Pós-Graduação em Ciência da Computação

"GO2S: A SYSTEMATIC PROCESS TO DERIVE THE BEHAVIOR OF CONTEXT-SENSITIVE SYSTEMS FROM REQUIREMENTS MODELS"

By

Jéssyka Flavyanne Ferreira Vilela

Master Dissertation

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, 2015

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JÉSSYKA FLAVYANNE FERREIRA VILELA

"GO2S: A SYSTEMATIC PROCESS TO DERIVE THE BEHAVIOR OF CONTEXT-SENSITIVE SYSTEMS FROM REQUIREMENTS MODELS"

A M.SC. DISSERTATION PRESENTED TO THE CENTER OF INFORMATICS OF UNIVERSIDADE FEDERAL DE PERNAMBUCO IN PARTIAL FULfiLLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE.

ADVISOR: PhD Jaelson Freire Brelaz de Castro

RECIFE, 2015

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

699g Vilela, Jéssyka Flavyanne Ferreira
GO2S: a systematic process to derive the behavior of context-sensitive systems from requirements models / Jéssyka Flavyanne Ferreira Vilela. – Recife: O Autor, 2015.
135 f.: il., fig., tab.

Orientador: Jaelson Freire Brelaz de Castro.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da computação, 2015.
Inclui referências e apêndice.

1. Engenharia de software. 2. Engenharia de requisitos. 3. Arquitetura de software. I. Castro, Jaelson Freire Brelaz de (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2015-67

Dissertação de Mestrado apresentada por **Jéssyka Flavyanne Ferreira Vilela** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"GO2S: A Systematic process to derive the behavior of context-sensitive systems from requirements models",** orientada pelo **Prof. Jaelson Freire Brelaz de Castro** aprovada pela Banca Examinadora formada pelos professores:

_____

Profa. Carla Taciana Lima Lourenço Silva Schuenemann
Centro de Informática / UFPE

_____

Prof. Julio Cesar Sampaio do Prado Leite
Departamento de Informática / PUC-Rio

_____

Prof. Jaelson Freire Brelaz de Castro
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 26 de fevereiro de 2015.

_____

Profa. Edna Natividade da Silva Barros
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

I dedicate this work to God,
to my parents Mário Vilela and Fátima Vilela,
to my sister Francynne Vilela
and to my fiancé Marcus Queiroz.

ACKNOWLEDGEMENTS

To GOD, its omnipotence and omnipresence, and all spiritual forces for guidance and support in every moment that I needed, for allowing my academic progress and for including people in my life who encouraged me.

To Fátima and Mário Vilela, my parents, my thanks for everything! Thanks for understand the life I live: always between books and computers. They have always taken care of me and my sister and they dedicated their lives to provide to us the conditions so that we could achieve our goals. I also thank my sister Francynne Vilela. I love you guys.

To my fiancé Marcus Queiroz for all attention, affection, understanding, support and love received over these years.

To my family and friends who have always believed in my capacity to accomplish another mission and accompanied me from near or far.

To Jaelson Castro, my advisor, for the freedom, guidance, opportunities and trust that with his expertise and experience provided to me the opportunity to work on this project.

To my friends of Laboratório de Engenharia de Requisitos (LER) for the friendship, support and sharing of their knowledge and experiences. Especial thanks to João, Jean and Edson.

To the subjects of the controlled experiment for their patience, collaboration and feedback.

To all teachers of Centro de Informática (CIN) that, in one way or another, contributed to my academic and personal growth.

To my examination committee Carla Silva and Júlio Leite for their availability and for all comments to improve this work.

To the Fundação de Amparo à Ciência e Tecnologia de Pernambuco (FACEPE) for the financial support.

To all people who helped me directly or indirectly to the achievement of my goals.

Mesmo que o futuro lhe pareça distante,
ele está começando neste exato momento.

—MATTIE J.T. STEPANEK

RESUMO

Contexto: Sistemas sensíveis ao contexto utilizam contexto com o intuito de se adaptar as necessidades atuais dos usuários ou falha de requisitos. Portanto, eles necessitam adaptar seu comportamento dinamicamente. É de suma importância especificar e analisar o comportamento desejado desses sistemas antes de serem totalmente implementados. A especificação do comportamento pode ser utilizada para validação dos requisitos de forma a verificar se estes sistemas serão capazes de alcançar os seus objetivos. Além disso, o raciocínio sobre propriedades desses sistemas, tais como deadlocks, alcançabilidade, completude e corretude do sistema, pode ser apoiado. Portanto, o comportamento dinâmico dos sistemas sensíveis ao contexto exige uma abordagem para sua especificação a partir de seus requisitos. **Objetivo:** Nesse trabalho é proposto o processo GO2S (Goals to Statecharts) para derivar sistematicamente o comportamento de sistemas sensíveis a contexto, expresso em statecharts, a partir de modelos de requisitos, descritos em modelo de objetivos. O processo considera o impacto dos requisitos não-funcionais desses sistemas através da sua operacionalização e priorização de alternativas a serem utilizadas em tempo de execução (variantes). **Método:** A pesquisa apresentada nessa dissertação seguiu a abordagem de engenharia para definir o processo GO2S e sua adequabilidade foi avaliada empiricamente. **Resultados:** O processo GO2S aborda a especificação das tarefas necessárias para o monitoramento da satisfação dos requisitos bem como a adaptação do sistema de acordo com o contexto, a operacionalização dos requisitos não-funcionais e priorização de variantes. Este é um processo iterativo centrado no refinamento incremental de um modelo de objetivo, obtendo diferentes visões do sistema (design, contextual, comportamental). Além disso, foi realizado um experimento controlado para avaliar os statecharts produzidos seguindo o GO2S (grupo experimental) com aqueles elaborados sem utilizá-lo (grupo de controle). **Conclusões:** Os resultados do experimento mostraram que a complexidade estrutural dos statecharts do grupo que usou a abordagem GO2S foi inferior ao grupo de controle. Além disso, a média do número de funcionalidades cujo comportamento foi modelado de acordo com a especificação e o tempo gasto para produzir os modelos do grupo experimental foram maiores em relação ao grupo de controle. Finalmente, os participantes concordaram que o processo GO2S é fácil de usar.

Palavras-chave: Sistemas sensíveis a contexto. Comportamento. Statecharts. Modelo de Objetivos. Derivação. Experimento Controlado.

ABSTRACT

Context: Context-sensitive systems use context in order to adapt to the user's current needs or requirements failure. Therefore, they need to dynamically adapt their behavior. It is of paramount importance to specify and analyze the intended behavior of these systems before they are fully implemented. The behavioral specification can be used for requirements validation in order to verify if these systems will be able to achieve their goals. Moreover, the reasoning about properties of these systems, such as deadlocks, reachability, completeness and correctness of the system, can be supported. Therefore, the dynamic behavior of the context-sensitive systems requires an approach to specify it from their requirements. **Objective:** In this work, we propose the GOals to Statecharts (GO2S) process to systematically derive the behavior of context-sensitive systems, expressed as statecharts, from requirements models, described as goal models. The process takes into consideration the impact of non-functional requirements of these systems through their operationalization and prioritization of alternatives to be used at runtime (variants). **Method:** The research presented in this dissertation followed the engineering approach to define the GO2S (GOals to Statecharts) process and its suitability was empirically evaluated. **Results:** The GO2S process addresses the specification of the tasks required for monitoring of the requirements satisfaction as well as the system adaptation according to the context, the operationalization of non-functional requirements and prioritization of variants. This is an iterative process centered on the incremental refinement of a goal model, obtaining different views of the system (design, contextual, behavioral). Furthermore, we conducted a controlled experiment to evaluate the statecharts produced following the GO2S (experimental group) in relation to the ones elaborated by not using it (control group). **Conclusions:** The experiment results show that the structural complexity of the statecharts of the group that used the GO2S approach was lower in relation to the control group. Moreover, the average of functionalities whose behavior was modeled according to the specification and the time spent to produce the models of the experimental group were higher in relation to the control group. Besides, the subjects agreed that the GO2S process is easy to use.

Keywords: Context-sensitive Systems. Behavior. Statecharts. Goal Models. Derivation. Controlled Experiment.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

SUMMARY

1 Introduction

In this chapter, we characterize the context of this work and the main motivations and justifications for this dissertation. Then, we present the objectives and the related works. Furthermore, we describe the methodology used to conduct this research. Finally, the work structure is defined.

1.1 Context

Requirements Engineering (RE) is a branch of software engineering that deals with elicitation, refinement, analysis, of software systems requirements. Therefore, it addresses the reasons why a software system is needed, the functionalities it must have to achieve its purpose and the constraints on how the software must be designed and implemented (LAPOUCHNIAN, 2005).

Goal-Oriented Requirements Engineering (GORE) approaches have become quite popular. GORE is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements (LAPOUCHNIAN, 2005).

The benefits of goal modeling are manifold (LAPOUCHNIAN, 2005) (ALI; DALPIAZ; GIORGINI, 2010): goals provide rationale for requirements that operationalize them; goals provide a precise criterion for sufficient completeness of a requirements specification; a single goal model can capture variability in the problem domain through the use of alternative goal refinements; goals provide a natural mechanism for structuring complex requirements documents, and they offer a very intuitive way to elicit and analyze requirements.

Goal models have been used as an effective means to capture the interactions and information-related requirements of adaptive systems and context-sensitive systems (PENSERINI et al., 2007), (MORANDINI et al., 2009), (PIMENTEL et al., 2014), (ALI; DALPIAZ; GIORGINI, 2010). A possible reason is that they incorporate the space of alternatives of a set of operations, i.e. variants, which gives more flexibility to meet stakeholders' goals in a dynamic environment (ALI; DALPIAZ; GIORGINI, 2010).

The requirements models, which describe the problem, need to be related to the solution

space, which often begins with the architectural description. However, a software architecture cannot be described in a simple one-dimensional fashion (CLEMENTS et al., 2002). It can be represented through different views: structural, behavioral, deployment, and configuration. In this work we are concerned with the behavioral view.

Many notations support the description of the system behavior such as Labelled Transition Systems (LTS) (NICOLA, 1987), Petri Nets (MURATA, 1989), and Statechart (HAREL, 1987). Statechart, adopted by Unified Modeling Language (UML) language, is a popular choice for representing the behavioral view of a system (PIMENTEL et al., 2014) (RANJITA; PRAFULLA; DURGA, 2012). Besides, this notation is an interesting visual formalism for modeling context-sensitive systems since they are reactive and adapt its requirements.

Computer systems that use context to provide more relevant services or information are called *Context-Sensitive Systems (CSS)* (ABOWD et al., 1999). CSS enable systems to distil available information into relevant information, to choose relevant actions from a list of possibilities, or to determine the optimal method of information delivery (SANTOS, 2008).

An important feature of CSS is the contextual adaptation (CHALMERS, 2002). In these systems, context can be used to trigger actions (when a certain set of contextual information reaches specific values); or services (tailored according to the limits and preferences imposed by the context). Hence, these systems use context to direct actions and behaviors to support communication between systems and their users. This support can be achieved changing their sequence of actions, the style of interactions and the type of information provided to users in order to adapt to the user's current needs or requirements failure (VIEIRA; TEDESCO; SALGADO, 2011).

Since context-sensitive systems are flexible and capable of reacting on behalf of their users, they need to dynamically adapt their behavior. The benefits of specifying the behavior of context-sensitive are manifold (CLEMENTS et al., 2002): the models can be used as a communication channel among stakeholders during system-development activities; they improve the confidence that the context-sensitive system will be able to achieve its goals. Moreover, the reasoning about systems' properties, such as deadlocks, reachability, completeness and correctness of the system, can be supported in the behavioral models.

1.2 Motivation and Rationale

Enabling computer systems to change their behavior according to the analysis of contextual information is a challenge that attracts the attention of researchers from several areas of computer science (SANTOS, 2008).

CSS must have the following characteristics: monitoring, awareness and adaptability. Hence, developing CSS is a complex and a labor-intensive task. When designing these systems, the software engineer needs to deal with issues associated to: which kind of information consider as context, how to represent this information, how to acquire and process it (considering that it

may come from several and heterogeneous sources) and how to integrate the context usage in the system (SANTOS, 2008).

However, software engineers have difficulties to understand and define what to consider as context and how to design context-sensitive systems. A possible reason is the lack of consensus in the literature regarding the terminology, characteristics and specificities related to context and CSS.

It is worth mentioning that context can influence the requirements of a system, and as a consequence, the choice of the variant (alternative to be executed at runtime) a system can adopt to meet its requirements (ALI; DALPIAZ; GIORGINI, 2010). Hence, there is a need for approaches that guide CSS designers on obtaining their behavior and performing activities related to the system's behavior specification.

It is important to note that Non-Functional Requirements (NFRs) affect both the structural and behavioral aspects of the system (architecture) (LIU; MA; SHAO, 2010). Hence, they need to be operationalized . Moreover, they 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.

Software-development organizations frequently begin their activities with one of these alternative starting points - requirements or architectures - often adopting a waterfall like development process. It is common to artificially freeze the requirements document and release it for use in the next step of the development life cycle. On the other hand, if the development is based on constrained architectures, it may restrict users and handicap developers by resisting inevitable and desirable changes in requirements.

In fact, it is well known that requirements and software architecture are intertwined (NUSEIBEH, 2001). Hence, it is of paramount importance that the architecture should be aligned with requirements.

The inherent variability of CSS requires the analysis of their behavior before they are fully implemented. Therefore, we need approaches to guide the software engineer of CSS to obtain the behavior of context-sensitive systems from requirements models. To the best of our knowledge, does not exist any approach to guide the software engineer to perform this specification.

In particular, we investigated the work of PIMENTEL et al. (2014) about the derivation of behavior from goal models. The authors assume that there is an uniform nature of the context in which the system operates. Unfortunately, this is not always the case.

Our work concerns on the specification of CSS behavior expressed as statecharts, and derived from requirements models, described as goal models. It is out of scope to develop a tool to assist the software engineer in performing such derivation.

The research described in this dissertation is targeted, especially, at designers of CSS, particularly those responsible for knowledge engineering, requirement analysis and architecture design. We envisage a Model-Driven Development (MDD) approach, where models play a key

role throughout the development (MELLOR; CLARK; FUTAGAMI, 2003).

1.3 Objectives

The main research question investigated by this dissertation is: *How can we obtain the behavior of context-sensitive systems from requirements models including their non-functional requirements?*

In order to answer this research question, we define the following specific objectives:

- Proposal of a systematic process for deriving the behavior of context-sensitive systems from requirements models;
- Definition of a systematic approach for the specification of monitoring and adaptation tasks;
- Definition of a metamodel to relate the requirements, architectural design, context and behavior in a unified approach.
- Illustration of the applicability of the process through an example;
- Empirical evaluation of the process through a controlled experiment to evaluate the time to implement, syntactic correctness, structural complexity, behavioral similarity and cognitive complexity of the generated statecharts using (or not) our approach.

1.4 Related Works

To the best of our knowledge, no process regarding the statecharts derivation of CSS from requirements models has been undertaken so far. However, we identified some works that are somehow related with behavior specification.

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 (YU et al., 2008). To this end, the authors employed three complementary design views: a feature model, a statechart and a component model. The process is guided by heuristic rules and patterns to map a goal hierarchy into an isomorphic state hierarchy in a statechart. However, the resulting statecharts of this approach do not support the specification of monitoring and adaptation tasks and the variants prioritization as supported in our work. The specification of these tasks are necessary in the development of context-sensitive systems since they must have three characteristics: monitoring, awareness and adaptation (KLEIN et al., 2008).

The STREAM-A (Strategy for Transition between Requirements and Architectural Models for Adaptive systems) approach (PIMENTEL et al., 2012) 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 not the behavior. It is of paramount importance to specify and analyze the intended behavior of context-sensitive systems before they are fully implemented since context acts like a set of constraints that influence the behavior of a system (a user or a computer) embedded in a given task (BAZIRE; BRÉZILLON, 2005). The behavioral specification is used as input to the analysis which explores the range of possible order of interactions, opportunities for concurrency, and time-based interaction dependencies among system elements. Hence, an important difference of our work is to specify and analyze the intended behavior of these systems.

In a subsequent work, PIMENTEL et al. (2014) proposed a process for deriving behavioral models from goal models. 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, they assume that there is an uniform nature of the context in which the system operates. Unfortunately, this is not always the case. This assumption is not valid in many types of systems, where it is essential to monitor and adapt to an inherently varying context in order to keep the system's goals satisfied. We consider the algorithm for the statecharts derivation developed in this work and adapt it to CSS. Besides considering the modeling of system's context, the monitoring and adaption tasks, we address the operationalization of NFRs. Moreover, given that it is possible that several variants may be enabled in certain contexts, it is necessary to determine the best option. Hence, we propose to perform this prioritization through the contribution of the variants to the satisfaction of the non-functional requirements.

An integrated approach to assist the design of context-sensitive systems is described in VIEIRA; TEDESCO; SALGADO (2011). Their work 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 UML profile to model the application behavior using the UML activity diagram with the semantics defined in the Contextual Graphs (CxG). However, the authors observed in their empirical study that the usage of UML Activity diagram to model the contextual graphs following the CxG profile, caused misunderstandings, since the semantics of the elements of this diagram conflicted with the contextual graph elements semantics. In our work, we use the statecharts to specify the intended behavior of context-sensitive systems since it is a popular choice for representing the behavioral view of a system and it is also adopted by UML language. Besides, this notation is an interesting visual formalism for modeling context-sensitive systems since they are reactive and adapt their requirements. Moreover, an activity diagram is a special case of a statechart. Statecharts, otherwise, are a powerful graphical notation to describe reactive systems that allow nested super-/sub-state structure for abstraction or decomposition. This hierarchical notation of statecharts allows the description of the behavior of context-sensitive systems at different levels of abstraction. This

property of statecharts makes them much more concise and usable than activity diagrams and contributes to a lower structural complexity of the models. Besides, we also concerned with the satisfaction of NFRs, thus we propose an activity in our process that deals with variants prioritization.

1.5 Research Methodology

Research in software engineering can be conducted through four research methods: scientific, engineering, empirical and analytical (GLASS, 1994) (WOHLIN et al., 2012). In the scientific method, the world is observed and then a model or theory is proposed. It is measured and analyzed, the hypotheses of the model or theory are validated, and repeated if possible.

The engineering method consists of the observation of current solutions, followed by the proposal of alternative, the development, and evaluation. In the empirical method, a model is proposed and evaluated through empirical studies, for example, case studies or experiments.

The analytical method consists of proposing a formal theory or set of axioms, developing a theory, deriving results, and if possible comparison with empirical observations.

The research presented in this dissertation followed the engineering and empirical methods. Initially, we performed a literature review about the main areas involved in this work: Self-Adaptive and Context-Sensitive Systems, Requirements Engineering and Goal Oriented Requirements Engineering, Non-Functional Requirements, Model-Driven Development, Software Architecture, Flow Expressions, and Statecharts.

After identifying the open issues in these fields and considering the acquired knowledge, we defined the GOals to Statecharts (GO2S) process for derivation of statecharts from goal models. Given a goal model, we aimed to specify the contextual information and the monitoring and adaptation tasks in a systematic way.

We propose the operationalization of NFRs in the goal model in order to specify its impact on the system behavior. Accordingly, in the GO2S process, this is achieved through the insertion of design tasks in the goal model and not through abstract and subjective attributes. Besides, the NFRs were also considered for the selection of the variant, at runtime, when more than one context hold simultaneously.

We also define the GO2S metamodel that describes the concepts of the behavioral contextual design goal model, their properties and the valid connections between the elements. This metamodel relates the requirements, architectural design, context and behavior annotations in a unified metamodel.

For the description of the GO2S process, the meeting scheduler example, described in next section, was used. In order to illustrate the approach, we considered the Znn exemplar.

Moreover, an empirical method was used to evaluate the GO2S process. We conducted a controlled experiment with eighteen subjects enrolled in a RE course divided in two groups. One group applied the treatment (GO2S process) and the other group (the control group) did not

apply it. The feedback provided by the subjects allowed the improvement of GO2S process.

1.6 Dissertation Structure

Chapter 1 consists of this introduction about the context, motivation and rationale and the objectives of this research. Furthermore, it discusses some related works, the methodology used to conduct this dissertation, and describes the running example that is going to be used to illustrate the proposed process.

Chapter 2 summarizes the basic concepts of context-sensitive systems, goal-oriented requirements engineering, and contextual goal model. Besides, we address the behavior modeling through flow expressions and statecharts. All these concepts are fundamental to understanding the notations adopted and the research carried out.

Chapter 3 explains the GO2S process, its metamodel and discusses its use with a running example.

Chapter 4 illustrates the use of the GO2S. The ZNN exemplar, a well known exemplar of the adaptive systems community, is described in detail.

Chapter 5 presents the design and results of a controlled experiment conducted to evaluate our process.

Chapter 6 discusses the results obtained in this dissertation. Moreover, it makes some final considerations on the development of this work, as well as summarizes the main contributions, scope and limitations found. We indicate some future works that are required to improve our approach.

2

Background

The systematic process proposed in this dissertation consists of an incremental refinement of a goal model, towards a statechart of a context-sensitive system. Our process follows the twin peaks concept (NUSEIBEH, 2001) which separates problem structure and requirements specification from solution structure and architecture specification, producing progressively more detailed requirements and design specifications.

Therefore, we present in this chapter the running example to illustrate our process as well as basic concepts of context-sensitive systems, goal oriented requirements engineering and contextual goal model. Besides, we address the behavior modeling through flow expressions and statecharts. All these concepts are fundamental for the understanding of the research carried out. The following sections provide a brief overview of these concepts.

2.1 Running Example: Meeting Scheduler

We selected a simplified version of the popular meeting scheduler (VAN LAMSWEERDE; DARIMONT; MASSONET, 1995) adapted from (ANGELOPOULOS; SOUZA; MYLOPOULOS, 2014) as a running example in order to demonstrate the activities of the GO2S process. This exemplar is a well known in the RE community, and provides a rich combination of challenging features - e.g., interfering goals, very high-level objectives such as usability, real-time performance constraints, and privacy concerns. Besides, the space of alternative decisions and compromises to be made throughout the process is fairly large (ANGELOPOULOS; SOUZA; MYLOPOULOS, 2014).

The purpose of the meeting scheduler system is to support the organization of meetings (VAN LAMSWEERDE; DARIMONT; MASSONET, 1995). Therefore, the system should determine, for each meeting request, a meeting date and location so that most of the intended participants will effectively participate.

Figure 2.1 presents a goal model of meeting scheduler example adapted from ANGELOPOULOS; SOUZA; MYLOPOULOS (2014). Its main goal is to have a *Meeting Scheduled*. In order to achieve this goal, the system should satisfy the *Meeting characterized*, *Timetables*

collected and the *Schedule defined* goals. Besides, the system has privacy concerns represented by the *Security* softgoal, Real-time performance constraints represented by the *Performance* softgoal and the user should be able to *Update meeting*. These requirements are part of an and-refinement,. Accordingly, all children nodes should be satisfied.

