some change in the CEs occurs. Therefore, we must create a new design task, called *Monitor Context* for example, to represent their monitoring. Then, for each dynamic contextual element, a new design task must be created, expressing the need to monitor it. These tasks can have the form of *Monitor [contextual element]*.

In this excerpt of the meeting scheduler system, we identified three CEs so far: meeting date, timetables responses, and participants' agenda. Thus, the tasks *Monitor meeting date, Monitor Participants agenda and Monitor Timetables responses*, in Fig. 3 are the ones needed to monitor their CEs.

Among the benefits enabled by context-sensitive systems is the possibility of adaptation. Therefore, we propose to use this characteristic to deal with the requirements adaptation when a goal fails. In order to achieve this, we add adaptation design tasks in the design goal model to represent the adaptation required for each requirement the software engineer wants to monitor. These adaptation design tasks will be activated when the associated context holds. The contextual information can be monitored at runtime to indicate the possible situations where the goal has failed.

We propose to add a new design task in the root node for adaptation management and design tasks in this new node for each critical requirement that must be monitored and adapted. Finally, adaptation design tasks should be added to represent the adaptation actions.

In our running example, the software engineer decided that the system has to adapt itself when the *Performance* softgoal and the *Schedule Defined* goal fail. Therefore, two adaptation design tasks were added to our running example: *Manage Performance adaptation* and *Manage Schedule adaptation*. The adaptation tasks are *Step Back* and *Reconfigure Schedule* for *Manage Schedule adaptation* design task; otherwise, *Delegate (Software Architect*) and *Add new server* are the adaptation tasks for the adaptation *Manage Performance adaptation* design task. These elements are represented in Fig. 3.

Besides adding the adaptation design tasks in the design goal model, it is also necessary to refine their contexts (see Section 2.1). In Fig. 3, C1-C5 are contexts previously identified, otherwise, C6-C11 are the ones related to the requirements adaptation of our running example identified in this activity. These contexts are required so adaptation design tasks presented above can be executed.

After all contexts, that influence the requirements are refined, and the context elements that need to be monitored are identified, the next step is to identify the equipments/technologies needed to monitor these contextual elements. In our running example, the only technology needed is the database; however, it can be of several types like GPS, RFID, camera, different types of sensors (presence, humidity, light), etc. These information can be listed in a table for example in order facilitate the visualization and management by the software engineer.

After defining the adaptation design tasks, we identified two more CE (response time and number of conflicts) in the meeting scheduler example. Therefore, the design tasks *Monitor number of conflicts* and *Monitor response time* were also added to Fig. 3.

