Deriving the behavior of context-sensitive systems from contextual goal models

Jéssyka Vilela, Jaelson Castro, João Pimentel,
Monique Soares, Paulo Cavalcanti
Universidade Federal de Pernambuco
Recife, Brazil
{jffv, jbc, jhcp, mcs4, plc2}@cin.ufpe.br

Márcia Lucena
Universidade Federal do Rio Grande do Norte
Natal, Brazil
marciaj@dimap.ufrn.br

ABSTRACT
Context-sensitive systems are flexible, capable of acting autonomously on behalf of users and to dynamically adapt their behavior. This work proposes a systematic process to derive the behavior of context-sensitive systems from contextual goal models considering the impact of non-functional requirements (NFRs). This process is centered on the incremental refinement of a goal model, obtaining different views of the system (design, contextual, and behavioral). A key contribution of this work is the *Behavioral Contextual Design Goal Model*, which depicts in a single artefact the operationalization of NFRs, the adaptation and the monitoring tasks.

Categories and Subject Descriptors
D.2.1 [**Requirements/Specifications**]: Methodologies.

General Terms
Documentation, Design.

Keywords
Behavior, Context-sensitive systems, Contextual goal model.

1. INTRODUCTION
Context-sensitive systems are flexible and capable of acting autonomously on behalf of their users and to dynamically adapt their behavior. Given their complexity, it is of paramount importance to specify and analyze its intended behavior before they are fully implemented. The behavioral specification can be analyzed to explore the range of possible orderings of interactions, opportunities for concurrency, and time-based interaction dependencies among system elements [1].

In this work, we propose an approach to obtain the behavior of context-sensitive systems (expressed as statecharts) from requirements (described as goal models). The benefits are manifold [1]: the models can be used as a communication channel among stakeholders during system-development activities; confidence improvement that the context-sensitive system fulfills its goals and quality attributes; ability to support reasoning to prove some system´s properties, such completeness and correctness. Besides, since the resulting models are pure statecharts, no extensions are required, 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 a popular running example, the meeting scheduler system. Some related works are discussed in Section 4. Finally, we conclude and present our future works in Section 5.

SAC'15 April 13 - 17 2015, Salamanca, Spain Copyright 2015 ACM 978-1-4503-3196-8/15/04$15.00
http://dx.doi.org/10.1145/2695664.2695981

2. BACKGROUND
The systematic process proposed in this paper consists of an incremental refinement of a goal model, towards a statechart. The following sub-sections provide a brief overview of these concepts.

2.1 Contextual Goal Model
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. The notation used in this paper to represent contexts in goal models is based on [5]. Thus, contexts in our work can be associated to 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.
- 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 to allow it to be checked. The contextual refinement has a tree-like structure (Figure 1) in which the root of this model is the context, and statements and facts are its nodes [5]. Statements cannot be verified directly in a context, e.g. *"the meeting is not urgent"*, and are subjective assertions and do not have clear criteria to be evaluated against. Facts, on the other hand, are predicates which truth-values can be verified in a given context, e.g., "*the meeting date is more two days away"*. In order to obtain a verifiable context, all statements are refined into facts and sub-statements, until only facts are left. Figure 1 presents the refinement of 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*).

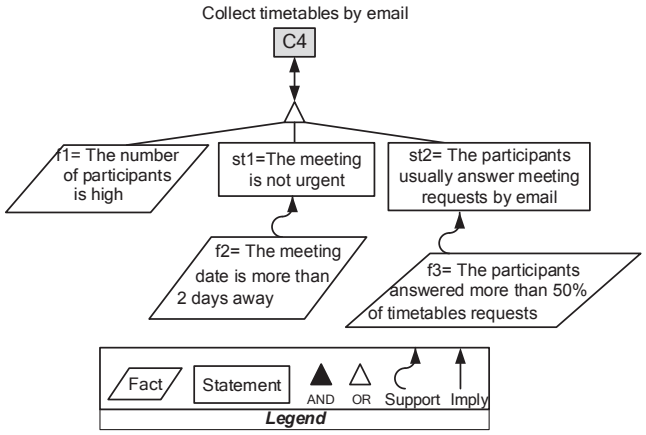

Figure 1. Refinement of the context *Collect timetables by email.*

2.2 Statecharts

Statecharts [2] are a popular visual formalism for modelling reactive systems. Therefore, 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. PROCESS

We propose a systematic process for deriving the behavior (modeled as statecharts) from contextual goal models of context-sensitive systems. This is an iterative process that produces progressively more detailed requirements and design specifications and comprises 6 activities presented in the following sub-sections.

3.1 Construction of design goal model

The first step of our proposal receives as input the requirements goal model and aims to construct its design goal model that includes design tasks and design constraints [3]. These elements are represented through dashed borders and emphasize the phase of the software development on which they appear. This model also allows the definition of assignments to indicate that some user is assigned to execute it, instead of the system itself.