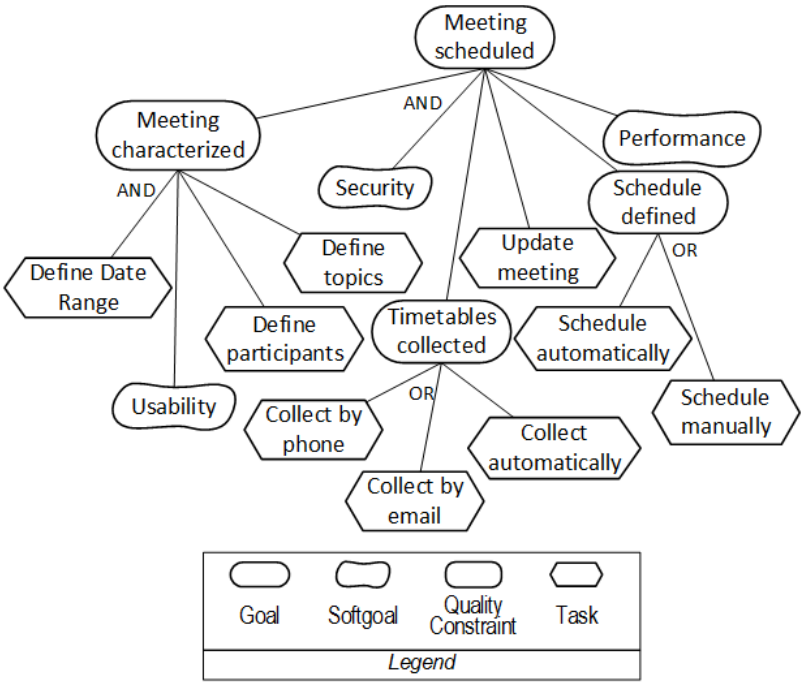


Figure 2.1: Goal Model of meeting scheduler example adapted from ANGELOPOULOS; SOUZA; MYLOPOULOS (2014).

The *Meeting characterized* goal is achieved when the *Usability* softgoal and the tasks *Define date range*, *Define participants* and *Define topics* are satisfied.

The Timetables can be collected through three alternative tasks: *Collect by phone*, *Collect by email* or *Collect automatically*. Finally, the *Schedule defined* goal can be achieved by the *Schedule manually* and *Schedule automatically* tasks.

2.2 Context-Sensitive Systems

People use daily, contextual information to make decisions, make judgments or interact with others. Understanding the context in which there is a certain interaction is essential for individuals to respond appropriately to the situation. Accordingly, context is used in different types of interactions such as "Person-Person", "Human-Computer" and "Computer-Computer" (VIEIRA et al., 2006).

In interaction between people, context improves the quality of conversations and interactions and it helps to solve ambiguities and conflicts since messages sent to communication bring embedded an associated context that supports its understanding. Besides, it helps to understand situations, actions and events. For example, consider that a user is using a computer with many

files and windows open. If someone says "close the window", probably she is talking about a window open in the computer. If someone is not in front of the computer, in a room with a physical window, perhaps she is refering to the window of the room. Moreover, context drives actions and behaviors. For example, when a person is in a football stadium or a movie, she knows that her behavior should be quite different in each of these locations. While in the movies she should turn off the phone, be quiet and remain seated, in the stadium she can scream, jump and stomp with her team and the phone can be used without restrictions (VIEIRA et al., 2006).

Context is also used in "Human-Computer" Interactions expanding the form of communication without the need for explicit user intervention. For example, if a user is in website seraching for a book, the system can search the previous orders of the user, investigate his profile and the people that already bought the book. From this contextual information, the system can make an analysis and provide a list of books that probably the user is interested. Hence, context allows system adaptation by enabling/disabling functionalities, providing services and information relevant to a situation (VIEIRA et al., 2006).

Another application of context is in "Computer-Computer" interactions helping the communication between devices. For example, in a smart home, the system should be able to turn on/off lights, open/close windows and doors and provide another personalized services. In order to perform these tasks, the different devices (sensors, camera, etc) should communicate with each other (VIEIRA et al., 2006).

Applications that use context to provide services and relevant information are called CSS. These systems must have the following characteristics: monitoring, awareness and adaptability (KLEIN et al., 2008). Accordingly, CSS are flexible, able to act autonomously on behalf of users and dynamically adapt their behavior.

Thus, CSS use context to direct actions and behaviors to support communication between systems and their users (VIEIRA; TEDESCO; SALGADO, 2011). To provide this support can be used (CHALMERS, 2002):

a) *Contextual sensing* - where the context is sensed and information describing the current context, e.g. location, temperature, can be presented to the user.

b) *Contextual augmentation* - consists of the association of context with data, for example, records of objects surveyed can be associated with location, meeting notes can be associated with people in the meeting and the place the meeting was held.

c) *Contextual resource discovery*, e.g., to cause printing to be on the nearest printer.

d) *Context triggered actions* - context is used to trigger actions in a system such as loading map data for an area to be entered.

e) *Contextual mediation* - using context to modify a service.

In our work, we are concerned with the last two cases *Context triggered actions* and *Contextual mediation*. They are two types of *Contextual adaptation* (CHALMERS, 2002) that are responsible for changing the sequence of actions, the style of interactions and the type of information provided to users in order to adapt to the user's current needs or requirements failure (VIEIRA; TEDESCO; SALGADO, 2011).

There is no consensus on the definition, terminology and related terms associated to context. BAZIRE; BRÉZILLON (2005) cataloged more than 150 definitions of the concept and concluded that there are many definitions for context that differs strongly across different domains (computer science, philosophy, economy, business, human computer interaction). As an example, in a context-sensitive search engine, if a user searches the term "java" that could mean a programming language or an island. To disambiguate the searched term, the engine may look to the context that can be the query history. If the user asked recently for any programming language such as Pascal, C++, PHP, then most probably he is looking for the Java programming language (ALI; DALPIAZ; GIORGINI, 2010).

One possible explanation for this lack of consensus is that the context is used supposing that everybody knows its meaning and the definitions of context are too much dependent of their own contexts (e.g. the discipline in which the definition is taken but also on both the kind and the goal of a given text) (BAZIRE; BRÉZILLON, 2005).

In this work, we adopted the definition from the perspective of goal-oriented requirement engineering proposed by ALI; DALPIAZ; GIORGINI (2010). They defined a framework that states that *context is a partial state of the world that is relevant to an actor's goals*. Accordingly, each context must be refined to allow it to be checked. The contextual refinement has a tree-like structure (see Figure 2.2) in which the root of this model is the context, and facts and statements are its nodes. In order to obtain a verifiable context, all statements are refined into sub-statements and facts, until there are only facts left.

Facts are predicates which truth values can be verified in a context, i.e. the system can capture the necessary data and compute the truth value of a fact. For example, *The meeting date is more two days away*.

Statements, on the other hand, cannot be verified directly in a context due lack of information or its abstract nature. For example, *The meeting is not urgent*, is a subjective assertion that does not have a clear criteria to be evaluated against. A statement S is supported (through **Support** relation) by a set of facts that gives enough evidence to the truth of S. Hence, a statement is not monitorable by itself but it is an abstraction of visible facts.

Statements are represented as shadowed rectangles and facts as parallelograms (ALI; DALPIAZ; GIORGINI, 2010). The support relation is represented as curved filled-in arrow, and the **implication logical operators and, or** are represented as black triangles, white triangles, filled-in arrows, respectively.

Figure 2.2 presents the refinement of the C4 context related to *Collect timetables by email* task of the meeting scheduler example. It will be true if *The number of participants is high*

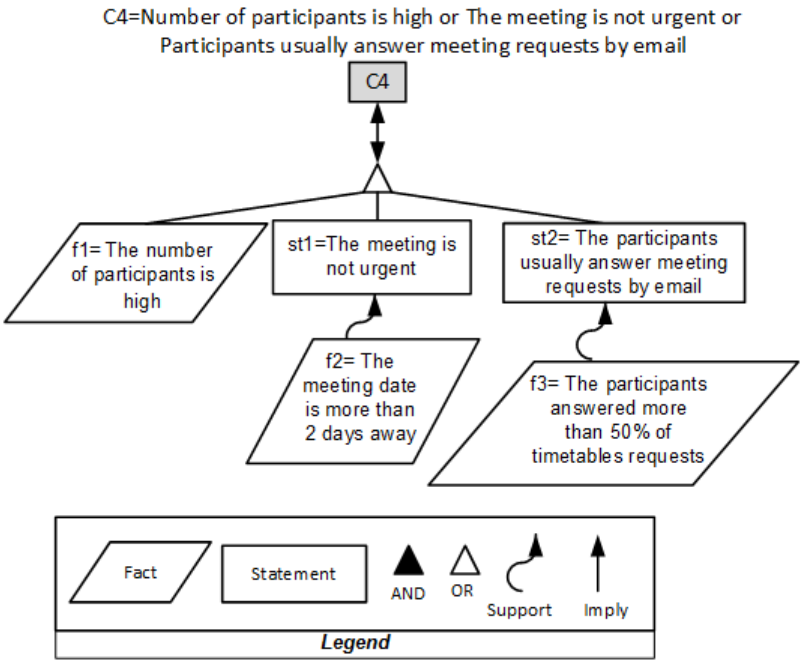


Figure 2.2: Refinement of C4 context of meeting scheduler example (AUTHOR, 2015).

(since it will be time consuming calling too many participants), or *The meeting is not urgent* (which can be checked by the fact *The meeting date is more than 2 days away*). If it is urgent, the participants should be contacted by phone. Likewise, it is required that *The participants usually answer meeting requests by email* (i.e. *The participants answered more than 50% of timetables requests*).

The context acts like a set of constraints that influence the behavior of a system embedded in a given task (BAZIRE; BRÉZILLON, 2005). Hence, context-sensitive systems must monitor the context at runtime in order to decide which action will be executed. Accordingly, in addition to the specification of the facts and statements that defines a context, it is necessary to specify the real-word properties of the facts that changes at runtime. These properties are called Contextual Elements (CEs) (VIEIRA; TEDESCO; SALGADO, 2011). These elements can be defined as data or information in the domain whose instantiated values influence the truth values of facts in the contextual refinements.

Each CE can be identified with respect to 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 for the specification of the context monitoring as will be discussed in Section 3.

The design of CSS entails more work in comparison to systems that do not consider context since they must care for context-related tasks, such as the acquisition, processing, storage

and presentation of contextual information. In the next section, we provide some concepts related to Requirements Engineering (RE) and a goal-oriented approach for context modeling.

2.3 Goal-Oriented Requirements Engineering

RE is a branch of software engineering that deals with elicitation, refinement, analysis, of software systems requirements. Therefore, it addresses the reasons why a software system is needed, the functionalities it must have to achieve its purpose and the constraints on how the software must be designed and implemented (LAPOUCHNIAN, 2005).

The popularity of GORE approaches has increased dramatically. GORE is concerned with the use of goals for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying requirements (LAPOUCHNIAN, 2005).

Goal models capture and refine stakeholder intentions to generate functional and non-functional requirements. A goal model consists of a hierarchy of goals, tasks, and softgoals that relates the high-level goals to low-level system requirements (see Figure 2.1). The notation used in this dissertation is based on the notation described by CASTRO; KOLP; MYLOPOULOS (2002) which has goals, tasks, softgoals and contribution links (Make (++), Help (+), Hurt (-) or Break (- -)).

Goals represent an intention of a system such as *Meeting Scheduled* and *Timetables collected* in the meeting scheduler example (Figure 2.1). They are iteratively decomposed into subgoals by And-refinement (all subgoals should be achieved to fulfill the top goal) and Or-refinement (at least one subgoal should be achieved to fulfill the top goal). Goals are satisfied by means of executable tasks.

Tasks are elements directly mapped to functionality in the running system and are satisfied if executed successfully. Examples of tasks of the meeting scheduler example are *Schedule automatically* and *Collect by email*.

Softgoals are goals whose satisfaction cannot be established in a clear-cut sense. Unlike regular goals, softgoals can seldom be said to be accomplished or satisfied through **contribution links (Make, Help, Hurt, Make**). These links allow us to qualitatively specify that there is evidence that certain tasks/softgoals contribute positively or negatively to the satisfaction of softgoals. *Make* is a positive contribution strong enough to satisfy a softgoal. *Help* is a partial positive contribution not enough by itself to satisfy the softgoal. *Hurt* is a partial negative contribution not enough by itself to deny softgoal. Finally, *Break* is a negative contribution enough to deny a softgoal.

The variability of a system is modeled in a goal model through OR decompositions of goals and tasks namely variants. The variants that a system can adopt to meet its requirements are influenced by context. Therefore, we describe in next section a RE modeling framework (ALI; DALPIAZ; GIORGINI, 2010) for systems operating in and reflecting varying contexts.

2.4 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. The notation used in this paper to represent contexts in goal models is based on ALI; DALPIAZ; GIORGINI (2010). Thus, contexts in our work can be associated with the following variation points in a goal model: or/and refinements and contribution to softgoals.

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. For example, consider the or-refinement of *Timetables Collected* goal in Figure 2.3. The *Collect automatically* task will be executed only if C5 holds, the same reasoning applies to the other tasks in this refinement.

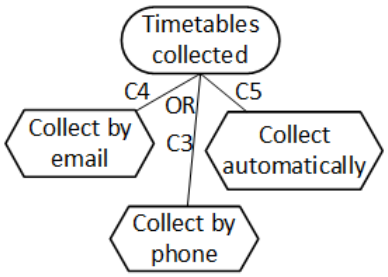


Figure 2.3: OR-Refinement of Timetables Collected goal of meeting scheduler example (AUTHOR, 2015).

The C5 context refined with facts and statements is presented in Figure 2.4.

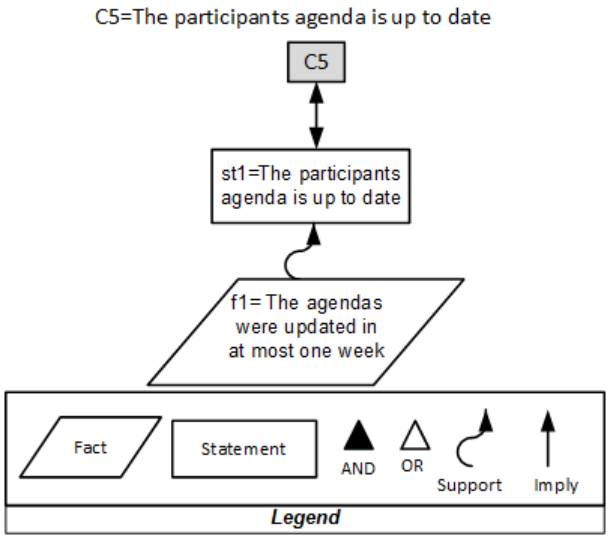


Figure 2.4: Refinement of C5 context of meeting scheduler example (AUTHOR, 2015).

And-refinement

The satisfaction (execution) of a subgoal (subtask) in this refinement is needed only in certain contexts. A subgoal/task in this refinement must be provided by its parent node but its execution is context-dependent. For example, consider the *Manage performance adaptation* task in Figure 2.5. The meeting scheduler system must be able to perform this task, but it has not to be in all cases, only when the C7 context holds.

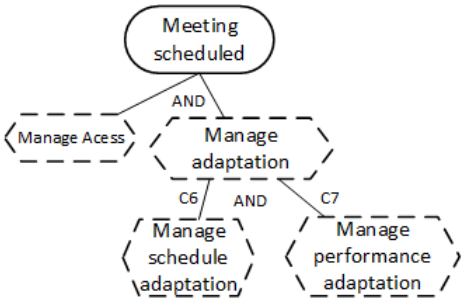


Figure 2.5: And-Refinement of Timetables Collected goal of meeting scheduler example (AUTHOR, 2015).

The C7 context refined with facts and statements is presented in Figure 2.6.

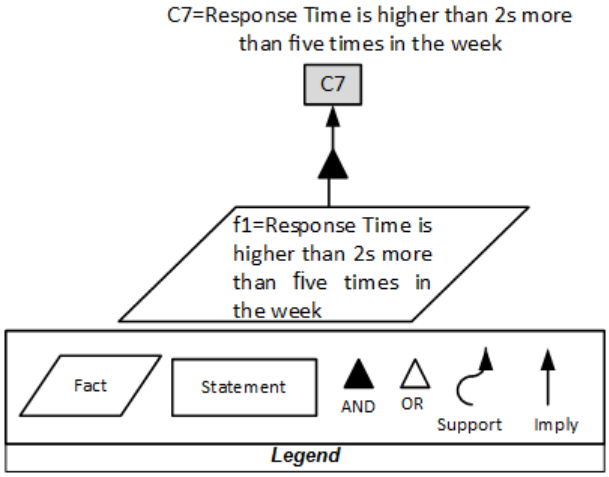


Figure 2.6: Refinement of C7 context of meeting scheduler example (AUTHOR, 2015).

Although this is syntactically equivalent to an or-refinement, the semantic is different. A context on an and-refinement influences 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

Softgoals can be contributed either positively or negatively by goals and tasks. The contribution of softgoals can vary from one context to another as demonstrated in Figure 2.7. For

example, the *Turn on air ventilator* task contributes positively to *Privacy* softgoal and negatively to *Energy spent wisely* softgoal when C12 (It is sunny and not very windy) holds.

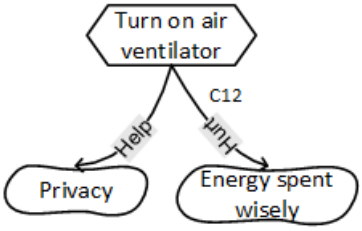
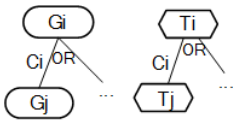
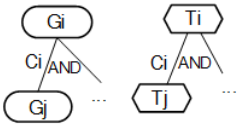
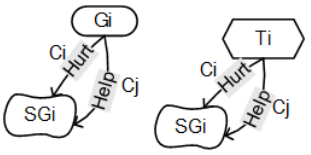


Figure 2.7: Contribution to softgoals context-dependent adapted from ALI; DALPIAZ; GIORGINI (2010).

The visual syntax and semantics for each type of variation point are presented in Table 2.1. A variation point consists in a refinement in the goal model that is context-dependent.

Table 2.1: Semantics for the contextual variation points adapted from ALI; DALPIAZ; GIORGINI (2010).

Variation point	Visual Syntax	Semantics
OR-Refinement		Goal Gi (task Ti) can be achieved (executed) via Gj (Tj) if context Ci holds.
AND-Refinement		The achievement of goal Gi (the execution of task Ti) requires Gj (task Tj) if context Ci holds.
Contribution to softgoals		Goal Gi (task Ti) contributes positively to softgoal SGi if context Cj holds. It contributes negatively to softgoal SGi if context Ci holds.

According the framework of ALI; DALPIAZ; GIORGINI (2010), for each variation point in the goal model, the software engineer has to decide if a variation point is context-dependent or not. When a contextual variation point is identified, the variants at the goal model are labelled by C1...Cn and annotated in the model (see Figure 3.8), and described as a sentence (see Table 2.2).

The framework of ALI; DALPIAZ; GIORGINI (2010) requires that each context identified in a contextual goal model must be refined to allow it to be checked. This context analysis allows the software engineer to identify the facts that the system has to verify. These facts are verifiable on the basis of data a system can collect of the world.

The analogy between goal analysis and context analysis is shown in Figure 2.8. While a goal is a partial state of the world that a system attempts to reach, context is a partial state of the world that a system attempts to judge if it holds. For example, consider the *Collect timetables by*

email task of the meeting scheduler example of Figure 2.3. This task will be executed only if C4 context holds. This context are refined in a set of facts and statements (see Figure 2.2) that should hold at runtime.

Accordingly, goal analysis justifies why a system takes some actions (tasks), while context analysis provides the rationale that explains why a system needs to collect data and check facts of its environment (ALI; DALPIAZ; GIORGINI, 2010).

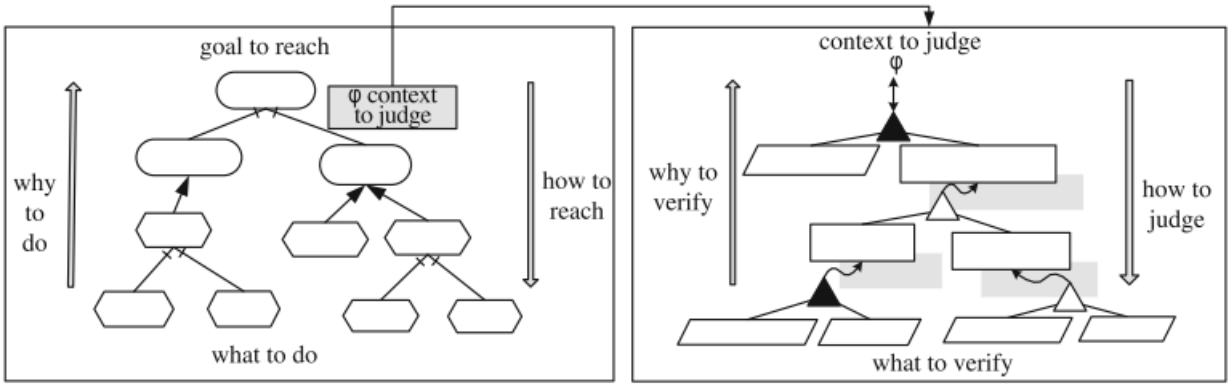


Figure 2.8: The analogy between goal analysis and context analysis (ALI; DALPIAZ; GIORGINI, 2010).

Content-sensitive systems operate in many different contexts and therefore it is important to specify the equipments or technologies necessary to install and to use in order to verify if the context holds (ALI; DALPIAZ; GIORGINI, 2013). These equipments can be of several types like GPS, RFID, camera, different types of sensors (presence, humidity, light), information stored in databases or another mechanism to information storage, etc. To provide such information, expertise in technologies is needed. They can be specified in a table as shown in Table 2.2. They are identified after analyzing the facts for each context in the contextual goal model.

Table 2.2: Example of descriptions of contexts and the equipments/technology needed to monitor them.

	Description	Variation Point Type	Equipment or Technology
C1	The number of participants is small and The number of rooms available is small and The date range is small	OR	Mechanism to information storage.
C2	The number of participants is high and The number of rooms available is high and The date range is high	OR	Mechanism to information storage.
C3	The number of participants is small or The meeting is urgent or Participants do not usually answer meeting requests by email	OR	Mechanism to information storage.

2.5 Design Goal Model

The design goal model is an extension of the goal model proposed by PIMENTEL et al. (2014) to include new elements that appeared in the design phase, such as design tasks and design constraints.

Design tasks allow the architect to model tasks that, although not relevant for the stakeholders at a first moment, are important for the definition of the system behavior. For example, a login task can be a design task identified by architects to satisfy the security softgoal.

Design quality constraints, on the other hand, restrict the initial goals and tasks or define requirements quality constraints into more concrete ones (PIMENTEL et al., 2014). For example, the performance softgoal can correspond to *The response time should be less than 2 seconds* constraint. Note that these design elements usually emerge after requirements elicitation.

Accordingly, these new elements are 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. However, these elements have the same semantic of tasks and quality constraints of a goal model. Therefore, by including these refined 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.

Design goal models do not specify the order of execution of the tasks. In order to perform such specification, we rely on flow expressions described in next section.