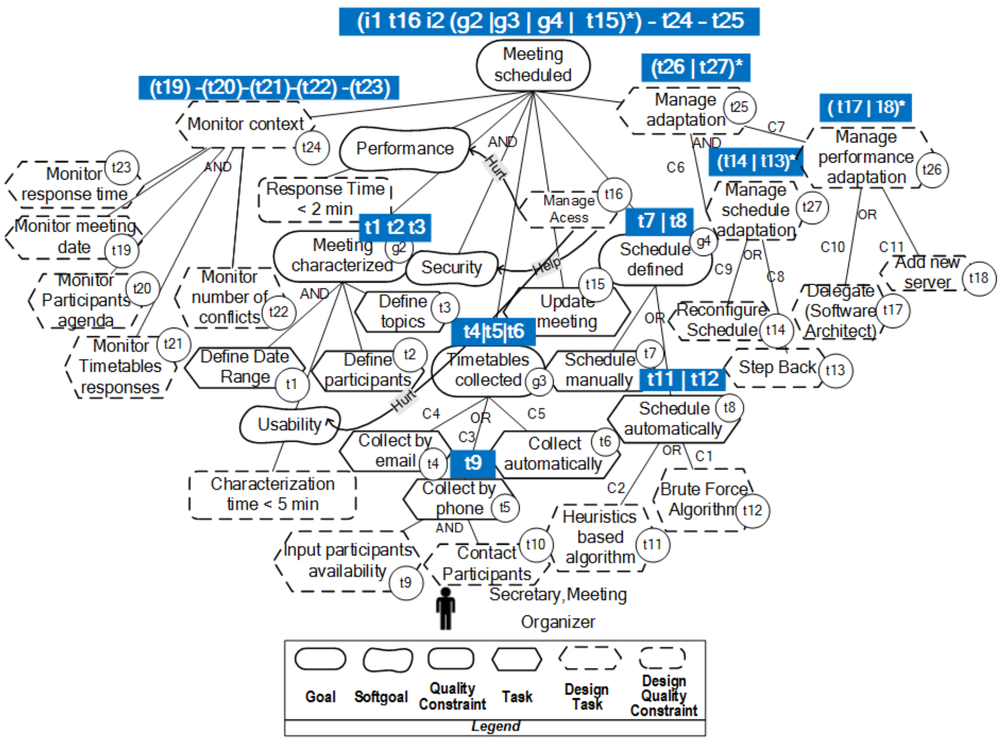


Fig. 3. Behavioral contextual design goal model of the meeting scheduler system.

3.4 Specification of flow expressions

Flow expressions are a set of enrichments to a goal model that allow (restricted) specification of the runtime behavior of a system in terms of how different system tasks and goals interact with each other. We adopted the symbols proposed by [7] with the purpose of facilitating their writing. They are used in our process as intermediary model in order to derive the statechart.

The first step of the specification of flow expressions is to assign an identification to each goal and task in the model, starting with the root goal. G_i was used as the identification for goals and t_i for tasks and design tasks where *i* is the number of the task.

After the assignment, we must visit each parent node, starting with the parents of leafs nodes, to define a flow expression, which describes the behavior of its children elements using the symbols proposed by [7]. Thereafter, the flow expressions are propagated to the parents nodes in the upper level combining with its children elements, and so on. When we reach the root goal, we have the flow expression from the entire system. The resulting flow expressions are annotated in the contextual design goal model.

A common practice when creating statecharts is to use intermediate states as a point where the system is idle, waiting for some input, e.g., waiting for a selection by the user. Considering how frequently these states appear, and aiming to reduce visual pollution in the behavioural contextual design goal model, such states must be inserted directly in the flow expressions identified as iX, where X is an integer.

The result flow expression, presented in Fig. 3 of our running example is (*i1 t16 i2 (g2 |g3 | g4 | t15)*) - t24 - t25*). Thus, from the *idle state* (i1), the system executes *Manage Access* (t16) task, entering in an *idle state* (i2). The *Meeting Characterized* (g2), *Timetables collected* (g3), *Schedule defined* (g4) goals, and the *Update meeting* (t15) task are alternatives that can be executed zero or more times. Besides, *Monitor Context* (t24) and *Manage Adaptation* (t25) design tasks are running concurrently with all tasks.

3.5 Statechart derivation and refinement

The statechart derivation is the last activity of our process. The flow expressions previously defined will be translated into states of the statechart that represents the system's behavior view. We adopted the set of derivation patterns related to the different flows that may be expressed (sequential, alternative and concurrent) as well as to their optionality and multiplicity defined by [7].

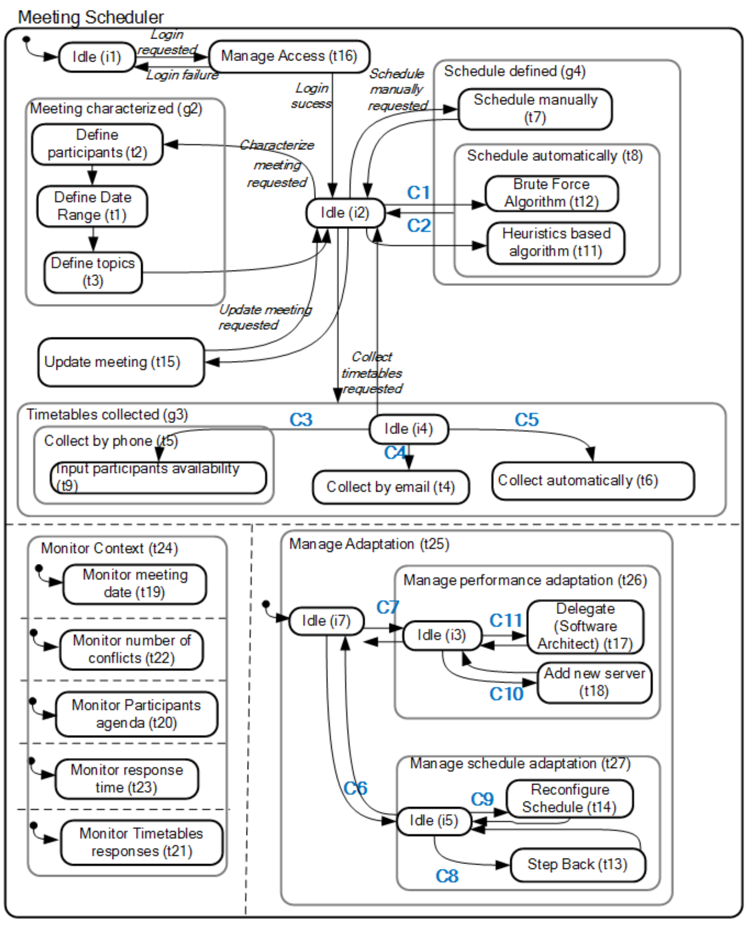
After generating the base statechart, we must specify its transitions in terms of their triggers and conditions. Any event can be used as a trigger, but there are five particular classes of events that are likely to appear in a statechart [7]: user request, timer, requested by another task, requested by another system and context activation. Fig. 4 presents the complete statechart of our running example. The context activation is represented in the statechart through the context labels (C1, C2… Cn) annotated in the behavioral contextual design goal model (Fig. 3).