NFRs are one of the sources for design tasks and constraints. Therefore, if a NFR needs to be operationalized, a design task must be included in the goal model.

The design goal model of a meeting scheduler system is shown in Figure 2. It encompasses both requirements and design elements. In this example, we have three NFRs. In order to satisfy the *security* NFR it was decided to perform access management, so the *Manage Access* design task (Figure 2) was added to satisfy this requirement. Besides, the *Contact Participants* design task may be performed either by the meeting organizer or by a secretary, which is expressed through an annotation in this task.

The outputs of this activity are the operationalization of NFRs and the design goal model. Next, we need to consider the variability.

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. Recall that contexts can be associated to the following links of 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, e.g. C1…Cn, and annotated in the model, as shown in Figure 2. We adopt the framework from [5] that considers that each context specified in the contextual design goal model must be refined through a set of statements and facts. The contextual refinements are required in order to allow the system to 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 will be performed and how to enable the required adaptation.

3.3 Specification of monitoring and adaptation

Among the benefits allowed by context-sensitive systems is the possibility of adaptation. We propose the following steps to define the adaptation and the monitoring of these systems:

(1) Define the requirements that requires an action in case of failure;

(2) Represent the adaptation management:

   a. Add a new design task in the root node for adaptation management;

   b. Add design tasks in the parent node previously created for the management of each requirement that must be monitored and adapted;

   c. Add design tasks to represent the adaptation actions for each monitored requirement;

(3) Associate each adaptation design task to a context label;

(4) Refine each context;

(5) Identify the dynamic contextual elements;

(6) Represent the context monitoring:

   a. Add a new design task in the root node;

   b. Add design tasks to monitor each dynamic contextual element;

(7) Specify the technology necessary to monitor the contexts.

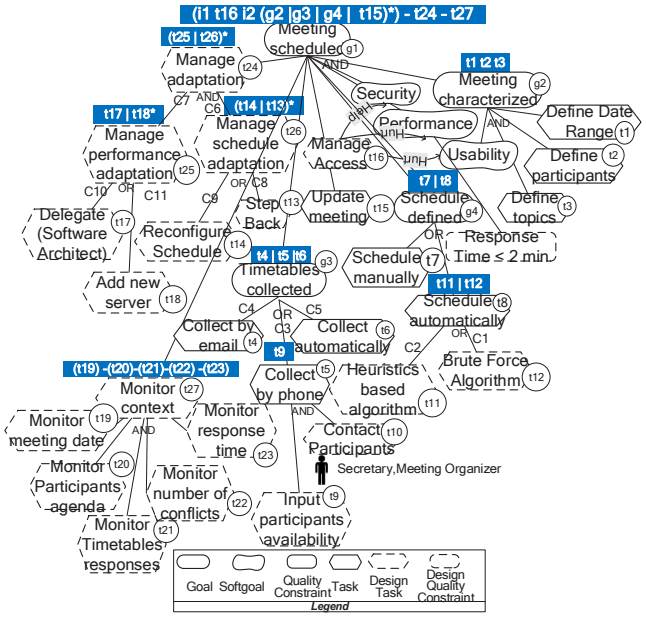

Figure 2. Excerpt of the behavioral contextual design goal model of the meeting scheduler system.

In our running example, the analyst decides that the system has to adapt itself when the *Performance* softgoal and the *Schedule Defined (g4)* goal fail. Its adaptation tasks are *Step Back* (t13) and *Reconfigure Schedule* (t14) for the *Manage Schedule adaptation (t26)* adaptation design task; otherwise, *Delegate (Software Architect)* (t17) and *Add new server* (t18) are the tasks for the *Manage Performance adaptation* (t25) adaptation design task. Figure 2 shows these adaptation design tasks, the context annotations as well as the tasks responsible for monitoring the contextual elements.

The technology required for the monitoring of the contextual elements (such as GPS, RFID, camera, different types of sensors) for example can be listed in a table to facilitate the visualization and management by the Requirements Engineer.

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 [3]. Its specification consists of the assignment of an identification for each goal and task in the model and the assignment to each parent node a flow expression that describes the behavior of its children elements using the symbols proposed by [3]. When we reach the root goal, we have the flow expression from the entire system.

A common practice when creating statecharts is to use intermediate states as a point where the system is idle, waiting for some input or waiting for a selection by the user. Such states should be inserted directly in the flow expressions identified as iX, where X is an integer.

Note in Figure 2 the flow expression of our running example: (*i1 t16 i2 (g2 | g3 | g4 | t15)*) - t24 – t27*. Thus, from the *idle state (i1),* the system executes the *Manage Access* (*t16*) state, entering an *idle state (i2*). *Meeting Characterized* (*g2*), *Timetables collected (g3), Schedule defined (g4), Update meeting (t15*) are alternatives states that can be executed zero or more times. Besides, the *Manage Adaptation (t24) and Monitor Context* (*t27*) are states running concurrently with all states.