2.6 Flow Expressions

Flow expressions are an extension to regular expressions and can be used to describe the behavior of software architectures through its goals and tasks in a goal model. Accordingly, flow expressions are an useful notation that can aid in the design, analysis, and understanding of software systems (SHAW, 1978).

We adopted the symbols proposed by PIMENTEL et al. (2014), described in Table 2.3, with the purpose of facilitating their writing. Each atomic symbol represents a state related to an element of the goal model. For example, if *t1* is a task, the atomic expression *Ti* represents the state where *t1* is being fulfilled.

Flow expressions can be composed in terms of regular expression operators, such as concatenation (t1 t2), meaning first satisfy t1 then t2 (sequential flow), or t2*, meaning that t2 is to be satisfied zero or more times. Flow expressions separated with a vertical bar | symbol represent alternative flows. The question mark ? is used to represent the optionality of the flow to its left, i.e., that flow may be executed zero or one times.

The star symbol * indicates that the flow to its left may be executed zero, one or more times, while the plus symbol + indicates that the flow may be executed one or more times. A

dash - indicates that two flows are to be carried out concurrently, in the sense that their states can be interleaved. These expressions allow the representation of the system behavior as annotations on a goal model. Table 2.3 presents each symbol, its meaning and an example.

Table 2.3: Symbols of flow expressions (PIMENTEL et al., 2014).

Expression	Meaning	Example
blank space	Sequence	(t1 t2), first t1 and then t2
|	Alternative	(t1 | t2), t1 xor t2
?	Optional	(t1 t2? t3), first t1 and then t3, or first t1 followed by t2 and t3
*	Zero or more times	(t1 t2* t3), first t1, then t2 zero or more times, then t3
+	One or more times	(t1 t2+), first t1, then t2 one or more times
-	Parallelism	(t1-t2), t1 is executed at the same time as t2

As an example of flow expressions, consider the following flow expression: (A B (C | D) E F* G) - (H*). This expression indicates that state A is followed by state B. After B, the possible states are C or D (exclusively), followed by E. After E, F may be reached any number of times. State G occurs after E or after F. Concurrently to all that, the state H may occur any number of times.

2.7 Software Architecture

Software architecture is composed of elements, connections or relations among them, and, usually, some other aspect (s) such as configuration; constraints or semantics; properties; or rationale, requirements, or stakeholders' needs (CLEMENTS et al., 2002). Accordingly, a software architecture is a complex entity that cannot be described in a simple one-dimensional fashion.

Therefore, software architecture can be represented through four views: structural, behavioral, deployment, and configuration (CLEMENTS et al., 2002). A view is a representation of a set of system elements and the relationships associated with them. Each vision type express different systems characteristics, and together are capable of defining the system architecture as a whole.

The structural view is concerned with logical relationships between the components and connectors. The behavioral view, expressed for example through state diagrams, describes the execution order of system tasks, including the definition of parallelism and temporal dependencies. It also allows analysis of potential deadlocks; performs resource optimization tests; being a primordial vision for a complete architectural description (CLEMENTS et al., 2002).

The deployment view is similar to the structural view. However, is focused on the

physical structure of the system. This view is used to define the devices to be used; the networks to be created, what parts of the system run on what equipment, among others.

Finally, the configuration view explores every possible variation of the system, either those that can be selected manually by the user as the ones chosen by the system itself autonomously (CLEMENTS et al., 2002). Hence, these possible variations are expressed in a feature model.

In this dissertation, we are concerned with the behavioral view since documenting behavioral aspects of an architecture provides many benefits, both during development of the architecture and during system maintenance. This information can be used to improve the understanding of a system. Moreover, it can also help stakeholders to reason about how a system built will be able to meet its quality-related goals (CLEMENTS et al., 2002).

Many notations may be used to capture behavioral information of context-sensitive systems such as LTS (NICOLA, 1987), Petri Nets (MURATA, 1989), and Statecharts (HAREL, 1987). In this dissertation we rely on statechart, a popular visual formalism to represent the behavioral view of a system. In the next section, we will detail this behavioral language.

2.8 Statecharts

Statecharts (HAREL, 1987) extend conventional state-transition diagrams with essentially three elements, dealing, respectively, with the notions of hierarchy, concurrency and communication. Besides, this notation is an interesting visual formalism for modeling context-sensitive systems since they are reactive and adapt its requirements.

Statecharts can be used as a stand-alone behavioral description, and, therefore, became a popular visual formalism for modeling reactive systems (HAREL, 1987). These systems are characterized by being, to a large extent, event-driven, continuously having to react to external and internal stimuli such as context-sensitive systems.

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.

Figure 2.9 presents the statechart corresponding to flow expression described in Section 2.6. State A is followed by state B. After B, the possible states are C or D (exclusively), followed by E. After E, F may be reached any number of times. State G occurs after E or after F.

Concurrently to all that, the state H may occur any number of times.

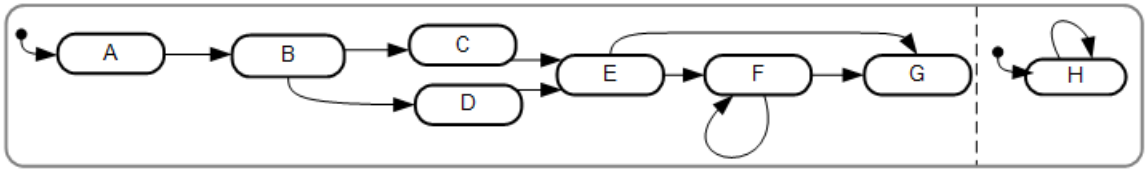


Figure 2.9: Statechart Example (AUTHOR, 2015).

2.9 Final Considerations

This chapter presented the meeting scheduler example used in the next dissertation to illustrate our process as well as the main concepts required for the understanding of this dissertation. GORE approach was presented showing the main advantages in the context of this dissertation. We discussed the concepts of context-sensitive systems and the contextual goal model, a goal-oriented requirements approach for modeling these systems. In order to define the system's behavior, we presented the flow expressions. Besides, the main concepts of software architecture behavior used in this work were outlined. In particular, we presented statecharts language, discussing its elements.

Developing CSS is a complex and a labor-intensive task. When designing these systems, the software engineer needs to deal with issues associated with: the kind of information to be considered as context, how to represent this information, how to acquire and process it (considering that it may come from several and heterogeneous sources) and how to integrate the context usage in the system (SANTOS, 2008). Hence, there is a need for approaches that guide CSS designers on obtaining their behavior from requirements models and performing activities related to the system's behavior specification. In the next chapter, we present the GO2S process - a systematic process to derive the behavior of context-sensitive system from requirements models.

3

The GOals to Statecharts Process

Processes provide steps that support an activity. They can be used as checklists and guidelines of what to do and how to do it (WOHLIN et al., 2012). In order to develop context-sensitive systems, several steps to obtain their behavior have to be taken and they have to be in a certain order. Thus, a systematic process for how to obtain their behavior is needed.

In this dissertation, we propose the GO2S process to systematically derive the behavioral view of context-sensitive systems architecture (modeled as statecharts), from system's requirements (modeled as goal models) following the twin peaks concept (NUSEIBEH, 2001).

To specify the GO2S process we use the Business Process Model and Notation (BPMN) (OMG, 2014). BPMN is a well-adopted process-modeling standard supported by many software tools that provides a graphical notation that describes the flow of a process. Since this notation has been specifically designed to coordinate the sequence of processes and the messages that flow between different process participants in a related set of activities, it facilitates the understanding of the procedures and ensure that software engineers understand themselves.

The process is divided into the following main steps: the first activity concerns the construction of Design Goal Model (DGM). It is followed by the specification of contextual variation points. In the third activity, the tasks required for the monitoring and adaptation activities are specified. Later, in the fourth activity, the system behavior is represented in flow expressions. The next one derives a statechart from the behavioral contextual design goal model. Finally, the last activity is the prioritization of variants.

We explored the idea that it is possible to modularize the development of CSS by organizing the GO2S process in six activities considering the main steps in order to develop a context-sensitive system: requirements specification, context specification, the adaptation and monitoring, the definition of system's behavior, the statechart derivation and the prioritization of variants.

The process is not supposed to be a waterfall model. Thus, it is not assumed that an activity must be totally finished before the next one is started. This is an iterative process centered on the incremental refinement of a goal model, obtaining different views of the system (design, contextual, behavioral). Accordingly, it may be necessary to go back and forth in the activities

until the desired level of detail is reached.

The activities of GO2S process are illustrated in Figure 3.1, and further elaborated in the following sub-sections. At the beginning of each activity, we present a summary of its goal, inputs, steps, and outputs.

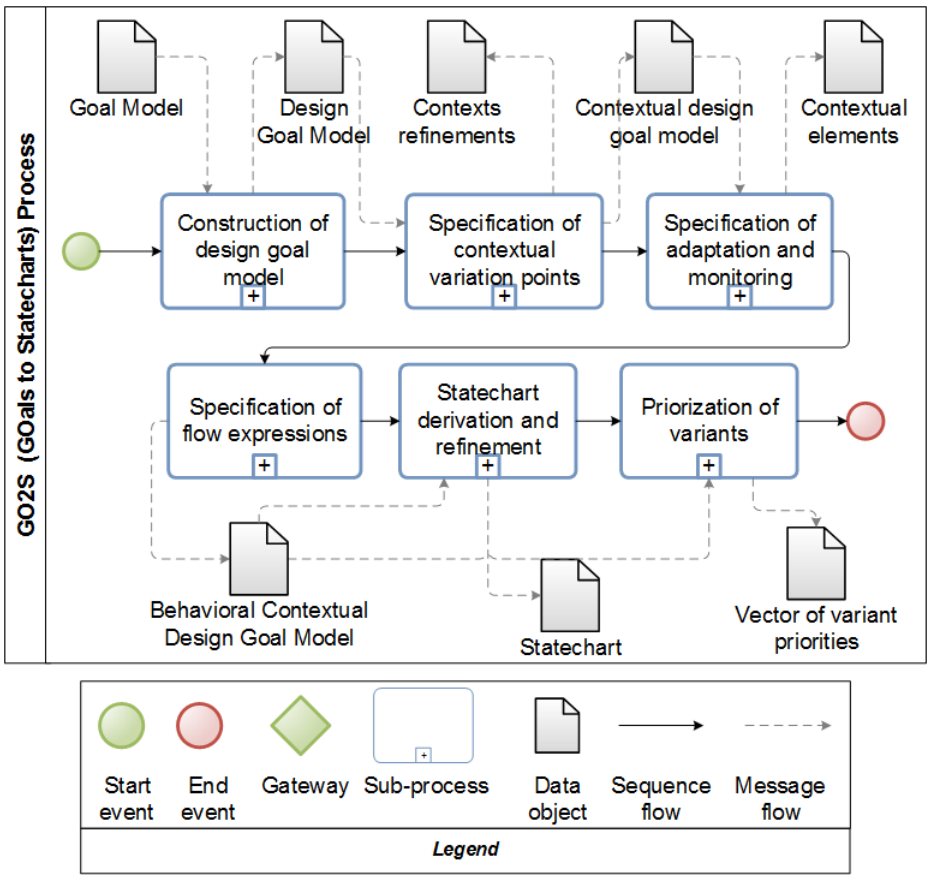


Figure 3.1: The GO2S process for deriving the behavior of context-sensitive systems (AUTHOR, 2015).

3.1 Activity 1: Construction of design goal model

- Goal: Refine a goal model with new design elements
- Input: A goal model
- Steps:
  - 1: Identify design tasks and constraints
  - 2: Perform the NFR analysis
  - 3: Include the design tasks that operationalizes the NFRs in the goal model
  - 4: Assign Tasks

- Outputs:

  DGM

  Operationalization of NFRs

Our overall objective is to obtain the behavioral view of context-sensitive systems architecture. Thus, we assume that requirements elicitation and analysis activities were previously performed and a goal model was generated. Hence, the first activity (see Figure 3.2) of our process consists of *Construction of DGM* and receives a goal model as an input.

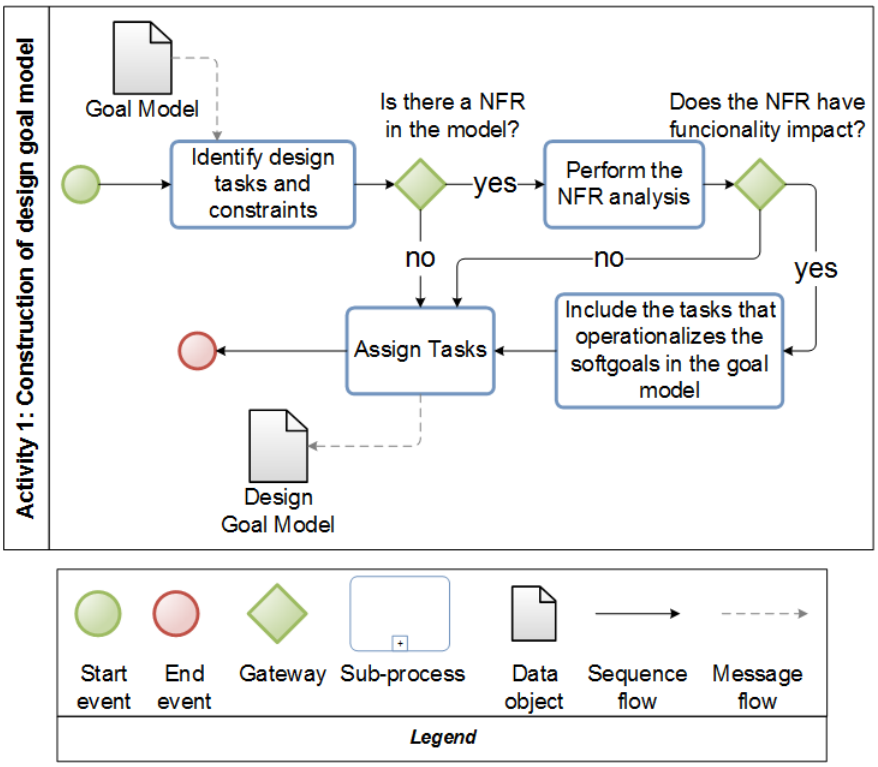


Figure 3.2: Steps of *Construction of Design goal model* activity (AUTHOR, 2015).

As we move to the solution space, it can be necessary to include new elements that appeared in the design phase, such as design tasks and design constraints (PIMENTEL et al., 2014). In this first activity of the process, the software engineer should check if there is critical design task or quality constraint that was not identified in the requirements phase that is necessary to the system. One source of these elements are NFRs. 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 (LIU; MA; SHAO, 2010).

Therefore, note that in this first activity, we can establish the relationship between NFRs and the goal model using techniques for NFR analysis such as Softgoal Interdependency Graph (SIG) (CHUNG et al., 2000). 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.

Moreover, the design goal model also allows the definition of assignments for its tasks. A task or design task may be assigned/delegated to one or more users. Assignments are expressed

by labels below the assigned element (PIMENTEL et al., 2014). The left side of the label shows the icon of a person, to represent the assignment. The users to whom the task was assigned to are listed to the right of the icon, as shown in Figure 3.3.

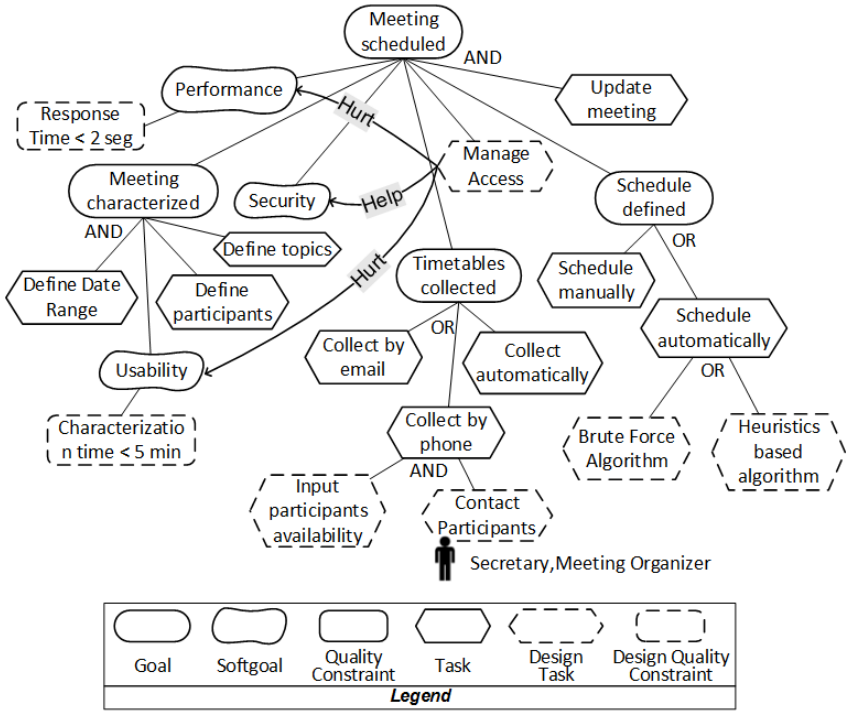


Figure 3.3: DGM of meeting scheduler example adapted from ANGELOPOULOS; SOUZA; MYLOPOULOS (2014).

In the meeting scheduler system, we have the usability, performance and security NFRs. In order 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. Besides, the *Contact Participants* design task may be performed either by the meeting organizer or by a secretary.

This assignment was chosen since developing the capability of making automatic phone calls and collecting timetables would be too costly. In order to make this kind of decision, it may be necessary to consult with the project stakeholders in order to find the most beneficial option. The DGM of the meeting scheduler which encompasses both requirements and design elements, is shown in Figure 3.3.

The outputs of this process activity are the design goal model which can include the design tasks that operationalizes the NFRs.

The OR refinement of DGMs introduces alternatives into the model accommodating many/all possible functionalities that fulfill stakeholder goals. The space of alternatives defined by a goal model can be used as a basis for designing fine-grained variability for highly customizable or adaptable software (YU et al., 2008). Hence, we need to consider variability of the model (ALI; DALPIAZ; GIORGINI, 2010) as described in next activity.

3.2 Activity 2: Specification of contextual variation points

- Goal: Refine a design goal model with contextual variation points
- Input: A design goal model
- Steps:

  1: Identify and specify the contextual variation points

  2: Refine contexts
- Outputs:

  Contextual design goal model

  Contexts refinements

In this second activity, the DGM constructed in the previous step is refined with contextual variation points and their associated contexts are refined as demonstrated in Figure 3.4.

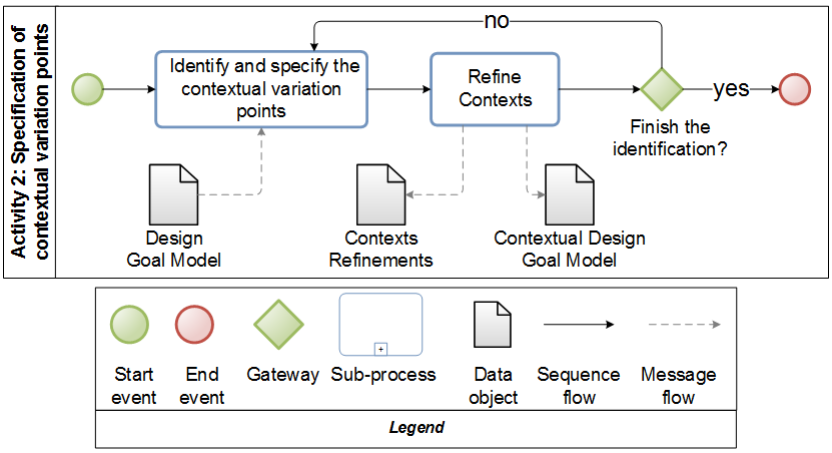


Figure 3.4: Steps of *Specification of contextual variation points* activity (AUTHOR, 2015).

The contextual variation points represent the context influence on the choice among the available variants of goals satisfaction. They are annotated in the DGM to visually specify the effects of context in the system's behavior. In the GO2S process, contexts can be associated to or/and refinements as well as contribution to softgoals present in a design goal model.

Accordingly, the architect should identify a contextual variation point and specify the variants at the design goal model with context labels such as C1...Cn as shown in Figure 3.5. We followed the notation proposed by ALI; DALPIAZ; GIORGINI (2010) which requires that each context specified in the contextual DGM must be refined through a set of statements and facts. 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.

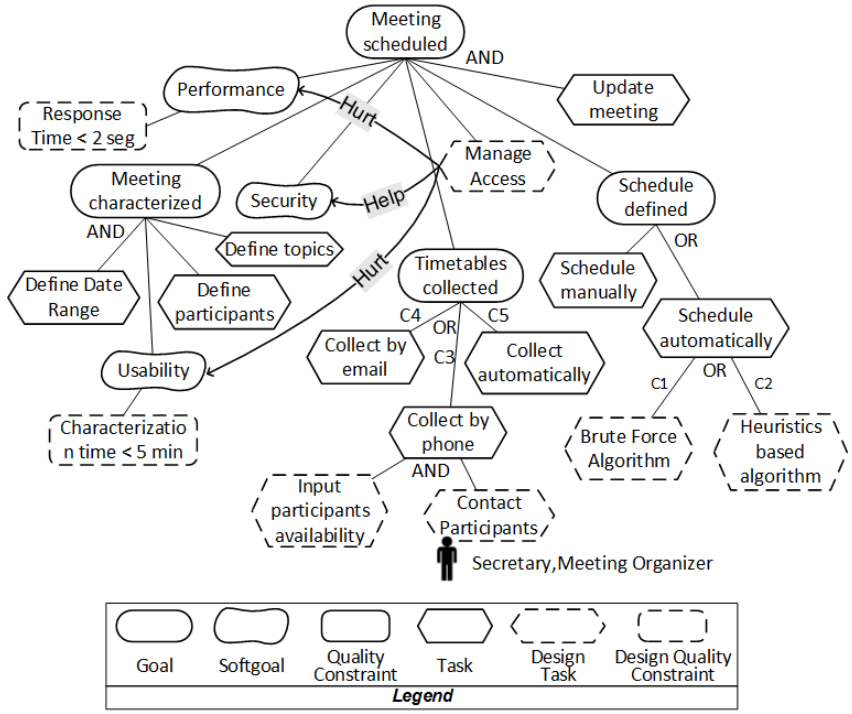


Figure 3.5: Contextual DGM of meeting scheduler example (AUTHOR, 2015).

As an example of context refinement, consider the refinement of C4 context related to *Collect timetables by email* task in Figure 2.2. C4 will be true if *The number of participants is high* fact, or *The meeting is not urgent* statement (which can be checked by *The meeting date is more than 2 days away* fact), or *The participants usually answer meeting requests by email* statement (verified by the *The participants answered more than 50% of timetables requests* fact) are true. The contextual DGM of the meeting scheduler example is shown in Figure 3.5.