Given that it is possible that several variants may be enable in certain contexts it is necessary to determine the best option. The prioritization is explained in the next section.

3.6 Prioritization of variants

The system's variants are applicable only if their associated contexts hold. However, in a certain execution, more than one variant may be applicable in the actual context.

For example, suppose that contexts C3, C4, and C5 of Fig. 3 hold. Hence, the system has to implement runtime mechanisms to decide which variant to adopt.



Meeting Scheduler
Idle (i1)
Login requested
Login failure
Manage Access (t16)
Login sucess
Schedule manually requested
Schedule defined (g4)
Schedule manually (t7)
Schedule automatically (t8)
Brute Force Algorithm (t12)
Heuristics based algorithm (t11)
C1
C2
Meeting characterized (g2)
Define participants (t2)
Define Date Range (t1)
Define topics (t3)
Characterize meeting requested
Idle (i2)
Update meeting requested
Update meeting (t15)
Collect timetables requested
Timetables collected (g3)
Collect by phone (t5)
Input participants availability (t9)
C3
Idle (i4)
C4
C5
Collect by email (t4)
Collect automatically (t6)
Monitor Context (t24)
Monitor meeting date (t19)
Monitor number of conflicts (t22)
Monitor Participants agenda (t20)
Monitor response time (t23)
Monitor Timetables responses (t21)
Manage Adaptation (t25)
Manage performance adaptation (t26)
Idle (i7)
C7
Idle (i3)
C11
Delegate (Software Architect) (t17)
Add new server (t18)
C10
C6
Manage schedule adaptation (t27)
C9
Reconfigure Schedule (t14)
Idle (i5)
Step Back (t13)
C8

Fig. 4. Statechart of the meeting scheduler system.

In this work, we are concerned with the impact of NFRs in the system's behavior, so, we rely on the variant contribution to the NFRs satisfaction. In order to achieve this, the software engineer should use a method to prioritize the variants considering the NFRs such as the Analytical Hierarchy Process (AHP) method [10].

The AHP method is used in the GO2S process to produce a ranking of variants (alternatives) that most contributes for the satisfaction of NFRs (criteria). First, it is necessary to establish priorities for the main criteria by judging them in pairs for their relative importance, thus generating a pairwise comparison matrix. Judgments which are represented by numbers from the fundamental scale are used to make the comparisons.

The number of judgments needed for a particular matrix of order n, the number of elements being compared, is n(n - 1)/2 because it is reciprocal and the diagonal elements are equal to unity [10].

In our running example, we have three NFRs (usability, security, performance). As demonstration of the application of this method for variant prioritization, consider that the contexts related to var3 (collect by phone), var4 (collect by email) var5 (collect automatically) hold. Thus, we must follow the steps required to use the AHP described in [10].

Considering the following pairwise comparisons Usability and Performance=0.14; Usability and Security=0.20; Performance and Security=3; the following priority vector is obtained [Usability = 0.074, Performance = 0.643 and Security = 0.283]. Thus, we notice that Performance is the NFR more critical to the software engineer followed by Security and Usability.