3.5 Statechart derivation and refinement

The statechart derivation is the last activity of our process. 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 [3].

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 [3]: user request, timer, requested by another task, requested by another system and context activation. Figure 3 presents the complete statechart of our running example.

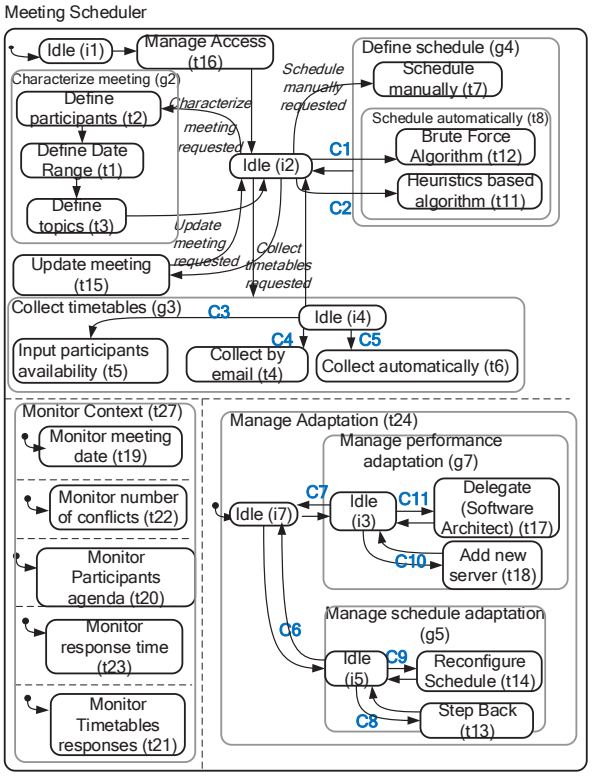

Figure 3. Excerpt of the statechart of the meeting scheduler system.

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

3.6 Prioritization of variants

Given that we are concerned with the impact of NFRs in the system's behavior we must consider the variant contribution to their satisfaction. Moreover, we use a prioritization method to choose the most critical variant. In this work we relied on Analytical Hierarchy Process (AHP) method [6]), which requires the performance of a pairwise comparisons between the NFRs in order to obtain its priority vector. The next step is to decide the variants that will be analyzed and their contribution to the satisfaction of each NFR. According the mapping proposed by [6] it is necessary to convert from the NFRs contribution to the scale used in AHP. After applying the remaining steps, the variant priority vector is obtained. Note that the Requirements Engineer

must perform this analysis for each variation point when more than one context can hold at runtime.

4. RELATED WORKS

The process presented in [9] generates complementary design views from a goal model with high variability in configurations, behavioral specifications, architectural and business. 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 systematic process for deriving behavioral models from goal models was also proposed [3]. 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 [3] the assumption is that the system operation is independent of context. Unfortunately, this may 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 [10] relies on goal models based on i* (istar) framework to support the design and evolution of systems that require adaptability. It comprises of the enrichment of the requirements model with contextual annotations and the identification of the data that the system 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 [11]. 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. However, an activity diagram is a special case of a statechart. Statecharts, otherwise, are a powerful graphical notation to describe reactive systems that allow tracing the behavior given specific inputs.

5. CONCLUSIONS AND FUTURE WORKS

In this work, we proposed a systematic process for deriving the behavior of context-sensitive systems, expressed as statechart, from contextual goal models. A key contribution of this work is that we present the operationalization of NFRs, the adaptation and monitoring tasks in the same contextual design goal model and also deal with the prioritization of variants. Therefore, our novel proposal does not need any additional notation for adaptation modelling; it uses the elements of a contextual design goal model, allowing the analyst to visualize its impact on the system's behavior.

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.

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 analyst to apply our process generating the different views (design, contextual and behavioral) to implement the statechart derivation. Finally, we acknowledge that a thorough experimentation must be performed in order to evaluate and improve our process. Such experiment is being defined using the framework proposed by [8] for performing experiments in software engineering.

6. 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] Pimentel, J., Castro, J., Mylopoulos, J., Angelopoulos, K, Souza, V. E. S. In *Proceedings of the Annual ACM Symposium on Applied Computing,* 995-1000.

[4] Castro, J., Kolp, M., Mylopoulos, J. 2002. Towards requirements-driven information systems engineering: the Tropos project. *Information systems,* 27, 6, 365-389.

[5] Ali, R., Dalpiaz, F. and Giorgini, P. 2010. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering,* 15, 4, 439-458.

[6] Saaty, R.W. 1987. The analytic hierarchy process—what it is and how it is used. *Mathematical Modelling*, 9, 3–5, 161-176.

[7] Santos, E. B. 2013. Business Process Configuration with NFRs and Context-Awareness. Thesis (Ph.D. in Computer Science), UFPE, Brazil.

[8] Wohlin, C., et al. 2012. Experimentation in software engineering. *Springer.*

[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] 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.

[11] 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.