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 Activity 3: Specification of adaptation and monitoring

- Goal: Refine the contextual DGM with elements necessary for the specification of adaptation DTs as well as the monitoring
- Input: Contextual design goal model
- Steps:

  1: Define the critical requirements that requires adaptation

  2: Represent the adaptation management

  2.1: Add a new design task in the root node for adaptation management

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

2.3: Add design tasks to represent the adaptation strategies for each monitored requirement

3: Associate each adaptation design task with a context label

4: Refine each context

5: Identify the dynamic contextual elements

6: Represent the context monitoring

6.1: Add a new design task in the root node

6.2: Add design tasks to monitor each dynamic contextual element

7: Specify the equipments/technology necessary to monitor the contexts

- Outputs:

  Contextual design goal model refined

  Contexts Refinements

In this activity, the contextual DGM is refined with elements necessary for the specification of adaptation design tasks as well as the monitoring. Figure 3.6 presents the required steps.

Context-sensitive systems have the ability of adapt themselves in order to provide personalized services for its users when enable/disable functions. Therefore, we propose to use this characteristic to deal with the requirements adaptation when a goal fails. In order to achieve this, we refine a contextual DGM with elements necessary for the specification of adaptation design tasks as well as the monitoring.

The input of this third activity is the contextual design goal model. The software engineer should analyze the system's requirements, aiming to define the requirements that are critical, and therefore require some action in case of failure. Our approach does not prescribe any specific technique for elicitation and analysis of the requirements. Thus, the software engineer should choose existing requirement elicitation techniques that best fit. However, we can suggest some common sources of requirements that usually require adaptation since usually they are critical and impact the system behavior:

- Softgoals present in the goal model;
- Goals that are critical for the system-to-be to fulfill its purpose, since some subsequent activities may depend on them;
- Government regulations and rules may require that certain goals cannot fail or be achieved at appropriate rates;
- Requirements related to Service Level Agreements (SLAs).

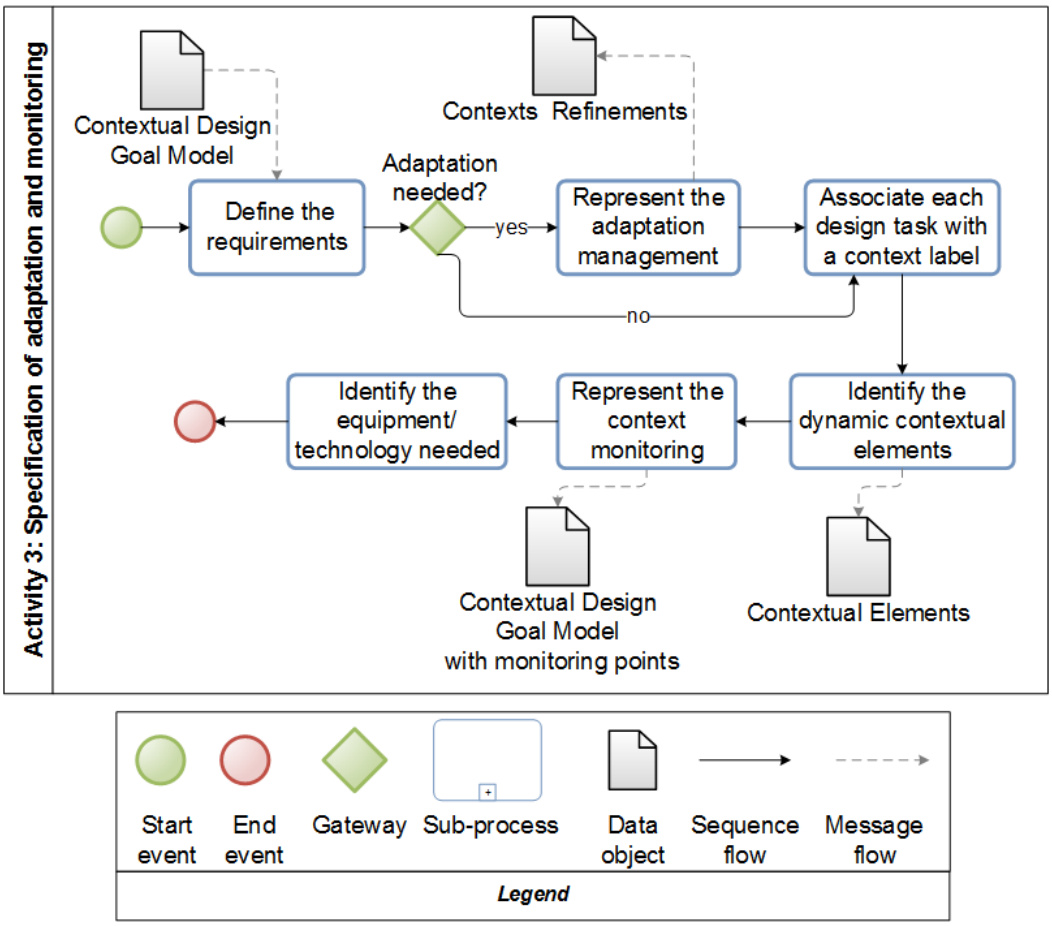


Figure 3.6: Steps of *Specification of adaptation and monitoring* activity (AUTHOR, 2015).

Next, adaptation design tasks should be added in the root node of the contextual DGM following the Step 2 described in the beginning of the section. These adaptation design tasks represent the tasks required for adaptation of each requirement the software engineer wants to adapt. We propose to add a new design task in the root node for adaptation management as well as design tasks in this node for each critical requirement that must be monitored and adapted.

Finally, design tasks should be added to represent the adaptation strategies. The adaptation design tasks can be of several types, such as reconfiguration of system's parameters, step back or delegate the task to an user. Note that we should add at least two adaptation design tasks since the variants are the cornerstone for adaptability, a system with only one variant cannot be adaptable (ALI; DALPIAZ; GIORGINI, 2010).

The next step is to associate each adaptation design task with a context label meaning that these elements will be activated when the associated context holds. For example, consider the *Manage Schedule adaptation* design task in Figure 3.8. This task will be executed when C6 context holds. According to the framework of ALI; DALPIAZ; GIORGINI (2010), the contexts should be refined in statements and facts. The refinement of C6 context is presented in Figure 3.7.

Therefore, C6 context will hold when *The Number of conflicts is higher than [MCA]*

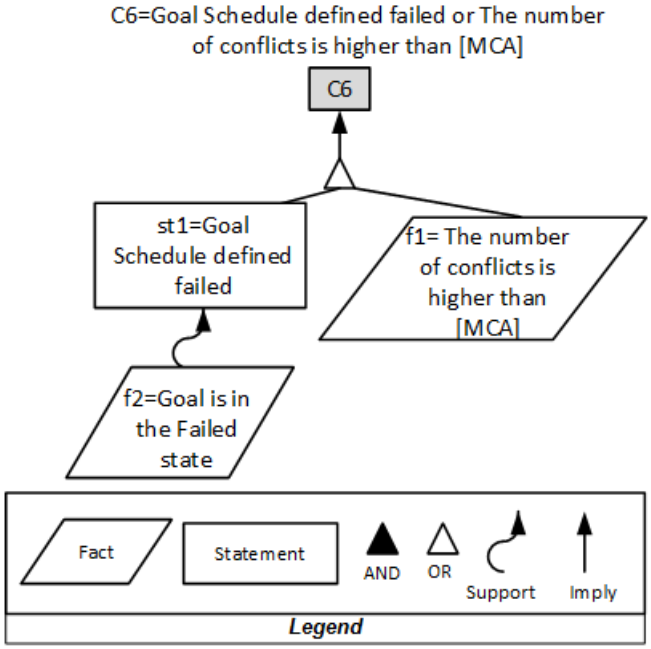


Figure 3.7: Refinement of C6 context of meeting scheduler example (AUTHOR, 2015).

(being MCA the Maximum Conflicts Allowed by the stakeholders) or when *Schedule defined* goal failed, i.e. the fact *Goal is in the failed state*.

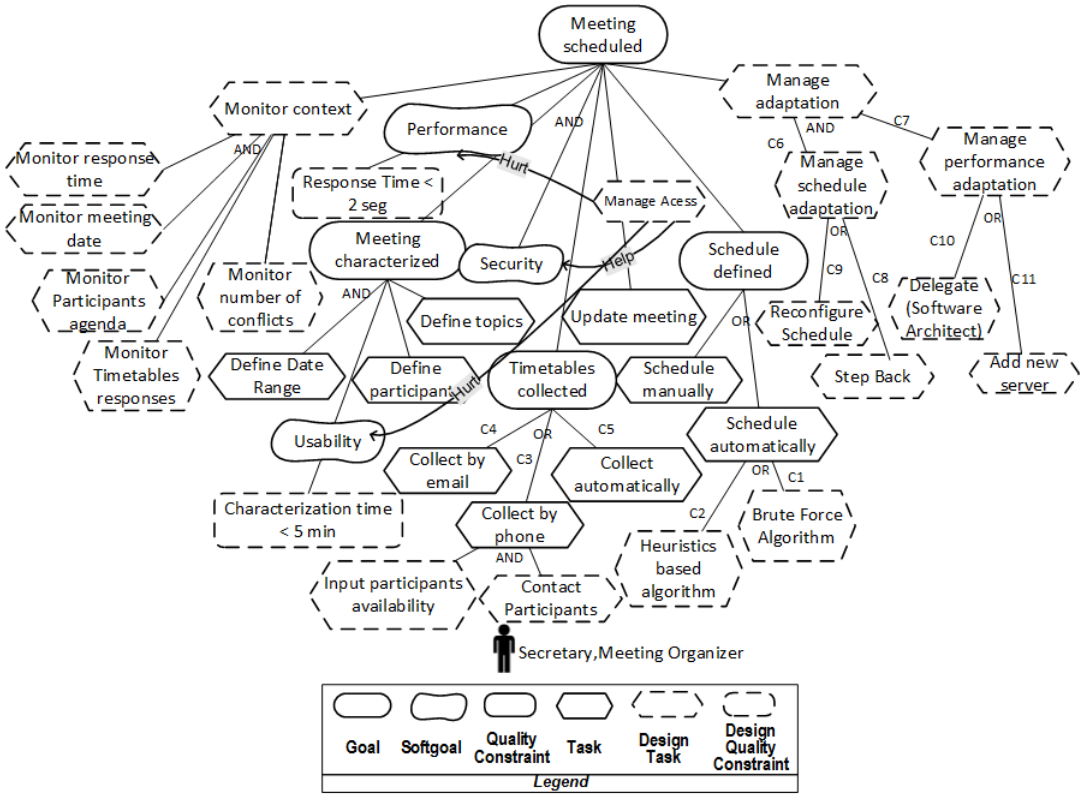


Figure 3.8: Contextual DGM of meeting scheduler example refined with adaptation elements (AUTHOR, 2015).

After all contexts, that influence the requirements are refined, and the contextual elements

that need to be monitored are identified, the next step is to identify the equipments/technologies needed to monitor these contextual elements (see Section 2.4). The contextual elements are the properties of real-world presented in the facts of context refinements that change their values dynamically. Therefore, the changes in the contextual elements imply in changes in the system context.

The context information can be represented in a table like Table 3.1 in order to facilitate the visualization and management by the software engineer.

Table 3.1: List of contexts of meeting scheduler example and the equipment/technology needed to monitor them m (AUTHOR, 2015).

	Description	Variation Point Type	Equipment or Technology
C1	The number of participants is small and The number of rooms available is small and The date range is small	OR	Mechanism to information storage.
C2	The number of participants is high and The number of rooms available is high and The date range is high	OR	Mechanism to information storage.
C3	The number of participants is small or The meeting is urgent or Participants do not usually answer meeting requests by email	OR	Mechanism to information storage.
C4	Number of participants is high or The meeting is not urgent or Participants usually answer meeting requests by email	OR	Mechanism to information storage.
C5	The participants agenda is up to date	OR	Mechanism to information storage.
C6	Goal Schedule defined failed or The number of conflicts is higher than [MCA]	AND	Mechanism to information storage.
C7	Response Time is higher than 2s more than five times in the week	AND	Mechanism to information storage.
C8	The Number of conflicts is higher than [MCA]	OR OR	Mechanism to information storage.
C9	The Step back strategy have failed	OR	Mechanism to information storage.
C10	The Add new server strategy have failed	OR	Mechanism to information storage.
C11	Response Time is higher than 2s more than five times in the week	OR	Mechanism to information storage.

In our running example, the software engineer decided that the system has to adapt itself when the *Performance* softgoal and the *Schedule Defined* goals fail. Therefore, two adaptation design tasks were added to our running example: *Manage Performance adaptation* and *Manage Schedule adaptation*.

The adaptation design tasks are *Step Back* and *Reconfigure Schedule* for the *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 adaptation elements of the meeting scheduler example are represented in Figure 3.8. In this figure, C1-C5 are the contexts previously identified, while C6-C11 are the ones related to the requirements adaptation identified in this activity. These contexts must hold so the adaptation design tasks can be executed.

In our meeting scheduler running example, the only technology needed to monitor the contexts annotated in the contextual goal model is a mechanism to information storage. The description of contexts and the equipment/technology are presented in Table 3.1.

However, the world is volatile and could be in different states. A partial state of the world that is uniform does not influence the decisions of a system (ALI; DALPIAZ; GIORGINI, 2010). Hence, the system should monitor the properties over the world that are dynamic and have a impact on system's behavior namely *contextual elements* (see Section 2.2).

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

In the meeting scheduler system, we identified five CEs: meeting date, timetables responses, participants' agenda, number of conflicts and response time. Thus, we added five design tasks to represent the monitoring of these contextual elements as shown in Figure 3.8.

The outputs of this activity are the refined contextual design goal model and context refinements. Having defined the adaptation strategies and the contextual elements that need to be monitored, we can now move on to specify the order of execution of tasks and goals. For this, we rely on flow expressions.

3.4 Activity 4: Specification of flow expressions

- Goal: Refine the contextual design goal model with flow expressions that represent the execution order of elements in the model
- Input: Contextual design goal model refined
- Steps:
  - 1: Assign an identification (ID) for each goal and task in the goal model
  - 2: Determine the flow expressions
  - 3: Specify idle states
- Output: Behavioral contextual design goal model

In this activity, the contextual design goal model is refined with flow expressions that represent the execution order of elements in the model through the steps shown in Figure 3.9.

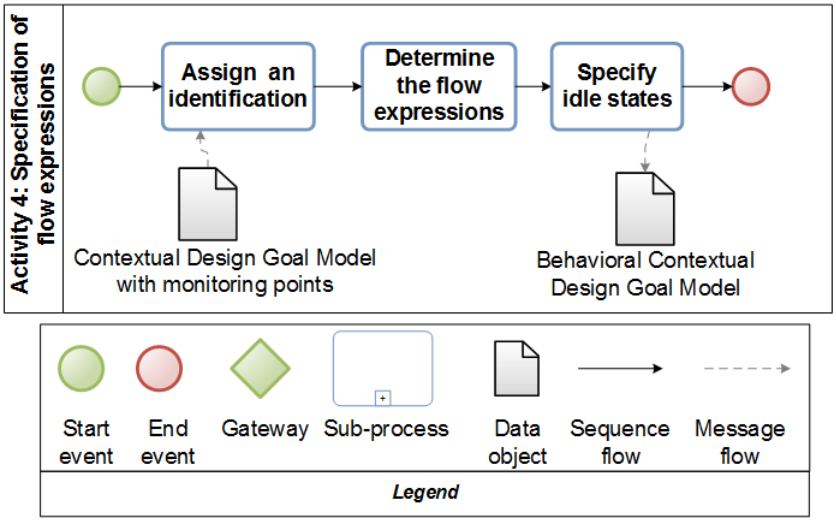


Figure 3.9: Steps of *Specification of flow expressions* activity (AUTHOR, 2015).

Flow expressions are a set of enrichments to a goal model that allow specification of the runtime behavior through the execution order of its elements (see Section 2.6). These expressions are used in the GO2S process as an intermediary model in order to derive the statechart.

The first step of the specification of flow expressions activity is to assign an identification to each goal and task in the model. This identification is necessary for reference in the flow expression later. *Gi* was used as the identification for goals and *Ti* for tasks and design tasks where *i* is the number of the task. Considering that there are different ways for a system to perform a set of tasks, determining the behavioral refinement (through flow expressions) is not a matter of direct translation (PIMENTEL et al., 2014).

After the IDs assignment, the next step is to define the flow expression for each parent node which describes the behavior of its children elements using the symbols proposed by PIMENTEL et al. (2014). The strategy can be bottom-up or top-down, the result will be the same. Thereafter, when we reach the root goal, we have the flow expression from the entire system. The resulting flow expressions should be annotated in the contextual design goal model as demonstrated in Figure 3.10.

The flow expressions could be defined taking in consideration the business rules of the system that were defined during the requirements phase. Therefore, the software engineer should analyze the requirements to determine the execution order of the tasks in the system, their optionality and multiplicity.

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

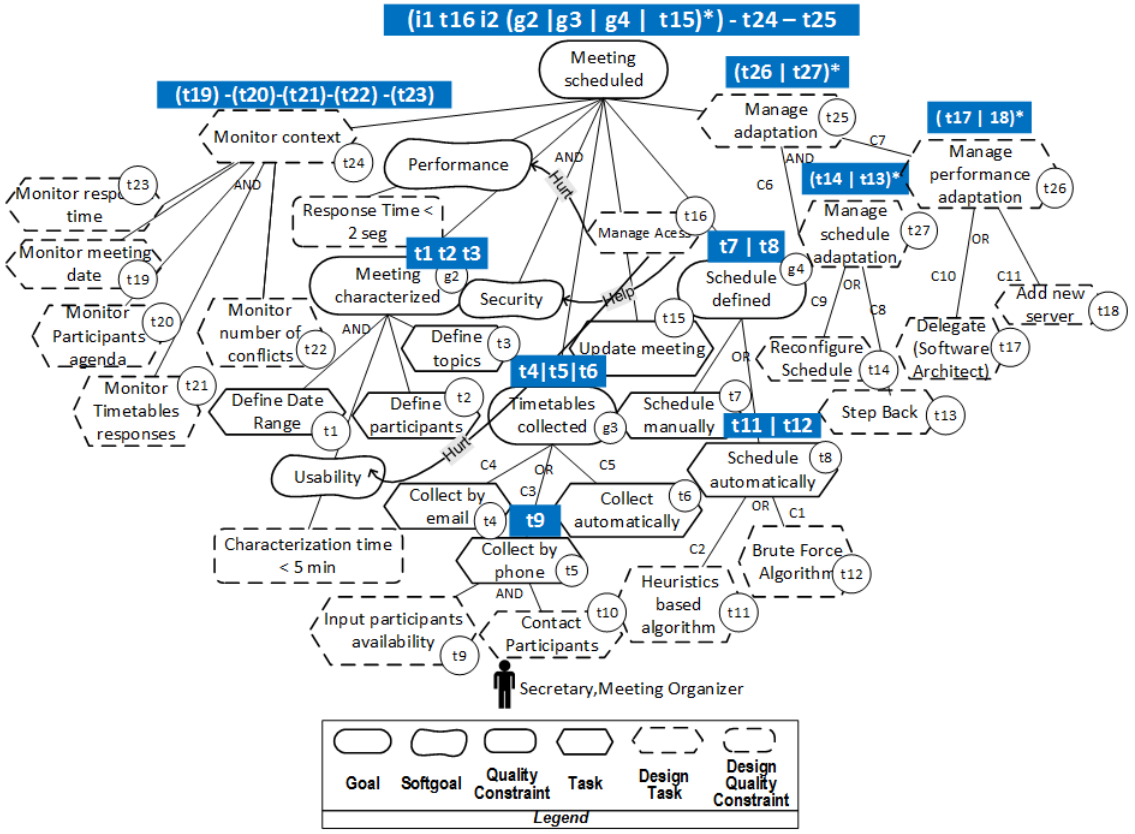


Figure 3.10: Behavioral contextual design goal model of the meeting scheduler example (AUTHOR, 2015).

The result flow expression, presented in Figure 3.10, 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)* design 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.

The output of this activity is the behavioral contextual design goal model. It is the contextual design goal model annotated with flow expressions. This model can represent in a unified way all the views developed in the GO2S (contextual, design and behavioral).

The behavior of a context-sensitive system can be simulated, using tools such as YAKINDU (2014) to help reason about the architecture's ability to support the range of functionality and related quality requirements of the system (CLEMENTS et al., 2002). Hence, after defining the flow expressions, the next activity is the statechart derivation, which later can be used for analysis.

3.5 Activity 5: Statechart derivation and refinement

- Goal: Obtain the statechart and perform the refinements
- Input: Behavioral contextual design goal model
- Steps:

  1: Generate the statechart using the derivation patterns:

  1.1: Create a state for each goal and task following the hierarchy of the design goal model

  1.2: If necessary, create idle states to model situations where the system is waiting for user interaction or for a given context to hold.

  2: Specify transitions in the statechart

- Output: Statechart

The statechart derivation and refinement activity relies on the behavioral contextual design goal model as in input, as indicated in Figure 3.11. The goal of this activity is to obtain the statechart and perform the refinements.

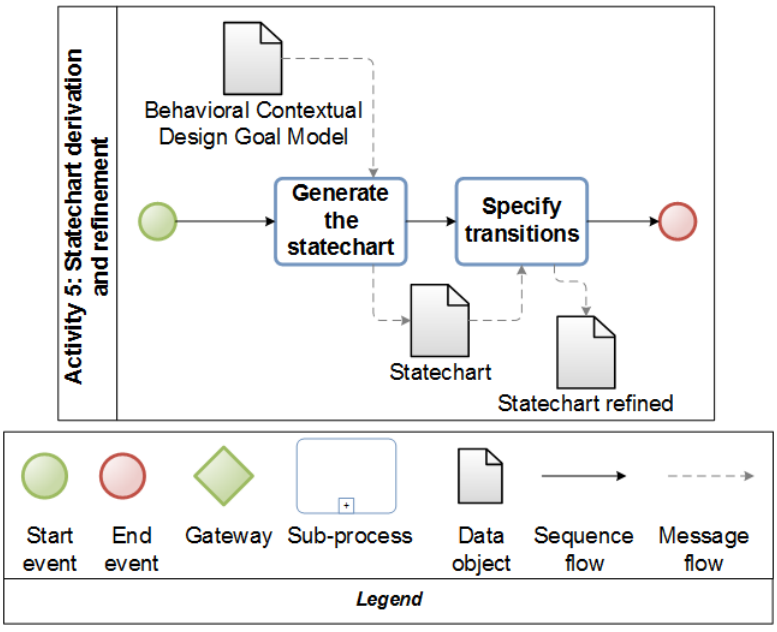


Figure 3.11: Steps of *Statechart derivation and refinement* activity (AUTHOR, 2015).

The flow expressions previously defined are translated into states of the statechart that represents the system's behavior view. We adopted the set of derivation patterns proposed by PIMENTEL et al. (2014). They are related to the different flows that may be expressed (sequential, alternative and concurrent) as well as to their optionality and multiplicity (see Figure 3.12).