The next step is to decide the variants that will be analyzed through this method. For example, suppose that contexts C3 (*Collect by phone*), C4 (*Collect by email*), and C5 (*Collect automatically*) of Fig. 3 hold. Therefore, we list the contribution of each variant to the satisfaction of each NFR according the mapping proposed by [11] to convert from the NFRs contribution to the scale used in AHP. The remaining steps of the method are applied and the variant priority vector [var3=0.14, var4=0.19, and var5=0.67] is obtained in our running example. We can notice that the var5 is the one that contributes mostly for the satisfaction of the NFRs followed by var3 and var4. Therefore, if their context associated hold at runtime, var5 will be chosen by the system. The software engineer must perform this analysis for each variation point when more than one context can hold at runtime.

4 DISCUSSION

This paper proposes the GO2S process for deriving the behavior of context-sensitive systems from a contextual goal model. In comparison with the work of [7], the main difference is that we address the system's context, the operationalization of the NFRs and prioritization of variants.

The prioritization was performed using the AHP method. This activity is useful for selecting which variant the system must adopt at runtime when more than one variant is enabled at the same time. The variant that will be executed is the one that mostly contributes for the satisfaction of the NFRs more critical to the software engineer. It should be noted that AHP analysis can be performed using a spreadsheet tool, which shows that there is no need for sophisticated tool to support this method.

We adopted the AHP method because of its benefits are well described in the literature [12]: it is a well-known and accepted method; it is appropriate for handling conflicting concerns problems; it has the ability to quantify subjective judgments. Moreover, it is capable of comparing alternatives in relation to established criteria; and it provides means to guarantee the logical consistency of the judgments.

The AHP has proven to be an effective method for prioritizing objectives. In industrial projects, this method has been experienced as being effective, accurate and also to yield informative and trustworthy results [13]. However, since all unique pairs must be compared, the required effort can be substantial.

We performed a thorough experimentation in order to evaluate and improve our process. Such experiment was being defined using the framework proposed by [14] for performing experiments in software engineering. The experiment results [15] show that the structural complexity of the group that used our GO2S approach was lower and the mean of behavioral similarity and the time spent was higher than control group. Besides, the subjects agreed that the GO2S process is easy to use indicating that it is possible to reproduce the process and it is understandable.

An experiment to study the scalability of statechart generation algorithm was previously conducted by [7]. The inputs of the simulation were five flow expressions with all possible operators and different number of elements (100, 300, 500, 700, and 900). The results demonstrated that the automatic derivation of statecharts from design goal models is feasible even for large models.

Besides, the contextual design goal model captures the inherent variability of the design space, through the definition of alternative refinements for the same design element. Thus, different solutions (statecharts) for a given problem can be devised.

It is important to note that the monitoring required to assess the context may have a significant impact on the system under development. The monitoring of the context often consumes many application resources and has the tendency to decrease the system's performance. Thus, the impact of monitoring the context data must also be taken in consideration when defining the context annotations.

5 Related works

A process for generating complementary design views from a goal model with high variability in configurations, behavioral specifications, architectural and business processes is presented in [9]. It defines heuristic rules and patterns to map a goal hierarchy into an isomorphic state hierarchy in a statechart. However, their approach does not support the development of context-sensitive systems, neither takes in consideration of the impact of NFRs in the system's behavior and the specification of monitoring and adaptation tasks as supported in our wok.

A process for deriving behavioral models from goal models was also proposed in [7]. The behavioral models, expressed as statecharts, are obtained through a series of refinements expressed within an extended design goal model that constitutes an intermediary model between requirements and architecture. However, in [7] the assumption is that the system operation is independent of context. Unfortunately, this is not always the case. Besides considering the system's context, our work addresses the requirements adaptation modelling, the operationalization of the NFRs and variants prioritization.

The STREAM-A (Strategy for Transition between Requirements and Architectural Models for Adaptive systems) approach [1612] uses goal models based on i* (istar) framework to support the design and evolution of systems that require adaptability. It comprises the enrichment of the requirements model with contextual annotations and the identification of the data that the system will have to monitor. However, it focuses only on the structure of a system architecture. Hence, an important difference of our work is to specify and analyze the intended behavior of systems before they are fully implemented.