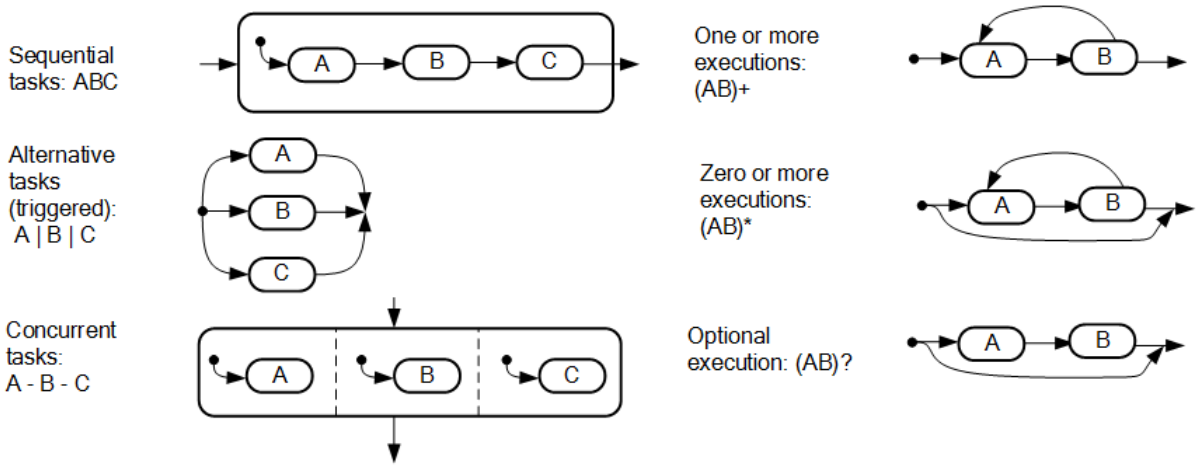


Figure 3.12: Statechart Derivation Patterns (PIMENTEL et al., 2014).

The generation of statechart consists of creating a state for each goal and task starting with the root node. The hierarchy of goals should be considered, therefore, each goal and task in and/or refinements becomes a superstate with its children nodes. We have six types of flows: sequential tasks, alternative tasks, concurrent tasks, optional execution, zero or more executions, one or more executions.

Sequential tasks are represented in the flow expression through the IDs separated by blank space ( ). Figure 3.13 shows an illustration of the statechart derivation of this type of flow from the behavioral design goal model.

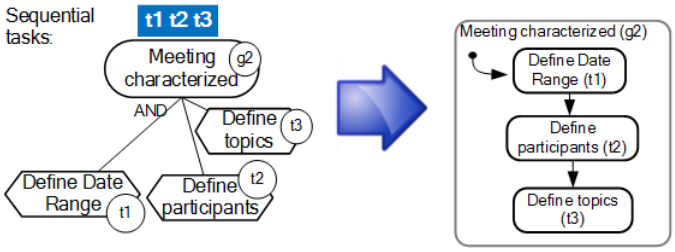


Figure 3.13: Sequential Tasks Pattern (AUTHOR, 2015).

Alternative tasks are represented in the flow expression through the IDs separated by a vertical bar (|). An illustration of the statechart derivation of this type of flow is presented in Figure 3.14.

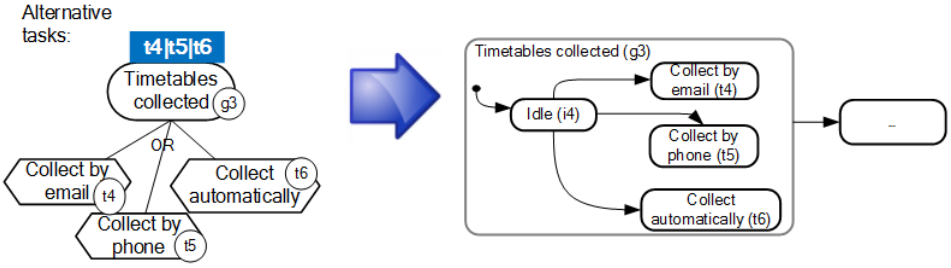


Figure 3.14: Alternative Tasks Pattern (AUTHOR, 2015).

A hyphen (-) should be used to separate the IDs of concurrent tasks in the flow expression as shown in Figure 3.15.

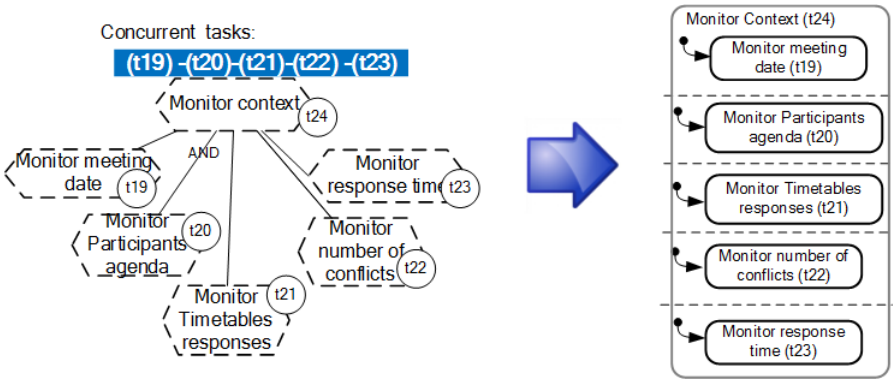


Figure 3.15: Concurrent Tasks Pattern (AUTHOR, 2015).

In a sequence flow, one of the tasks possibly will not be executed. Accordingly, a question mark (?) on the right of the ID in the flow expression should be used to represent this optionality as demonstrated in Figure 3.16.

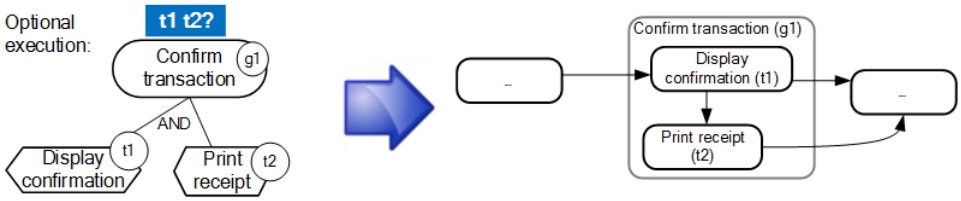


Figure 3.16: Optional Tasks Pattern (AUTHOR, 2015).

An optional task can be executed repeatedly. Hence, we should use an asterisk (*) on the right of the ID in the flow expression to represent that this task can be executed zero or more times. Figure 3.17 shows an illustration of the statechart derivation of zero or more executions tasks from the behavioral design goal model.

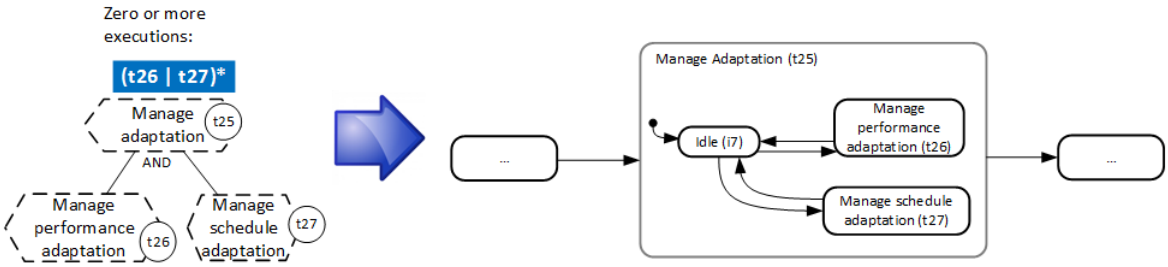


Figure 3.17: Zero or more executions Pattern (AUTHOR, 2015).

Finally, we should use a plus sign (+) on the right of the ID in the flow expression to represent the repetition of tasks as presented in Figure 3.18.

During the statechart derivation it can be necessary to specify more idle states to represent that the system is waiting for some input. These states should be represented though new states and the conditions required to hold should be specified in the transition of these idle states.

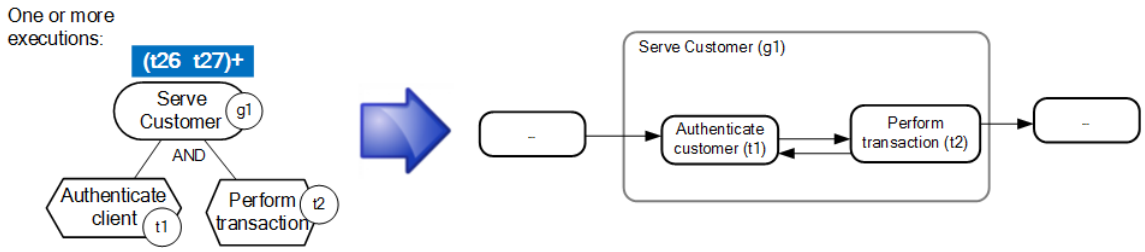


Figure 3.18: One or more executions Pattern (AUTHOR, 2015).

The procedure described above should be carried traversing the behavioral contextual design goal model until it reaches the leaf nodes. After generating the 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 (PIMENTEL et al., 2014): user request, timer, requested by another task, requested by another system and context activation.

Figure 3.19 presents the complete statechart of our running example. The context activation is represented through the context labels (C1, C2...Cn) annotated in the behavioral contextual design goal model (Figure 3.10).

The system starts in *idle state (i1)*. If the user requests login in the system, *Manage Access (t16)* state is executed. In case of failure of the login, the system returns to the *idle state (i1)*, otherwise, it enters in *idle state (i2)*. From *i2*, the user can start to characterize the meeting, so, the system enters in *Meeting Characterized (g2)* superstate in which *Define participants (t2)*, *Define Date Range (t1)* and *Define topics (t3)* states are reached, followed by the *idle state (i2)*.

If the user requests to update the meeting, the system reaches the *Update meeting (t15)* state, later returning to *idle state (i2)*. On the other hand, if collect timetables is requested, the system enters in *Collect timetables (g3)* superstate from *Idle state (i4)* where it should wait C3 or C4 or C5 context to hold. If C3 holds, *Input participants availability (t5)* state is reached, otherwise if C4 holds, *Collect by email (t4)* state is reached, and finally, if C5 holds, *Collect automatically (t6)* state is reached. After that, the system returns to *idle state (i2)*.

If the user requests manually schedule, the system reaches the *Schedule manually (t7)* state. On the other hand, if C1 or C2 context holds, the system enters in *Schedule automatically (t8)* superstate. If C1 holds, *Brute Force Algorithm (t12)* state is reached. Otherwise, if C2 holds, *Heuristics based algorithm (t11)* state is reached.

Therefore, *Characterize meeting (g2)*, *Timetables collected (g3)*, *Schedule defined (g4)*, *Update meeting (t15)* are alternatives states that can be executed zero or more times. All these states are running concurrently to *Monitor Context (t24)* and *Manage Adaptation (t25)* states.

Monitor Context (t24) is a superstate which has five states running concurrently: *Monitor meeting date (t19)*, *Monitor number of conflicts (t22)*, *Monitor Participants agenda (t20)*, *Monitor response time (t23)* and *Monitor Timetables responses (t21)*. These sub-states represent the monitoring of dynamic contextual elements. Hence, when some change occurs in any of these

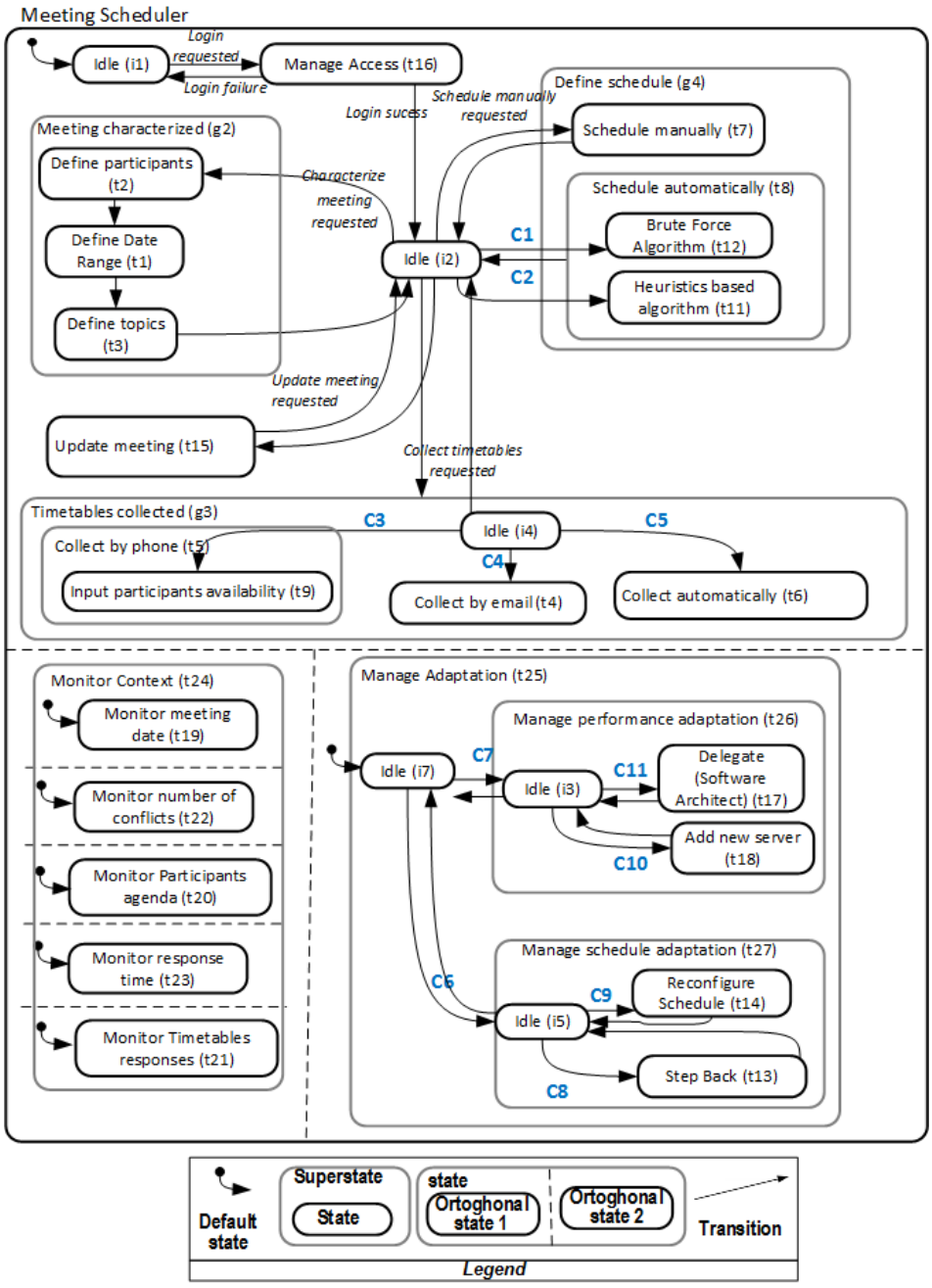


Figure 3.19: Statechart of meeting scheduler example (AUTHOR, 2015).

CEs, some context may hold and thus, the system should take some adaptation action.

Manage Adaptation (t25) is a superstate that represents the adaptation design tasks the system should perform in case of failure of critical requirements. It starts in a *idle state (i7)* and waits until C6 or C7 context holds. If C6 context holds, it enters in the *Manage schedule adaptation (t27)* superstate from *idle (i5)* state. When C9 context holds, the *Reconfigure Schedule (t14)* state is reached, otherwise, if C8 holds, *Step Back (t13)* state is entered. These states are executed as many times as necessary, until the contexts do not hold anymore.

When C7 context holds, the system enters the *Manage performance adaptation (t26)* superstate from the *idle state (i3)*. When C11 context holds, the *Delegate (Software Architect)*

(t17) state is reached. Otherwise, if C10 holds, *Add new server (t18)* state is entered. Similarly these states are executed many times as necessary until the contexts do not hold anymore.

The statechart can be used to requirements validation, hence, the software engineer will be able to check if the system will behave as expected. This model can also be used in the next phase of the software development (implementation) to generate code through tools such as YAKINDU (2014).

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

3.6 Activity 6: Prioritization of variants

- Goal: When more than one context holds prioritize variants
- Input: Behavioral contextual design goal model
- Steps:
  - 1: Define the preferences for variants over each NFR
  - 2: Determine the weights of each NFR
  - 3: Synthesize the results
  - 4: Verify the consistence
- Output: Vector of variants priorities

The system's variants are applicable only if their associated contexts hold. However, in a certain execution, more than one variant may be enabled in the actual context. Therefore, the system has to implement runtime mechanisms to decide on the best choice of variant to adopt. In our work, we are concerned with the satisfaction of the non-functional requirements since they have a impact on the system's behavior. Hence, we consider the variant contribution for the NFR satisfaction when choosing the best one when more than one hold at runtime.

In order to determine the variant contribution, we propose the use of the Analytical Hierarchy Process (AHP)) method (SAATY, 1987). We adopted this method because of its benefits are well described in the literature (BRITO et al., 2007): it is a well-known and accepted method; it is appropriate for handling conflicting concerns problems; it has the ability to quantify subjective judgements; it is capable of comparing alternatives in relation to established criteria; and it provides means to guarantee the logical consistency of the judgements.

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 (SAATY, 1987).

It should be noted that AHP analysis can be performed using a spreadsheet tool, which shows that there is no need for a sophisticated tool support to implement this method. The activities required for NFRs prioritization are depicted in Figure 3.20.

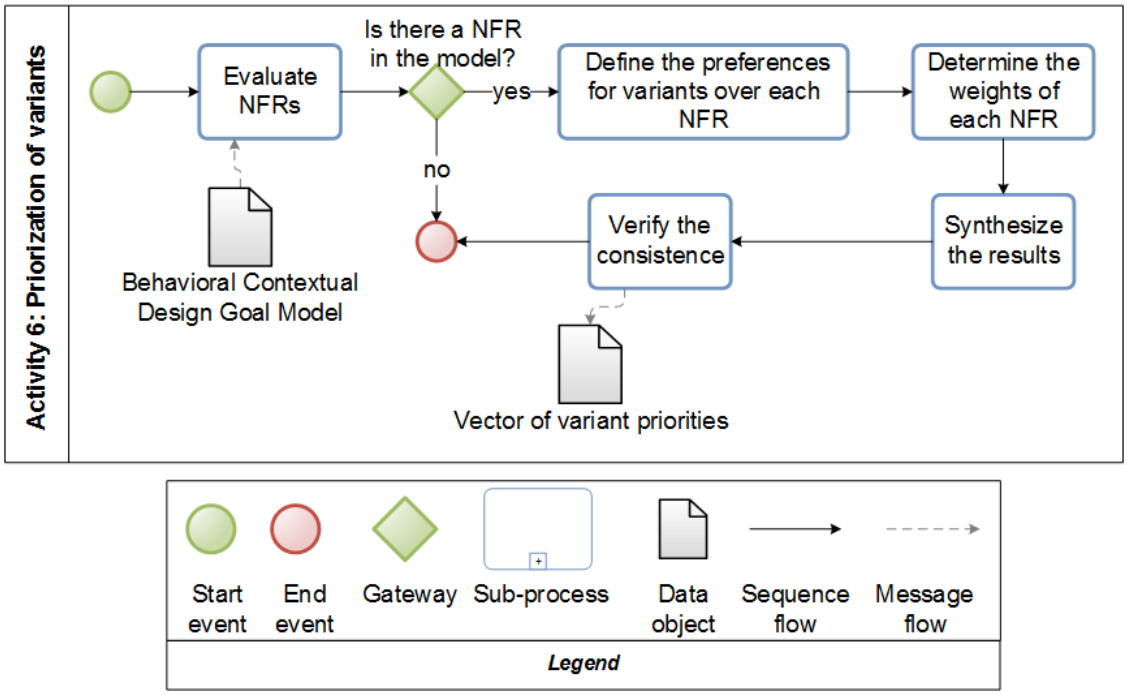


Figure 3.20: Steps of *Prioritization of variants* activity (AUTHOR, 2015).

In our running example, we took in consideration three NFRs (usability, security, performance) and that it is possible that C3, C4, and C5 contexts of Figure 3.10 hold at the same time. Figure 3.21 illustrates the hierarchical tree of this example.

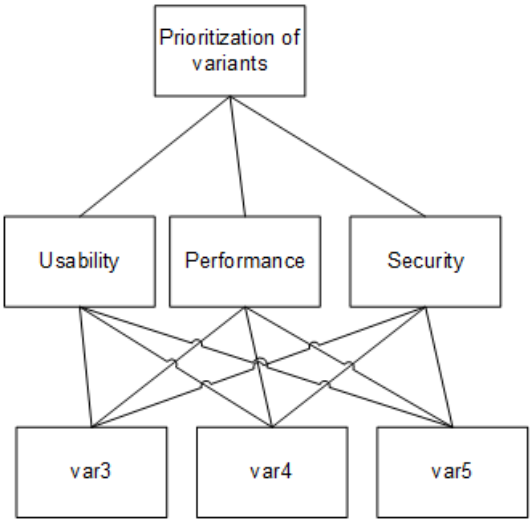


Figure 3.21: AHP hierarchical tree of the meeting scheduler example (AUTHOR, 2015).

The first step of this activity consists in defining the preferences for variants (alternative) over each NFR (criteria). In order to achieve this, we considered the contributions links Make, Help, Hurt and Break that can be presented in the behavioral contextual design goal model. In case of changes in the contribution links that are context-dependent, the new values of the contributions should be considered. The system should update the contribution strength at runtime and perform new analysis of the variants.

The contributions can also be represented, respectively, by ++, +, - and - - (SANTOS, 2013). The variants contribution for each NFR are represented through the contribution links as demonstrated in Table 3.2 for the meeting scheduler example. These contributions values presented in this table are subjective and based on our experience. In a real setting, the stakeholders should be consulted.

Table 3.2: Variants and their contribution for the NFRs of meeting scheduler example (AUTHOR, 2015).

ID	Variant	Performance	Security	Usability
var3	Collect by phone	+	-	
var4	Collect by email	-	+	+
var5	Collect automatically	++	++	++

SANTOS (2013) defined a mapping, presented in Table 3.3, to convert from the variants contribution (demonstrated in Table 3.2) to AHP scale (SAATY, 1987). Note that in the first row, ++ sign has a importance from 1 (when compared to ++) to 9 (when compared to - -). On the other hand, in the last row, the - - sign has an inverse importance varying from 1/9 (when compared to ++) to 1 (when compared to - -).

Table 3.3: Mapping from NFRs Contributions to AHP values (SANTOS, 2013).

	++	+	=	-	- -
++	1	3	5	7	9
+	1/3	1	3	5	7
=	1/5	1/3	1	3	5
-	1/7	1/5	1/3	1	3
- -	1/9	1/7	1/5	1/3	1

The pairwise comparisons between the variants for each NFR is done by creating a matrix for each NFR to compare all values of variants contributions. Hence, matrices n x n (with n as the number of variants) will associate the variants according to their contribution to the selected NFR. In our running example, we have three NFRs, hence we constructed three matrices. The results of these comparisons are presented in Table 3.4, Table 3.6, and Table 3.8.

In order to determine the vector of variants priorities for the Performance NFR, we should calculate the normalized pairwise comparison matrix. Its elements are determined by dividing each element of the comparison matrix (Table 3.4) by the sum of each column. The

Table 3.4: Variant's contributions to the Performance NFR of meeting scheduler example (AUTHOR, 2015).

Performance	var3	var4	var5
var3	1	5	1/3
var4	1/5	1	1/7
var5	3	7	1
Sum	5	9	1.48

normalized pairwise comparison matrix for the variants contributions to the Performance NFR of meeting scheduler example is presented in Table 3.5.

Table 3.5: Normalized pairwise comparison matrix for the variants contributions to the Performance NFR (AUTHOR, 2015).

Performance	var3	var4	var5	Weight
var3	0.24	0.38	0.23	0.28
var4	0.5	0.08	0.10	0.074
var5	0.71	0.54	0.68	0.643

Then, we find the estimated relative priorities by calculating the average of each row of the normalized matrix. Finally, we have the priority vector for the pairwise matrix. The vector of variants priorities for the Performance NFR is [var3 = 0.28 , var4 = 0.074, and var5 = 0.643].

The results of the comparisons of the variant's contributions to the Security NFR of meeting scheduler example are presented in Table 3.6.

Table 3.6: Variant's contributions to the Security NFR of meeting scheduler example (AUTHOR, 2015).

Security	var3	var4	var5
var3	1	1/5	1/7
var4	5	1	1/3
var5	7	3	1
Sum	13	4.2	1.48

The normalized pairwise comparison matrix for variants contributions to the Security NFR is presented in Table 3.7. Accordingly, the vector of variants priorities for the Security NFR is [var3 = 0.074 , var4 = 0.283, and var5 = 0.643].

The results of the comparisons of the variant's contributions to the Usability NFR of meeting scheduler example are presented in Table 3.8.

The normalized pairwise comparison matrix for variants contributions to the Usability NFR is presented in Table 3.9.

Accordingly, the vector of variants priorities for the Usability NFR is [var3 = 0.106 , var4 = 0.26, and var5 = 0.633].

Table 3.7: Normalized pairwise comparison matrix for the variants contributions to the Security NFR (AUTHOR, 2015).

Performance	var3	var4	var5	Weight
var3	0.08	0.05	0.10	0.074
var4	0.38	0.24	0.23	0.283
var5	0.54	0.71	0.68	0.643

Table 3.8: Variant's contributions to the Usability NFR of meeting scheduler example (AUTHOR, 2015).

Usability	var3	var4	var5
var3	1	1/3	1/5
var4	3	1	1/3
var5	5	3	1
Sum	9	4.33	1.53

Table 3.9: Normalized pairwise comparison matrix for the variants contributions to the Usability NFR (AUTHOR, 2015).

Performance	var3	var4	var5	Weight
var3	0.118	0.08	0.13	0.106
var4	0.33	0.23	0.22	0.26
var5	0.56	0.69	0.65	0.633

In order to determine the weights of each NFR, the software engineer should compare all pairs of NFRs and assign a value to each pair using the AHP scale. The results of the pairwise comparisons of the meeting scheduler example with three NFRs (usability, security, performance) are shown in Table 3.10. These assigned values are subjective and based on our experience.

Table 3.10: Pairwise comparison values for the NFRs of the meeting scheduler example (AUTHOR, 2015).

	Usability	Performance	Security
Usability	1	1/7	1/5
Performance	7	1	3
Security	5	1/3	1
Sum	13	1.48	4.2

The normalized pairwise comparison matrix for the NFRs of the meeting scheduler example is presented in Table 3.11.

The NFR priorities vector of the meeting scheduler example is [Performance = 0.643, Security = 0.283 and Usability = 0.074]. Hence, we can notice that Performance is the most critical NFR followed by Security and Usability.

After defining the preferences for variants over each NFR, as well as the weights of each

Table 3.11: Normalized pairwise comparison matrix for the NFRs of the meeting scheduler example (AUTHOR, 2015).

	var3	var4	var5	Weight
var3	0.08	0.10	0.05	0.074
var4	0.54	0.68	0.71	0.643
var5	0.38	0.23	0.24	0.283

NFR, the next step is to synthesize these results. Accordingly, the vectors of variants priority for each NFR are combined into a single matrix. This new matrix is multiplied by the NFR priority vector obtained from the NFR importance matrix to obtain the overall objective (i.e., ranking of variants).

Table 3.12 shows the final ranking in our running example. We can notice that the var5 (*collect automatically*) is the one that contributes mostly for the satisfaction of the NFRs (0.67) followed by var3 (*collect by phone*) with 0.21 and var4 (*collect by email*) with 0.15.

Table 3.12: Final Ranking of meeting scheduler example (AUTHOR, 2015).

	var3	var4	var5	NFR priority
Usability	0.11	0.26	0.63	0.074
Performance	0.28	0.07	0.64	0.643
Security	0.07	0.28	0.64	0.283
Variant priority	0.21	0.15	0.64	

The final step of the prioritization is to check the consistency of the judgments that the software engineer demonstrated during the pairwise comparisons. The logical quality of the decisions is guaranteed by computing the consistency ratio (CR), which measures the consistency of the pairwise comparison judgments. The required procedure to calculate these ratios are described in SAATY (1987). When the consistency ratio exceeds 0.10 appreciably the judgments often need reexamination. This reduces any possible error that might have been introduced during the judgment process (SAATY, 1987).

We used a spreadsheet tool to determine the consistency ratios of the meeting scheduler example. For the pairwise comparison matrix for NFRs (Table 3.10), we obtained a consistency ratio of 0.056. Otherwise, the consistency ratios for the variants contribution for performance, security, and usability were 0.056, 0.056, and 0.033 respectively. These ratios are a good indication that logical consistent judgments were made on all pairwise comparisons, because they are below the required 0.1 threshold (SAATY, 1987).

3.7 GO2S Metamodel

Modeling languages allow the creation of high-level models raising the level of abstraction and hiding implementation details. In order to prevent invalid models, the abstract syntax

of a modeling language must define well-formedness rules that state how the concepts may be legally combined (FIDALGO et al., 2012).

The abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models. This abstract syntax is formalized by means of a metamodel, which also serves as a basis to interchange models with other tools (FIDALGO et al., 2012).

Accordingly, a metamodel describes the concepts of a modeling language, their properties and the legal connections between language elements. Therefore, metamodels are mandatory and important artifacts in the specification of modeling languages since they precisely define how tools and models should work together (FIDALGO et al., 2012).

An abstract syntax model of the behavioral contextual design goal model include concepts such as Nodes, Context and Behavioral Annotations, and Requirements and Design Elements. In addition, there are relationships between concepts, such as Links and Contributions.

Figure 3.22 presents the metamodel of the behavioral contextual design goal model. It has three main entities: *BehavioralContextualDesignGoalModel*, *Node*, and *Link*. *BehavioralContextualDesignGoalModel* is the root entity that corresponds to the drawing area of a behavioral contextual design goal model. For this reason, it can have many instances of *Node* and many instances of *Link*, which cannot exist without *BehavioralContextualDesignGoalModel*.

The behavioral contextual design goal model has requirements and design elements. Accordingly, the GO2S metamodel has two specialized entities for *Node*: *RequirementsElement* and *DesignElement* that has an attribute called *Name* (i.e. the content presented in these elements).

The requirements elements can be goals, tasks, quality constraints and softgoals. Hence, the *RequirementsElement* entity is specialized in four entities that are used to represent the requirements concepts of a goal model: *Softgoal*, *Task*, *QualityConstraint*, and *Goal*.

The design elements of a behavioral contextual design goal model can be design tasks and design constraints. Therefore, the *DesignElement* entity is specialized in two entities: *Design Task*, and *DesignConstraint*, which are used to represent the design concerns that appeared in the design phase.

Since tasks and design tasks can be assigned to an user, the *Task* and *DesignTask* entities have an attribute called *Assignment*. Besides, the *Task*, *Goal* and *Design Task* entities can have behavioral annotations. Hence, they are associated with the *BehavioralAnnotation* entity.

Besides the Node entity, our metamodel has two specialized entities for *Link*, namely *RefinementLink* and *ContributionLink*. Since, a link can have a context annotation, the *Link* entity is associated with the *ContextAnnotation entity* that has an attribute named *Description*.

The *RefinementLink* entity is associated the *Node* entity. A refinement link has one parent and one child. Furthermore, we represented the two types of refinement links (AND, OR-refinements of a behavioral contextual design goal model) through the same class. For this reason, the *RefinementLink* entity has an enumerator attribute named *RefinementType* that

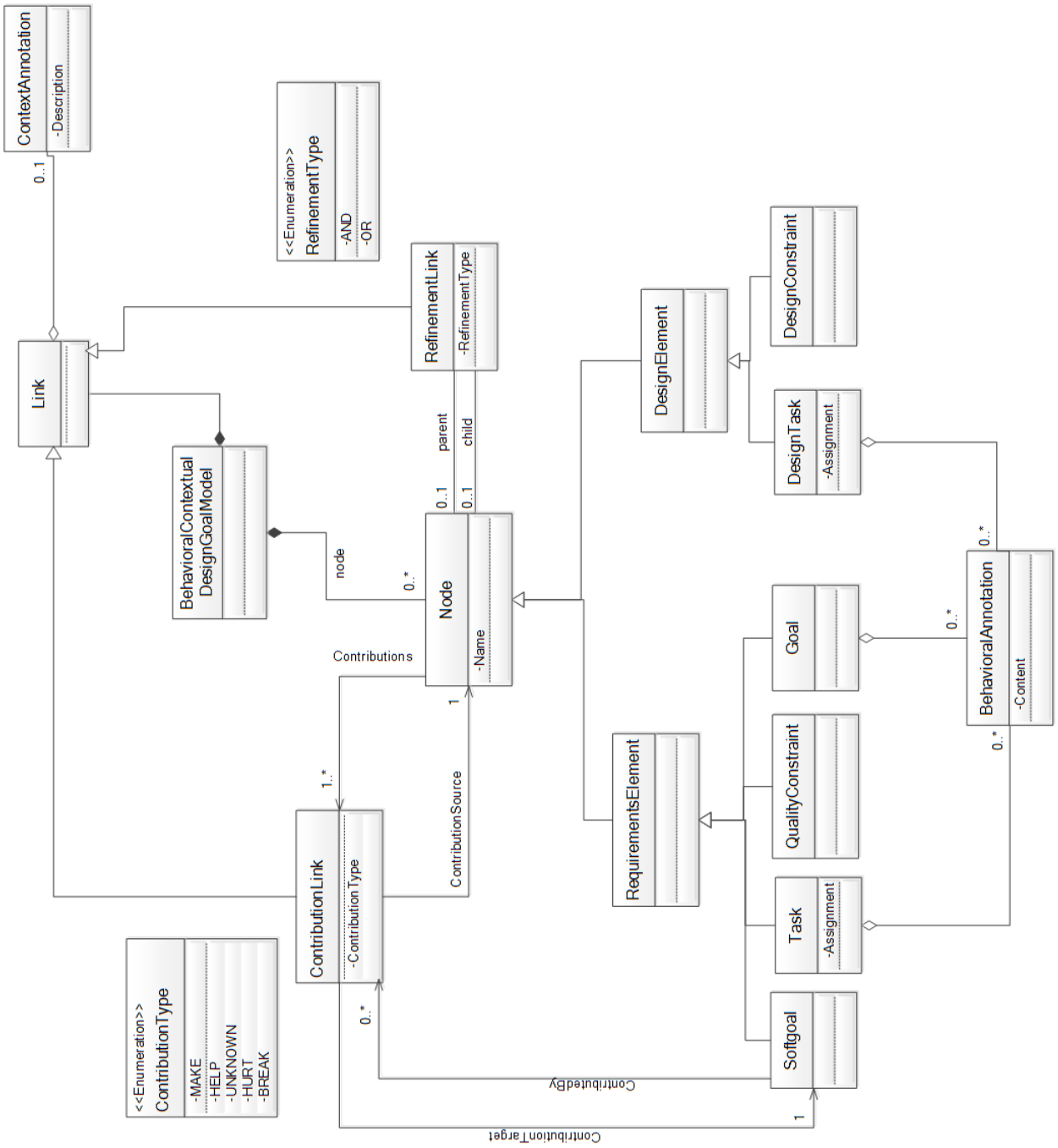


Figure 3.22: GO2S metamodel that relates requirements, design, context and behavioral annotations (AUTHOR, 2015).

encompasses these two possible refinement types.

As we have explained previously, a softgoal has different degrees of satisfaction. With this in mind, nodes (the *Node* entity) contribute to some degree of satisfaction to softgoals (*Softgoal Entity*). Hence, based on the NFR Framework (CHUNG et al., 2000), we propose to represent the contributions from the nodes to satisfy the softgoals through the *ContributionLink* entity. Therefore, a contribution has a *Node* as source and a *Softgoal* as target. In order to specify the contribution degree to satisfy each softgoal, the *ContributionLink* entity has an enumerator

attribute – named *ContributionType* – indicating five possible contribution degrees: Make, Break, Unknown, Help or Hurt.

3.8 Final Considerations

In this section we presented the GO2S, a systematic process for deriving the behavioral view of context-sensitive systems architecture (modeled as statecharts), from system's requirements (modeled as goal 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 described its six activities: *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* presenting a summary of their goals, inputs, steps, and its outputs. We also presented the GO2S metamodel that describes the concepts of the behavioral contextual design goal model, their properties and the legal connections between the elements. In the next chapter, we illustrate the application of the GO2S process through the Znn exemplar.

4

Illustration

In this chapter, we illustrate the application of the GO2S process through the ZNN.com exemplar. It is a problem available in the repository of examples and challenge problems that the software engineering for self-adaptive systems community can use to motivate research, exhibit solutions and techniques, and compare results (CHENG; SCHMERL, 2014). This example has already been used by several other studies (ANGELOPOULOS; SOUZA; PIMENTEL, 2013)(CHENG; GARLAN; SCHMERL, 2009) (LUCKEY et al., 2011).

4.1 System's Description

Znn.com is a news service, based on real sites, that serves multimedia news content to its customers through a website. The business objectives at Znn.com are to serve news content to its customers within a reasonable response time range while keeping the cost of the server pool within its operating budget.

From time to time, due to highly popular events, Znn.com experiences spikes in news requests that it cannot serve adequately, even at maximum pool size. In these cases, response times for user requests might become unacceptable and the system has two possible adaptation strategies: enlisting new servers to divide the load of requests or switch from multimedia to text-mode to make each request quicker to respond.

However, these strategies may cause problems in two other requirements of this system: (i) the website managers would like to run the system at the lowest cost possible and adding new servers costs money; (ii) second, the users would like to see news with high content fidelity (i.e., high presentation quality), preferring multimedia over simple text.

We borrow the goal model for this exemplar by (ANGELOPOULOS; SOUZA; PIMENTEL, 2013) (see Figure 4.1). It is important to note that the model does not represent complete requirements for a news service (which would include concerns such as adding news, searching, managing advertisement, etc.), but concentrates on the adaptation scenario.

The challenge of such systems is to achieve their mandate even when they operate under critical conditions. The difficulty lies in taking the right decision at the right time, in the

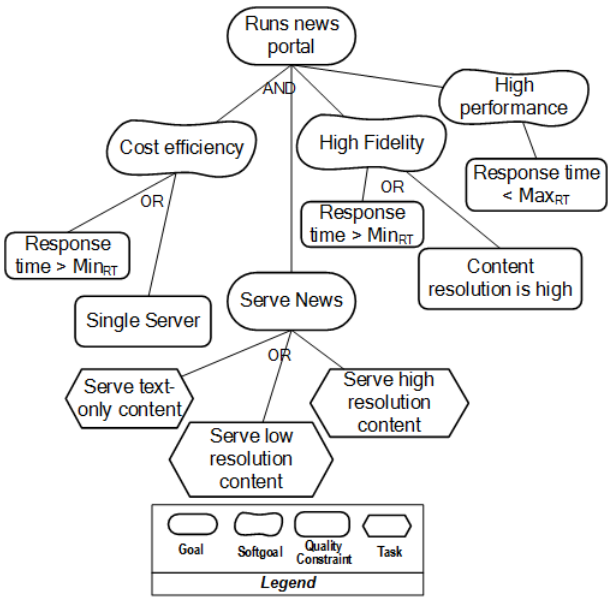


Figure 4.1: Goal model of the ZNN.com exemplar (ANGELOPOULOS; SOUZA; PIMENTEL, 2013).

sense that the problem should be detected promptly and the most efficient strategy to stabilize operation should be applied immediately. Under such circumstances, human intervention can be insufficient and automated mechanisms are required to carry out both decision-making and adaptation.

When ZNN.com experiences spikes in news requests it may cause the failure of the *High performance* softgoal. In this case, four adaptations are possible (ANGELOPOULOS; SOUZA; PIMENTEL, 2013):

a) **Simple Reduce Response Time:** In case a client experiences response time above a predefined threshold, then the fidelity is lowered by one step. In case response time remains high, fidelity is decreased again one more step;

b) **Smarter Reduce Response Time:** If an unacceptable percentage of clients experiences high response time, then enlist one server, then enlist another server and finally lower fidelity by one step. Repeat twice the last two actions until response time is restored;

c) **Reduce Overall Cost:** If server cost is higher than a threshold value, then reduce the number of servers by one. If response time is low and cost remains high repeat the previous action, until cost is returned to normal;

d) **Improve Overall Fidelity:** If content fidelity level is below threshold then raise fidelity of all servers by one step. If response time is low and fidelity remains low then raise fidelity level one more step.

In the next sections we describe the results of applying each activity of GO2S process in this exemplar.

4.2 Activity 1: Construction of design goal model

The first activity of GO2S process has the objective of refining a goal model with new design elements. Hence, the software engineer should check if there is critical design task or design quality constraint not previously identified in the requirements phase that is necessary for the system. Moreover, the DGM also allows the definition of assignments for its tasks. These assignments are expressed by labels below the assigned element.

We are concerned with the adaptation scenario in the Znn example. The goal model provided by ANGELOPOULOS; SOUZA; PIMENTEL (2013) is appropriate, so there was no need to add new design tasks and constraints. All critical requirements were identified and modeled as shown in Figure 4.1.

The OR refinement of DGM introduces alternatives into the model accommodating many/all possible functionalities that fulfill stakeholder goals. Hence, we need to consider variability of the model as indicated in next activity.

4.3 Activity 2: Specification of contextual variation points

In this activity, the (contextual) variation points – VP are annotated in the design goal model to visually specify the effects of context in the system's behavior as shown in Figure 4.2.

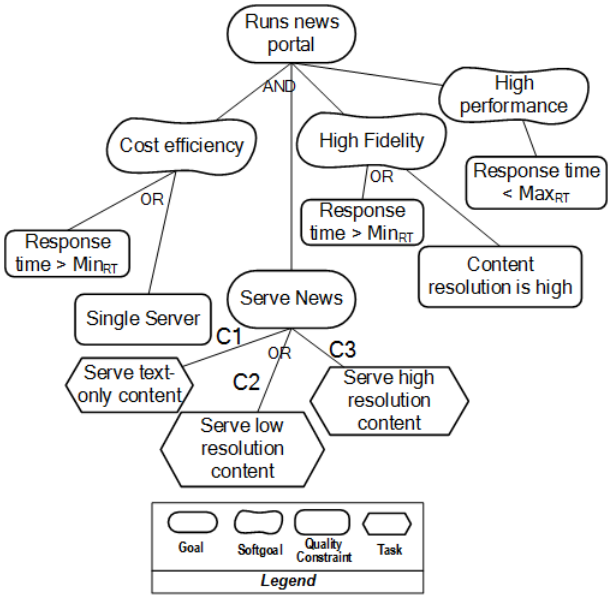


Figure 4.2: Contextual DGM of the ZNN exemplar (AUTHOR, 2015).

Considering that the news can be showed in three different ways: only-text context, low resolution context and high resolution content, we identified three variation points at the

or-refinement of *Serve news* goal. These VPs are annotated in the contextual goal model trough the C1, C2 and C3 labels.

The contextual VPs should be specified following the notation of ALI; DALPIAZ; GIORGINI (2010). Hence, each context specified in the contextual design goal model must be refined through a set of statements and facts. These refinements are required in order to enable the system to check at runtime if context is enabled. Figure 4.3 presents the refinements of contexts C1, C2 and C3.

C1 context holds when the *st1=softgoal high performance keeps failing* statement is true. This refinement is evaluated through the *f1=the response time is higher than [MAXRT] for more than a second* fact. If this fact is true, then the statement holds. Therefore, we can infer that C1 context is true and then the *Serve text-only context* task can be executed.

C2 context holds when the *f3=Content fidelity level is below than [threshold]* fact or the *st1=softgoal high performance is failing* statement are true. If *f1=the response time is higher than [MAXRT] in less than a second* fact or *f2=the number of servers is equal to 3* fact is true, the statement holds. Therefore, we can infer that C2 context is true and then the *Serve low resolution content* task can be executed.

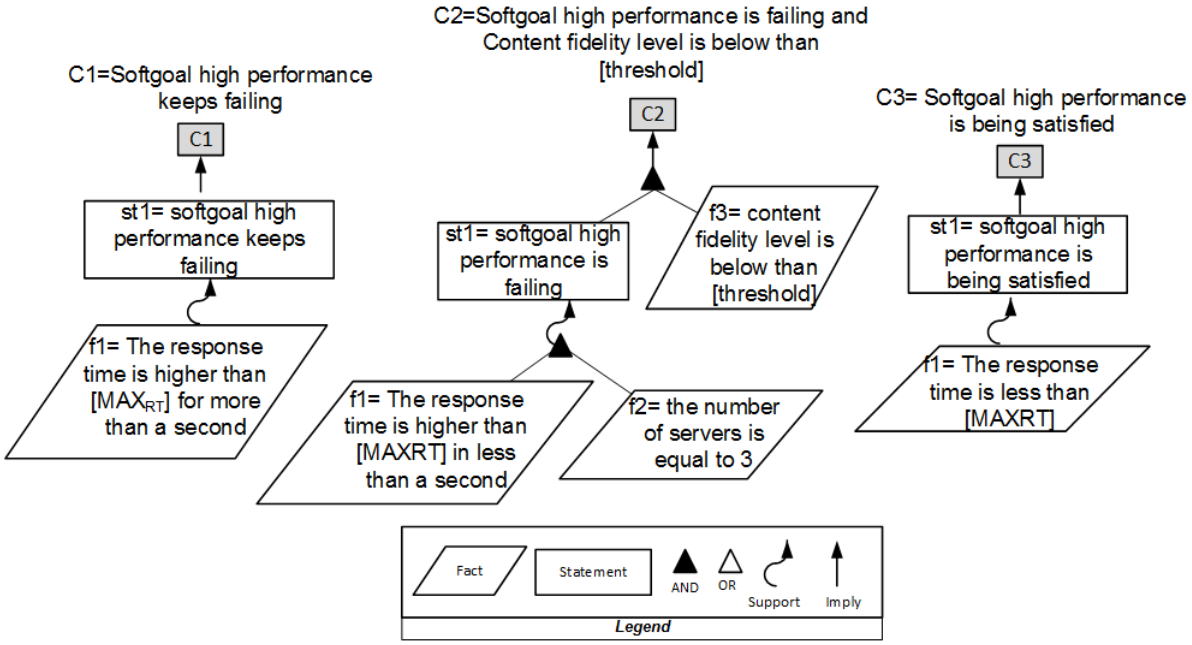


Figure 4.3: Refinements of contexts of ZNN exemplar of activity 2 (AUTHOR, 2015).