An integrated approach to assist the design of Context-sensitive systems (CSS) is presented in [17]. Their approach includes a context metamodel for representing structural and behavioral aspects on CSS. In order to support the modeling of behavioral concepts, the authors propose the use of a profile to model the application behavior using the UML activity diagram with the semantics defined in the Contextual Graphs (CxG). Moreover, an activity diagram is a special case of a statechart [18] in which states are activities ("functions"). Hence, an activity diagram illustrates the flow from activity to activity and clarifies the sequence of actions. Statecharts, otherwise, are a powerful graphical notation to describe reactive systems that support concurrency and hierarchy of states, and allow tracing the behavior given specific inputs.

6 Conclusions and Future Works

In this work, we proposed a systematic process for deriving the behavior of context-sensitive systems, expressed as statechart, from a contextual goal model. The process consist of six activities to guide the software engineer.

The first step of the process concerns the construction of design goal model. It is followed by the definition of context annotations. In the third step, the tasks required for the monitoring and adaptation activities are specified. Later, the system behavior is represented in flow expression in the fourth step. The next one derives a statechart from the behavioral contextual design goal model. Finally, the last step is the prioritization of variants.

We performed a controlled experiment in order to evaluate our process. The experiment results [15] show that the structural complexity of the group that used our GO2S approach was lower and the mean of behavioral similarity and the time spent was higher than control group. Besides, the subjects agreed that the GO2S process is easy to use indicating that it is possible to reproduce the process and it is understandable.

As future work, we expect to develop tool support for our process. Such tool shall support the modelling of the goal model and guide the software engineer to apply our process generating the different views (design, contextual and behavioral) of our process to implement the statechart derivation. Further studies are required in order to develop mechanisms to perform the reasoning of properties such as system's completeness and correctness of context-sensitive systems from the generated statecharts.

Acknowledgments

The following Brazilian institutions have supported this work: FACEPE, CAPES and CNPQ.

7 References

1. Bachmann, F., et al. 2011. Documenting Software Architectures: Views and Beyond. Pearson, USA.
2. Harel, D. Statecharts: A visual formalism for complex systems. 1987. In Science of computer programming, 8, 3, 231-274.
3. Castro, J., Kolp, M., Mylopoulos, J. 2002. Towards requirements-driven information systems engineering: the Tropos project. Information systems, 27, 6, 365-389.
4. Ali, R., Dalpiaz, F. and Giorgini, P. 2010. A goal-based framework for contextual requirements modeling and analysis. Requirements Engineering, 15, 4, 439-458.
5. Nuseibeh, Bashar. 2001. Weaving together requirements and architectures. In Computer, 34, 3, 115-119.
6. Lamsweerde, A.V., R. Darimont, and P. Massonet. 1995. Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In Proceedings of the Second IEEE International Symposium on Requirements Engineering.
7. Pimentel, J., Castro, J., Mylopoulos, J., Angelopoulos, K., Souza, V. E. S. From requirements to statecharts via design refinement. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14), pp. 995-1000, 2014.
8. Yi L. Zhiyi Ma; Weizhong S. 2010. Integrating Non-functional Requirement Modeling into Model Driven Development Method. In 17th APSEC, 98-107.
9. Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J. C. S. P. 2008. From goals to high-variability software design. Foundations of Intelligent Systems, Springer Berlin Heidelberg, 1-16.
10. Saaty, R.W. 1987. The analytic hierarchy process—what it is and how it is used. Mathematical Modelling, 9, 3–5, 161-176.
11. Santos, E. B. 2013. Business Process Configuration with NFRs and Context-Awareness. Thesis (Ph.D. in Computer Science), UFPE, Brazil.
12. Brito, Isabel Sofia et al. Handling conflicts in aspectual requirements compositions. In: Transactions on aspect-oriented software development III. Springer Berlin Heidelberg, 2007. p. 144-166.
13. Karlsson, J. Software requirements prioritizing. In Proceedings of the Second International Conference on Requirements Engineering, pp.110,116, 1996.
14. Wohlin, C., et al. 2012. Experimentation in software engineering. Springer.
15. J. F. F. Vilela. MSC Dissertation - Federal University of Pernambuco, Centers of Informatics, 2015.
16. Pimentel, J. et al. 2012. Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach. Requirements Engineering Journal, 17, 4, 259-281.
17. Vieira, V., Tedesco, P. and Salgado, A. C. 2011. Designing context-sensitive systems: An integrated approach. In Expert Systems with Applications, 38, 2, 1119-1138.
18. Gogolla, Martin; Kobryn, Cris (Ed.). UML 2001-The Unified Modeling Language. Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5. Proceedings (vol. 2185). Springer Science & Business Media, 2001.