C3 context holds when the *st1=softgoal high performance is being satisfied* statement is true. This refinement is evaluated through the *f1=the response time is less than [MAXRT]* fact. If it holds, the statement is true. Therefore, we can infer that C3 context is true and then the *Serve high resolution content* task can be executed.

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 support the required adaptation.

4.4 Activity 3: Specification of adaptation and monitoring

Context-sensitive systems have the ability to adapt themselves in order to provide personalized services for their users, i.e. to enable/disable functions. Therefore, we propose to use this characteristic to deal with the requirements adaptation when a goal fails. In order to achieve this, we refine the contextual design goal model with elements necessary for the specification of adaptation design tasks as well as the monitoring of the facts.

In the Znn examplar, we observed that the system has to adapt itself when the *Performance* softgoal fails. Accordingly, the *Manage Performance Adaptation (t9)* design task was added and its adaptation tasks are *Add new server (t10)* and *Remove a server (t11)*. These adaptation design tasks, the context annotations as well as the tasks responsible for monitoring the contextual elements are presented in Figure 4.4.

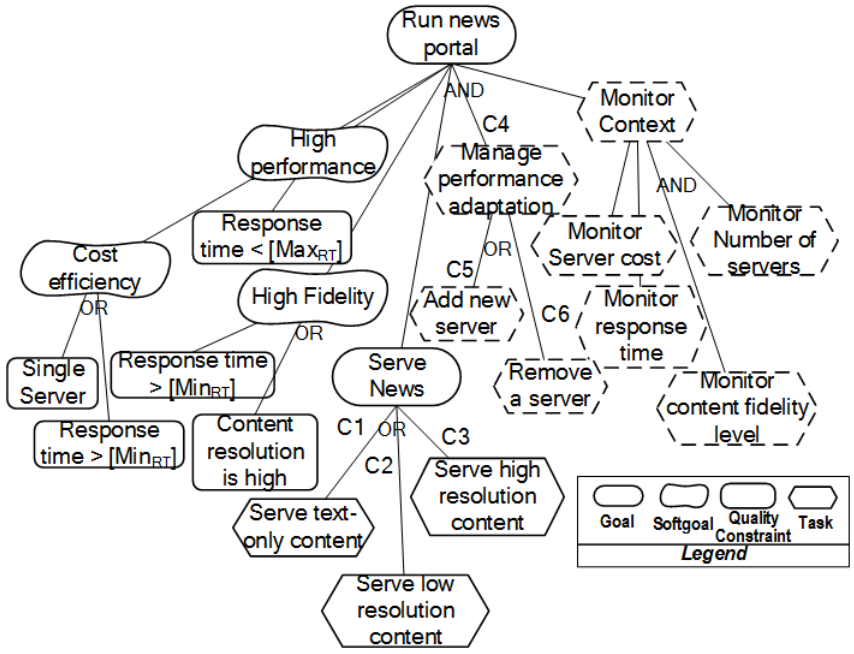


Figure 4.4: Contextual DGM of ZNN refined with adaptation elements (AUTHOR, 2015).

Figure 4.5 depicts the refinements of C4 and C5 contexts. C4 context holds when the *st1=it is necessary to add a new server* statement is true. This statement holds if *f1= [x]% of clients experiences response time higher than [MAXRT]* fact or *f2= the number of servers is less or equal to 3* fact holds. When the C4 context is true, the *Add new server* design task is executed.

On the other hand, C5 context holds when the *st2=it is necessary to remove a server* statement is true. This statement holds if *f1= server cost is higher than [threshold] value* fact or *f2= response time is less than [MAXRT] and cost remains higher than [threshold] value* fact hold. When the C5 context is true, the *Remove a server* design task is executed.

The refinements of each context, annotated in Figure 4.5, were used to identify the contextual elements. Thus, the elements in the znn.com exemplar identified were *response time*,

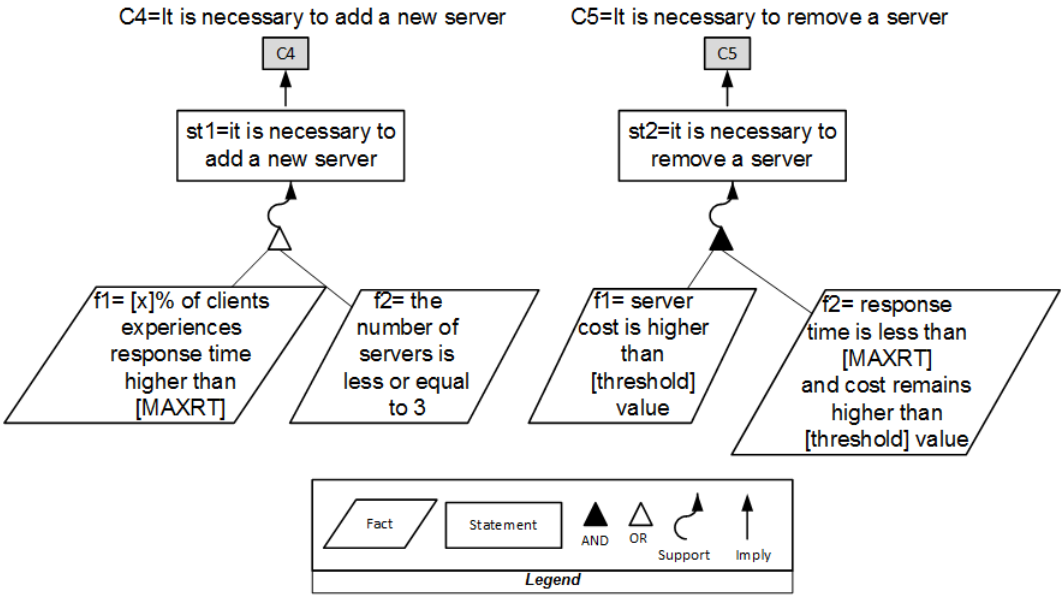


Figure 4.5: Refinements of contexts of ZNN exemplar of activity 3 (AUTHOR, 2015).

number of servers, *server cost* and *content fidelity level*.

Therefore, we added the *Context Monitored* goal in the root goal to represent the monitoring of these contextual elements and consequently the system's context. Thereafter, we added a design task for monitoring each contextual element as depicted in Figure 4.5.

After all contexts, that influence the requirements are refined, and the contextual elements that need to be monitored are identified, the next step is to identify the equipments/technologies needed to monitor these contextual elements (see Section 2.4). In this examplar, the only technology needed to monitor the contexts annotated in the contextual goal model is the information stored in a database. The description of contexts and the equipment/technology are presented in Table 4.1.

Table 4.1: List of contexts of Znn example and the equipment/technology needed to monitor them (AUTHOR, 2015).

	Description	Variation Point Type	Equipment/Technology
C1	Softgoal high performance keeps failing	OR	Mechanism to information storage.
C2	Softgoal high performance is failing and Content fidelity level is below than [threshold]	OR	Mechanism to information storage.
C3	Softgoal high performance is being satisfied	OR	Mechanism to information storage.
C4	It is necessary to add a new server	OR	Mechanism to information storage.
C5	It is necessary to remove a new server	OR	Mechanism to information storage.

The outputs of this activity are the contextual design goal model refined and contexts

refinements. Having defined the adaptations strategies and the contextual elements that need to be monitored, we can now move on to specify the order of execution of tasks and goals and how they interact. For this, we rely on flow expressions.

4.5 Activity 4: Specification of flow expressions

In this activity, the contextual design goal model is refined with flow expressions that represents the execution order of elements in the model. The output of this activity is the behavioral contextual goal model which is the contextual goal model annotated with flow expressions.

The behavioral contextual design goal model of the Znn exemplar is presented in Figure 4.6. Its flow expression is *(i1 g2*)- t4 – t9*

Thus, from the *idle state (i1)*, the system executes the *Serve News (g2)* state zero or more times. Besides, the *Manage Performance Adaptation (t9)* and *Monitor Context (t4)* are states running concurrently with all states.

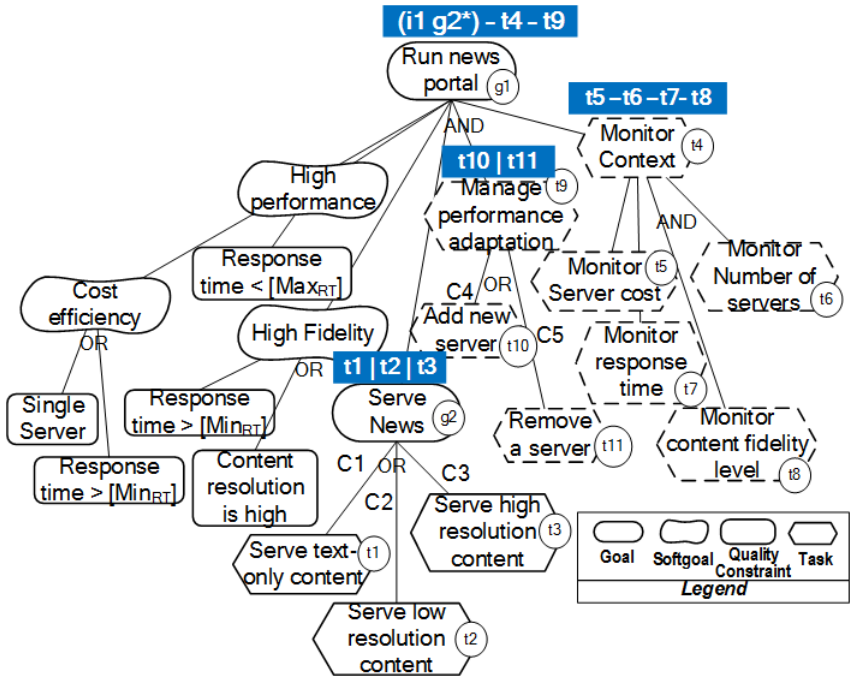


Figure 4.6: Behavioral Contextual DGM of Znn Exemplar (AUTHOR, 2015).

The behavior of context-sensitive system can be simulated to help to reason about the architecture's ability to support the range of functionality and related quality requirements of the system. Hence, after defining the flow expressions, the next activity is to derive a statechart that later can be executed/simulated.

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

4.6 Activity 5: Statechart derivation and refinement

The statechart derivation and refinement activity uses the behavioral contextual goal model as in input. The goal of this activity is to obtain the statechart and perform the refinements.

The flow expressions previously defined are translated into states of the statechart that represents the system's behavior view. We adopted the set of derivation patterns proposed by PIMENTEL et al. (2014) related to the different flows that may be expressed (sequential, alternative and concurrent) as well as to their optionality and multiplicity (see Figure 3.12). The complete statechart of Znn example is shown in Figure 4.7.

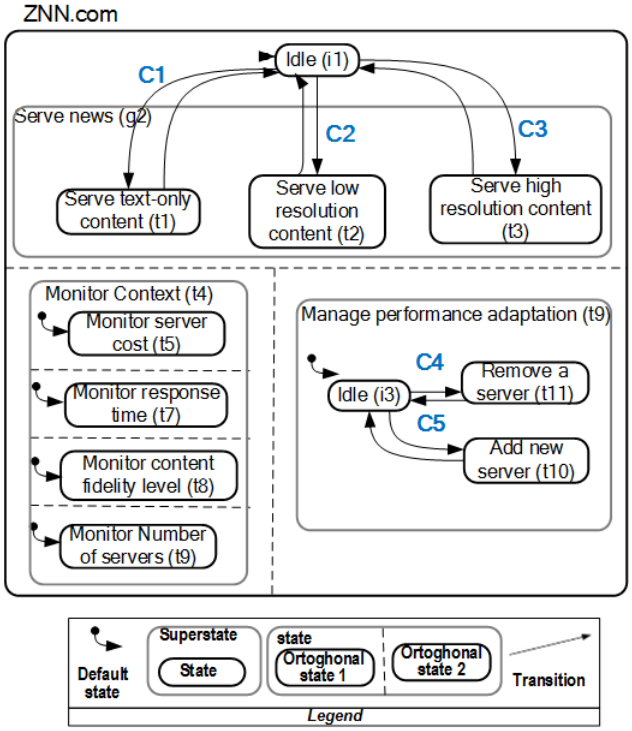


Figure 4.7: Statechart of Znn Exemplar (AUTHOR, 2015).

4.7 Activity 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 enabled in the actual context. In our work, we take in consideration the impact of the NFRs in the system's behavior when choosing the best variant. Hence, we consider the variant contribution to the NFR satisfaction.

In order to determine the variant contribution, we propose the use of the Analytical Hierarchy Process (AHP) method. With this method, we obtain a ranking of variants (alternatives) that the best one contributes to the satisfaction of NFRs (criteria).

In Znn example, we considered three NFRs (Cost efficiency, High performance, High Fidelity) and that C1, C2, and C3 contexts of Figure 4.6 hold at the same time. The first step of

this activity consists of defining the preferences for variants over each NFR. In order to achieve this, we considered the Make, Help, Hurt and Break contributions links represented, respectively, by ++, +, - and - -.

For the Znn examplar, the variants contribution for each NFR are represented by the contribution links in Table 4.2. These contributions values presented in this table are subjective and based on our judgement and experience. Of course, in a real setting the stakeholders should be consulted.

Accordingly, *Serve text-only context* variant makes *Cost efficiency* and *High performance*, but breaks *High Fidelity*. *Serve low resolution content* variant contributes positively to *Cost efficiency* and *High performance*, but hurts *High Fidelity*. Finally, *Serve high resolution content* variants hurts *Cost efficiency*,breaks *High performance* and makes *High Fidelity*.

Table 4.2: Variants and their contribution for the NFRs of Znn example (AUTHOR, 2015).

ID	Variant	Cost efficiency	High performance	High Fidelity
var1	Serve text-only context	++	++	- -
var2	Serve low resolution content	+	+	-
var3	Serve high resolution content	-	- -	++

In order to convert from the variants contribution (as depicted in Table 4.2) to AHP scale we used the mapping proposed by SANTOS (2013). The next step is to perform the pairwise comparisons between the variants for each NFR. Accordingly, it is necessary to create a matrix for each NFR to compare all values of variants contributions. Given that, in the Znn example, there are three NFRs, three matrices are required. The results of these comparisons are presented in Table 4.3, Table 4.4, Table 4.5.

Table 4.3: Variant's contributions to the Cost efficiency NFR of Znn exemplar (AUTHOR, 2015).

Cost efficiency	var1	var2	var3
var1	1	3	7
var2	1/3	1	5
var3	1/7	1/5	1

The variants priority vector for the Cost efficiency NFR, after solving for the main eigenvector, is [var1 = 0.643 , var2 = 0.283, and var3 = 0.074].

Table 4.4: Variant's contributions to the High performance NFR of Znn exemplar (AUTHOR, 2015).

High performance	var1	var2	var3
var1	1	3	9
var2	1/3	1	7
var3	1/9	1/7	1

The variants priority vector for the High performance NFR, after solving for the main eigenvector, is [var1 = 0.649 , var2 = 0.295, and var3 = 0.057].

Table 4.5: Variant's contributions to the High Fidelity NFR of Znn exemplar (AUTHOR, 2015).

High Fidelity	var1	var2	var3
var1	1	1/3	1/9
var2	3	1	1/7
var3	9	7	1

The variants priority vector for the High Fidelity NFR, after solving for the main eigenvector, is [var1 = 0.069 , var2 = 0.155, and var3 = 0.777].

In order to determine the weights of each NFR, we should compare all pairs of NFRs and assign a value to each pair using the AHP scale. The results of the pairwise comparisons of znn example with three NFRs (Cost efficiency, High performance, High Fidelity) are shown in Table 4.6. Recall that these assigned values are subjective and based on our experience. In a real setting the stakeholders should be consulted.

Table 4.6: Pairwise comparison values for Znn example (AUTHOR, 2015).

	Cost efficiency	High performance	High Fidelity
Cost efficiency	1	1/3	5
High performance	3	1	7
High Fidelity	1/5	1/7	1

According to this method, the NFR priority vector is obtained by solving for the main eigenvector of the matrix followed by the normalization of the result. The NFR priority vector of Znn examplar is [High performance = 0.643, Cost efficiency = 0.283 and High Fidelity = 0.074]. We can notice that High performanc is the most critical NFR, followed by Cost efficiency and High Fidelity.

After defining the preferences for variants over each NFR, as well as the weights of each NFR, the next step is to synthesize these results. Accordingly, the variants priority vectors for each NFR are combined into a single matrix. This new matrix is multiplied by the NFR priority vector obtained from the NFR importance matrix to obtain the overall objective (i.e., ranking of variants).

Table 4.7 shows the final ranking of Znn example. We can notice that the var1 (*Serve text-only context*) is the one that contributes mostly for the satisfaction of the NFRs (0.61) followed by var2 (*Serve low resolution content*) with 0.28 and var3 (*Serve high resolution content*) with 0.12.

The final step is the evaluation of the consistency of the judgments that the software engineer used during the pairwise comparisons. We used a spreadsheet tool to determine the consistency ratios. For the pairwise comparison matrix for NFRs (Table 4.6), we obtained a

Table 4.7: Final Ranking of Znn example (AUTHOR, 2015).

	var1	var2	var3	NFR priority
Cost efficiency	0.64	0.28	0.07	0.28
High performance	0.65	0.29	0.06	0.64
High Fidelity	0.07	0.15	0.78	0.07
Variant priority	0.61	0.28	0.12	

consistency ratio of 0.056. Otherwise, the consistency ratios for the variants contribution for Cost efficiency, High performance, and High Fidelity were 0.056, 0.07, and 0.071 respectively. These ratios are a good indication that logical consistent judgments were made on all pairwise comparisons, because they are below the required 0.1 threshold (SAATY, 1987).

4.8 Final Considerations

In this chapter, we illustrated the application of the GO2S process in the Znn exemplar. In the next chapter we will describe the controlled experiment we conducted to evaluated the GO2S process empirically.

5

Evaluation

Experiments are appropriate to investigate different aspects of a research such as (WOHLIN et al., 2012): Confirm theories, i.e. to test existing theories; Confirm conventional wisdom, i.e. to test people's conceptions; Explore relationships, i.e. to test that a certain relationship holds; Evaluate the accuracy of models, i.e. to test that the accuracy of certain models is as expected; and to Validate measures, i.e. to ensure that a measure actually measures what it is supposed to.

In order to evaluate our proposal we designed a controlled experiment. We conducted a multi-test within an object study since we examined a single object (the Smart Home System) across a set of subjects. This empirical method can be used when it is possible to control those using the approach method, and when and where they are used (WOHLIN et al., 2012). Hence, it allows the control of, for example, subjects, objects and instrumentation. This ensures that we are able to draw more general conclusions.

Experimentation is a labor-intensive task and is not simple; therefore, we had to prepare, conduct and analyze the experiment properly. In order to make sure that the proper actions were taken to ensure a successful experiment, we followed the framework proposed by WOHLIN et al. (2012) for performing experiments in software engineering. The authors propose five activities in order to perform an experiment: *Scoping, Planning, Operation, Analysis & interpretation,* and *Presentation & package*. The results obtained in each activity are described in the following subsections.

5.1 Scoping

The scoping specifies the motivation for performing the experiment. Hence, we began setting its objectives. The goal of our experiment is summarized in Table 5.1.

After the definition of the experiment scope, the next activity is the *planning* (WOHLIN et al., 2012).

Table 5.1: Goal of the experiment.

Analyze	the GO2S process for deriving statecharts from goal models of context-sensitive systems.
For the purpose of	evaluation.
With respect to	the time to implement, syntactic correctness, structural complexity, behavioral similarity and cognitive complexity.
From the point of view of	software engineers.
In the context of	students of a requirements engineering undergraduate and graduate course, with some industry expertise, implementing the GO2S process in an example.

5.2 Planning

Planning defines how the experiment should be conducted. It is divided into six steps: *Context Selection, Hypothesis formulation, Variables selection, Selection of subjects, Experiment design, Instrumentation* and *Validity evaluation*. The steps are described in details in next subsections.

5.2.1 Context selection

The context of our experiment is students of undergraduate, master's and doctor's degree from a Requirements Engineering course at an university. The subjects were eighteen students enrolled in the course. The experiment was run off-line (not in an industrial software development environment). The experiment was specific, since it focuses on the following metrics to evaluate the generated statecharts: time to implement, syntactic correctness, structural complexity, behavioral similarity, and cognitive complexity described in Section 5.2.3. The ability to generalize from this specific context is further elaborated in Section 5.6 where we discuss threats to the experiment.

5.2.2 Hypotheses formulation

The main hypothesis is the null hypothesis that states there is no difference between using or not the GO2S process. Therefore, the study tries to reject this hypothesis. There are fourteen null hypotheses, one for each metric the study analyzes.

Null hypothesis (H01): The implementation time using the GO2S process is **not different** than those not using the process.

Null hypothesis (H02): The syntactic correctness using the GO2S process is **not different** than those not using the process.

Null hypothesis (H03): The structural complexity using the GO2S process is **not different** than those not using the process.

Null hypothesis (H04): The behavioral similarity using the GO2S process is **not different** than those not using the process.

Null hypothesis (H05): The GO2S is not easy to understand.

Additionally, alternative hypotheses were defined to be accepted when the corresponding null hypothesis is rejected.

Null hypothesis (H11): The implementation time using the GO2S process is **smaller** than those not using the process.

Null hypothesis (H12): The syntactic correctness using the GO2S process is **smaller** than those not using the process.

Null hypothesis (H13): The structural complexity using the GO2S process is **smaller** than those not using the process.

Null hypothesis (H14): The behavioral similarity using the GO2S process is **higher** than those not using the process.

Null hypothesis (H15): The GO2S is easy to understand.

5.2.3 Variables selection

In the design of experiments, we have to consider what independent variables or factors are likely to have an impact on the results. In our experiment the independent variable was the GO2S process.

On the other hand, we considered five dependent variables, based on the metrics related to evaluate our process: time to implement, syntactic correctness, structural complexity, behavioral similarity, and cognitive complexity. These metrics were already used in the literature to evaluate behavioral models (DIJKMAN et al., 2011) (MIRANDA; GENERO; PIATTINI, 2005). They are described as following:

a) **Implementation time:** the time that subjects spent to develop the statecharts measured in minutes. This metric was used to investigate if there was a significant difference in time spent using the GO2S process.

b) **Syntactic Correctness:** we measured this variable through the number of syntactic errors and warnings indicated by the YAKINDU (2014) modeling tool used in our experiment. This metric was used to investigate how well the subjects *of each group* learned the statechart language as well as to verify if the groups were well balanced.

c) **Structural complexity:** this variable is determined by the different elements that compose the model (MIRANDA; GENERO; PIATTINI, 2005), such as states, transitions, activities, etc. Hence, we evaluated the structural complexity of statecharts through different metrics: the number of super states, orthogonal states, idle states, final state, simple states, state transitions, choice, variables and actions. High structural complexity has an impact on the cognitive complexity of statecharts (MIRANDA;

GENERO; PIATTINI, 2005). This metric was used to investigate the structural complexity of the statecharts obtained using (or not) the GO2S process.

d) **Cognitive complexity:** this metric can be defined as the mental burden that the persons have to deal with the process. Hence, the cognitive complexity of each activity of the GO2S process was evaluated through the subject's opinion about the steps and notations used in the GO2S process. Thus, we applied an anonymous questionnaire with different assertions (see Table 5.10) about the GO2S process. The subjects had to choose an option using the following scale: Totally Disagree, Disagree, Indifferent, Agree, and Totally Agree. This metric was used to investigate if the GO2S process were specified in a understandable way.

e) **Behavioral similarity:** each software engineer constructs a model according his experience and knowledge. Accordingly, the behavioral similarity intends to explore the behavior of different models despite their structural differences (DIJKMAN et al., 2011). We measured the behavioral similarity analyzing if the statechart, produced by the control and experimental groups, behaves as expected through the percentage of the number of functionalities modeled as described in the requirements model in relation of the total number of functionalities. This metric was used to check if the behavioral similarity of the statecharts produced of the experimental group was higher than those of control group.

5.2.4 Selection of subjects

According to WOHLIN et al. (2012), the selection of subjects is also called a sample from a population. In our experiment, we performed convenience sampling: the nearest and most convenient persons are selected as subjects.

In order to identify the impacts of the use of the GO2S process, the subjects were divided in two groups with nine subjects each: the subjects of one group generating statecharts using the proposed process (experimental group), whereas the other subjects did not use the proposed process (i.e. they belonged to the control group). This distribution of the subjects in the two groups was performed randomly.

5.2.5 Experiment Design

In our experiment, we compared two treatments: the use or not of the GO2S process. Therefore, the design of our experiment was classified as *one factor* with *two treatments* being of the type completely randomized design. The design setup uses the same objects for both treatments and assigns the subjects randomly to each treatment. Each subject uses only one treatment on one object (WOHLIN et al., 2012). Since we had the same number of subjects per treatment, the design was balanced.

5.2.6 Instrumentation

The instruments for an experiment are of three types, namely objects, guidelines and measurement instruments (WOHLIN et al., 2012). Therefore, for each subject, we prepared a set of materials (see the Appendix) to be used in the experiment as described in the following subsections.

5.2.6.1 Experimental object

The subjects received a specification of a *Smart Home System* designed to make life easier for people with dementia problems and provide continuous care about them to ensure their safety and comfort. To this end, the *Smart Home* had to act in response to the context. In this experiment, we used a simplified version of the system adapted from description available at ALI (2010).

5.2.6.2 Guidelines

The subjects were aware that their data would be used by the experimental study. The experimental group that applied the GO2S process also received a reference guide with a summary of the activities and the notations used by our process. In addition to the reference guide (presented in Appendix), the subjects of the experimental group also attended 4 hours of course to learn our process.

5.2.6.3 Measurement instruments

All subjects answered a pre-experiment questionnaire to inform their profile and experience in system modeling. Moreover, the subjects also filled a questionnaire post-experiment to express their opinions about each process activity. We collected their statecharts and all the material used in the experiment for evaluation.

5.3 Operation

In the operational phase of the experiment, the treatments were applied to the subjects. This phase consists of three steps: preparation, execution and data validation.

5.3.1 Preparation

As previously mentioned, this study was performed using subjects enrolled in undergraduate and graduate course and consisted of two trainings to the subjects. In the first one, we provided classes about the goal model, the statecharts theory and the yakindu tool to the participants of both groups (control and experimental group).

We took extra care to check if the subjects had indeed learned adequately the statecharts theory it was a pre-requisite to participate in the experiment. They were asked to work on a course project in which they had to model a statechart from a given requirements specification (provided by the instructors). Besides the project modeling, we also performed an oral argumentation with each subject, to check the quality of their project handout and aquaintacy/familiarity with the statechart theory (HAREL, 1987) and modeling tool (YAKINDU, 2014).

The second training consisted in presenting/teaching the students the GO2S process to the subjects of the experimental group (9 subjects) and performing a dry run to give the subjects a chance to familiarize with activities of the process. The control group also had an opportunity to exercise the statecharts language in a domain different from the one of the experiment.

Finally, the experiment was executed. The time spent in each activity was the following:

- Classes about goal model, statecharts theory and tool to all subjects: 8hrs
- Oral argumentation: 3hrs
- Training about the process to the subjects of the experimental group: 4hrs
- Dry run with all subjects: 4hrs
- Experiment: 3hrs

The time spent to execute the experiment was 22hrs. Besides this execution time, there was the time spent in meetings for decision-making, the preparation of the project, answering questions of students and correcting all projects. In addition, there was the time spent on preparing slides, the material used in the experiment and the time required to analyze the results. Hence, the total time was approximately 132 hours.

5.3.2 Execution

The experiment was carried out in a computer laboratory with the two groups. Each subject of the control group developed a statechart for the *Smart Home System* from the requirements specification (presented in Appendix). On the other hand, each subject of the experimental group developed a statechart following the GO2S process. We asked the subjects of the experimental group to perform the activities 1 to 5 of the GO2S process, since the last one concerns with the variants prioritization that was not possible to compare to the control group. All subjects had to perform the following experimental tasks:

- To write down the start and end time of the experiment.
- To fill out a pre-experiment questionnaire, which included academic and industry experience.

- To construct the statechart using the Yakindu tool (YAKINDU, 2014).
- The experimental group also had to fill out a post-experiment questionnaire.

5.3.3 Data validation

The subjects' data was validated. When subjects finished the experiment, we checked for each one of them, if their data forms were filled in a reasonable way. Furthermore, we also checked that everybody has understood how to fill in the data in a correct way.

5.4 Analysis & interpretation

The data collected during operation step provided input to the Analysis & interpretation activity. During the analysis phase, we understand the data while the interpretation phase determines whether the hypothesis was accepted or rejected (WOHLIN et al., 2012).

In the operation phase, we applied a pre-experiment questionnaire to each subjects of both groups in order to investigate their profile and previous experience. From the analysis of Figure 5.1, we can notice that both control group and experimental group were well balanced. They were students of different course types (undergraduate, and graduate - including master's and doctoral students) and they had different background, i.e. they attended/or were attending different undergraduate courses. The majority studied/is studying computer science.

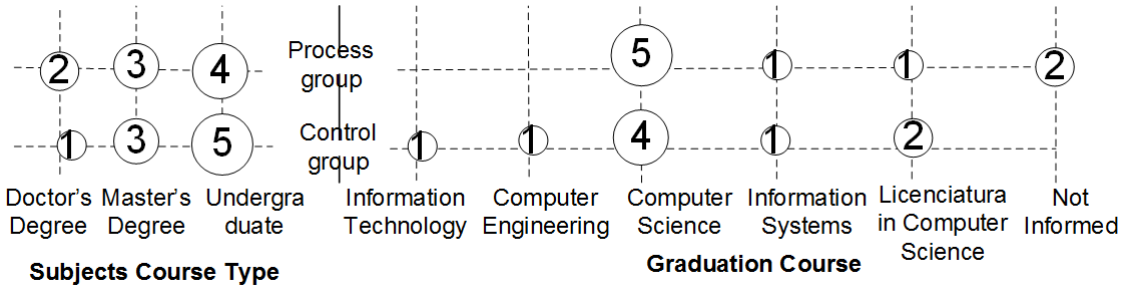


Figure 5.1: Subject's Profile (AUTHOR, 2015).

We also asked the participants about their experience with behavior modeling and the results are shown in Figure 5.2. The subjects had some experience with behavior modeling and both groups were more experienced with use case diagram and class diagram. These notations have a different level of abstraction compared to statecharts.

We requested the subjects to answer about their level of proficiency on behavioral modeling (statecharts and other modeling languages). The results are presented in Figure 5.3. The majority (of both groups) said that they had proficiency in modeling languages. It is important to note that this is a subjective affirmation that depends on how the subjects evaluate their knowledge.

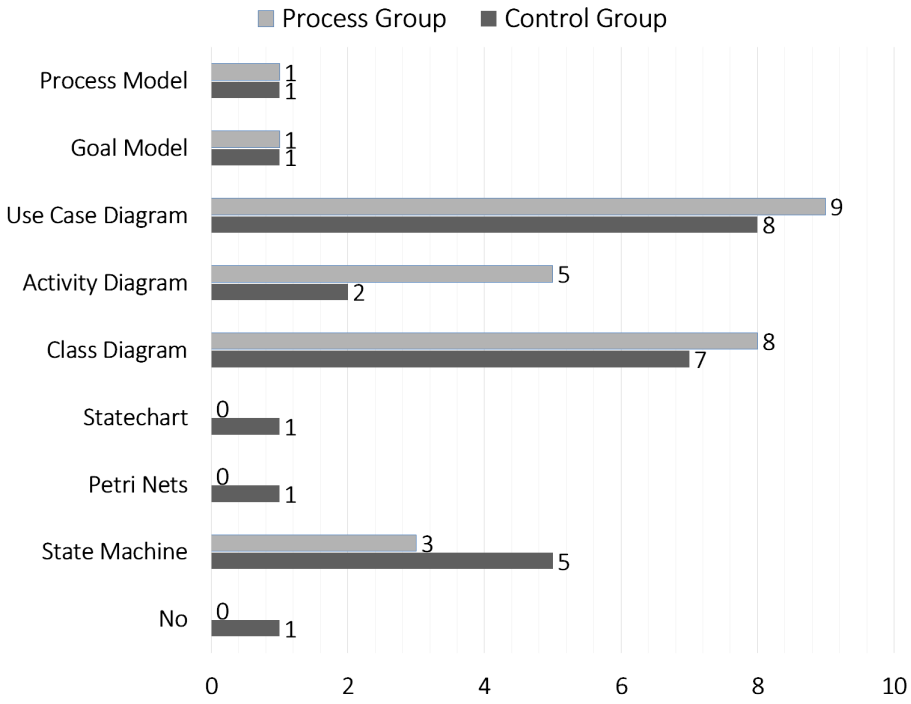


Figure 5.2: Experience in behavior modeling (AUTHOR, 2015).

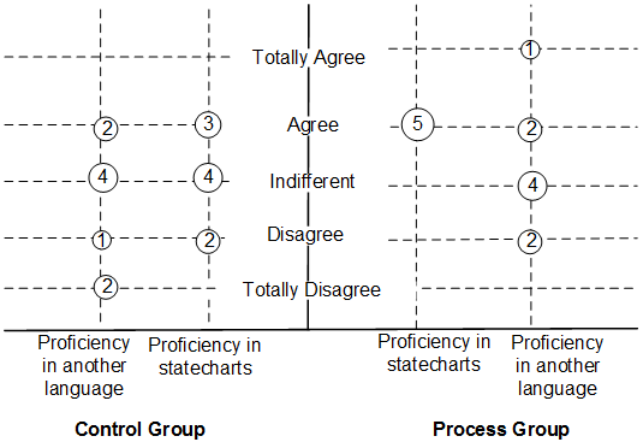


Figure 5.3: Proficiency in behavior modeling languages (AUTHOR, 2015).

After conducting the experiment, we analyzed each statechart produced by each subject of the the control group and experimental group in order to measure the dependent variables.

In relation to the syntactic correctness, the number of syntactic errors and the number of warnings of each subject of the control group is presented in Table 5.2. The results of the experimental group are presented in Table 5.3.

From the results presented in Table 5.2 and Table 5.3, we can conclude that the number of syntactic errors was 66.67% higher than the number of control group. Analysing these results, we noticed that the syntactic errors were made by two subjects (4 and 8 mistakes) in the control group and by one subject in the experimental group (4 mistakes).

This high number of syntactical errors in the control group was caused by two participants.

Table 5.2: Syntactic Correctness of statecharts of control group.

Subject	Number of Syntactic Errors	Number of Warnings
#6	4	0
#7	0	0
#8	0	2
#9	8	0
#10	0	0
#11	0	0
#15	0	3
#18	0	0
#20	0	0
Mean	**1.33**	**0.56**

Table 5.3: Syntactic Correctness of statecharts of experimental group.

Subject	Number of Syntactic Errors	Number of Warnings
#1	0	0
#3	0	0
#4	0	0
#12	0	7
#13	0	7
#14	0	0
#16	0	0
#17	0	0
#19	4	0
Mean	**0.44**	**1.56**

The other subjects did not make any syntactic error. The subject that made 4 syntactic errors in the control group did not consider himself/herself proficient in any system modeling language although he/she have already used a language for system modeling before the experiment. The subject that made 8 syntactic errors in the control group agree that he/she has proficiency in system modeling languages. The syntactical errors in the experimental group were made by one subject. This subject did not agree that he/she is proficient in state diagrams or other system modeling languages.

The number of warnings, on the other hand, was 64,29% higher in the experimental group as shown in Figure 5.4. In relation to the control group, these errors were made by two subjects (2 and 3 errors). The subject that made 2 errros agree that he/she has proficieny in state diagrams including statecharts and said that he/she is indifferent to others system modeling languages. The subject that made 3 errors did not considered himself/herself proficient in any system modeling languages.

In relation to the number of warnings of the experimental group, these errors were made by two subjects (7 errors each). One subject agreed that he/she is proficient in state diagrams and statecharts and he/she is indifferent to other system modeling languages. The other subject

also agreed in his/her proficiency in state diagrams and totally agreed that he/she is proficient in other system modeling languages.

Accordingly, we can infer that the level of proficiency in statecharts and other system modeling languages said by the subjects does not have a direct impact on the results of syntactic correctness.

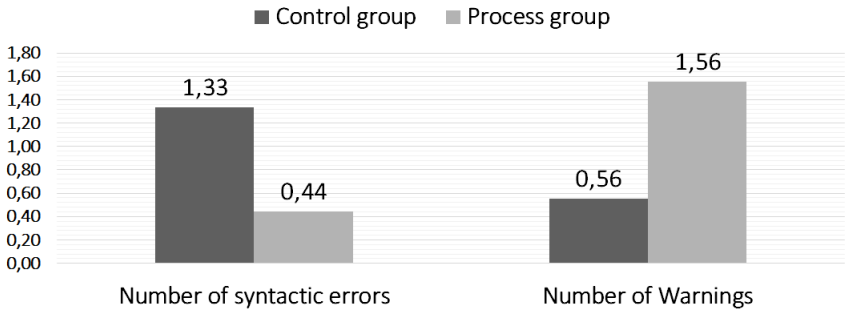


Figure 5.4: Syntactic correctness (AUTHOR, 2015).

Table 5.4 and Table 5.5 present the results of the metrics used to characterize the structural complexity of statecharts (superstates, orthogonal states, idle states, final states, simple states, state transitions, choice, variables and actions) of each subject in the experiment.

Table 5.4: Structural complexity of statecharts of control group.

	Super-states	**Ortho-gonal states**	**Idle states**	**Final**	**Simple states**	**State transi-tions**	**Choice**	**Variables**	**Actions**
#6	2	6	1	0	19	22	0	4	2
#7	1	4	0	0	21	22	0	1	0
#8	1	6	1	0	17	21	0	1	0
#9	5	4	0	1	13	15	0	21	26
#10	1	7	7	0	21	32	9	9	8
#11	3	6	0	0	17	26	0	6	5
#15	1	8	0	0	20	34	0	23	18
#18	4	0	4	0	19	28	0	2	0
#20	5	4	0	0	22	26	0	0	0
Mean	**2.56**	**5.00**	**1.44**	**0.11**	**18.78**	**25.11**	**1.00**	**7.44**	**6.56**

From the analysis of Table 5.4 and Table 5.5 we can notice that the experimental group used more superstates, orthogonal states, idle states. The control group, on the other hand, used more final states, simple states, state transitions, choice, variables and actions. When we grouped these values (see Figure 5.5), we found that the structural complexity of control group was 8.33% higher (mean of 68 elements) than the experimental group (62.33 elements). These results indicate that for the scenario in which the experiment was conducted that the structural complexity of the statecharts generated using the GO2S process was lower than the satecharts produced not using it.

Table 5.5: Structural complexity of statecharts of experimental group.

	Super-states	Ortho-gonal states	Idle states	Final	Simple states	State transi-tions	Choice	Variables	Actions
#1	6	5	4	0	15	19	0	0	0
#3	5	7	7	0	21	23	0	5	0
#4	3	12	5	0	18	19	0	9	0
#12	3	6	1	0	17	15	0	7	0
#13	3	14	2	0	24	23	0	6	0
#14	5	8	9	0	23	25	0	7	0
#16	6	9	7	0	20	26	0	8	0
#17	5	8	4	0	15	16	0	8	0
#19	4	4	0	0	15	20	0	5	0
Mean	**4.33**	**8.17**	**3.83**	**0**	**19**	**20.83**	**0**	**6.83**	**0**

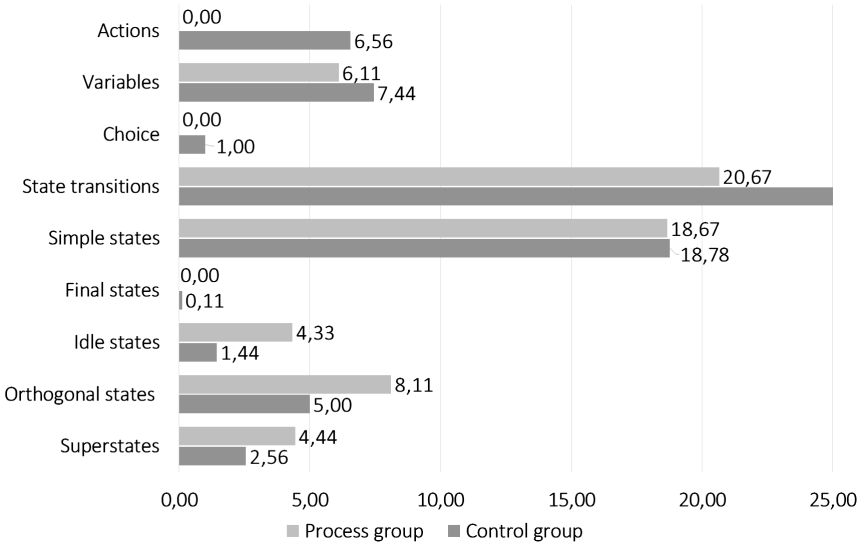


Figure 5.5: Structural complexity (AUTHOR, 2015).

The behavioral similarity was another dependent variable analyzed in the statecharts. Table 5.6 and Table 5.7 show the results of each subject of control and experimental groups. These values were calculated by analyzing each statechart of all subjects and verifying if the functionalities behave (in sequence, parallel, alternative and other possible flows) as described in the requirements specification. Finally, we calculated the porcentage of correct funcionalities from the total number of funcionalities.

Considering the behavioral similarity, we observed that the number of correct functionalities i.e. they behave as described in requirements specification is higher in the experimental group. Therefore, the mean of behavioral similarity of the experimental group was 21.49% higher than the control group as indicated in Figure 5.6.

Moreover, the time spent by each subject to construct the statechart is presented in Table 5.8 and Table 5.9.

Table 5.6: Behavioral Similarity of statecharts of control group.

Subject	Behavioral Similarity (%)
#6	66.67
#7	46.67
#8	60
#9	20
#10	73.33
#11	66.67
#15	73.33
#18	33.33
#20	46.67
Mean	**58.67**

Table 5.7: Behavioral Similarity of statecharts of experimental group.

Subject	Behavioral Similarity (%)
#1	53.33
#3	93.33
#4	80
#12	66.67
#13	100
#14	80
#16	80
#17	73.33
#19	53.33
Mean	**75.56**

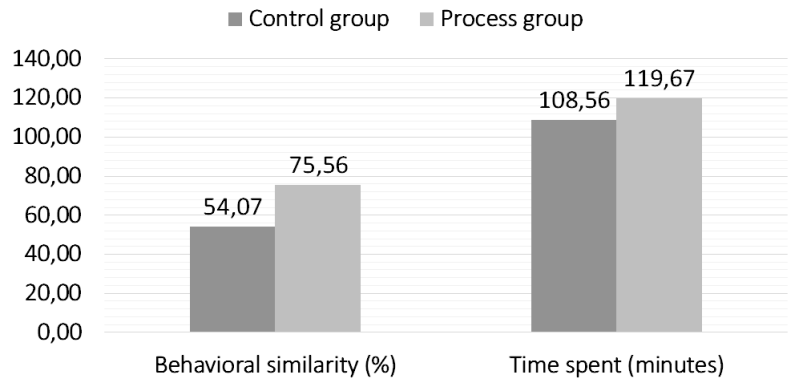


Figure 5.6: Behavioral similarity (AUTHOR, 2015).

From the results of these tables, we can conclude that the time to construct the statecharts was slightly higher (119.67 minutes) than the control group (108.56 minutes) with a small difference of 11.11 minutes (9.29%) as shown in Figure 5.6. These results indicate that, although there is an extra effort to perform all the activities of the GO2S process, the extra time required is not significant compared to not using it.

We also asked the subjects of the experimental group to fill out a post-questionnaire form

Table 5.8: Time spent to implement the statecharts of control group.

Subject	Time spent (minutes)
#6	70
#7	68
#8	65
#9	115
#10	113
#11	126
#15	109
#18	153
#20	158
Mean	**108.56**

Table 5.9: Time spent to implement the statecharts of experimental group.

Subject	Time spent (minutes)
#1	112
#3	91
#4	76
#12	110
#13	120
#14	124
#16	141
#17	147
#19	156
Mean	**119.67**

in order to obtain their cognitive complexity about the process. We provided fifteen statements (presented in Table 5.10) where they had to choose an answer using the following scale: Totally Disagree (TD), Disagree (D) Indifferent (I), Agree (A), Totally Disagree (TA). For the questions not answered by the subjects, we marked as Not Answered (NA). The results of the cognitive complexity of GO2S process are listed in Table 5.11. These results indicate that the GO2S process is easy to understand.

5.5 Presentation & package

The subjects of our experiment were students of different course types (graduate and undergraduate students) having the mostly studied/is studying computer science course. They stated that they have modeled the behavior of systems previously and both groups have more experience with use case diagram and class diagram. Besides, the majority of both groups agree that they had proficiency in modeling languages.

After analysing the statecharts produced by the subjects, we can conclude in relation to the syntactic correctness, the number of syntactic errors of control group was 66.67% higher

Table 5.10: Statements used to evaluate cognitive complexity.

Number	Statement
1	The process for statecharts derivation from goal models is understandable.
2	Step 1 is easy to understand.
3	The notation of goal model is easy to understand.
4	The use of goal models facilitates the creation of statecharts.
5	Step 2 is easy to understand.
6	The notation for context specification is easy to understand.
7	Step 3 is easy to understand.
8	Step 4 is easy to understand.
9	The use of flow expressions facilitates the creation of statecharts.
10	The use of flow expressions makes the creation of statecharts more systematic.
11	Step 5 is easy to understand.
12	Statecharts makes easy to understand the system's behavior.
13	The creation of statecharts contributes to a more complete system specification.
14	The mapping of tasks to states facilitates the creation of statecharts.
15	The mapping between goals and super-states improves the organization of the statechart.

Table 5.11: Results of cognitive complexity.

Number	TD (%)	D (%)	I (%)	A (%)	TA (%)	NA (%)
1				66.67	33.33	
2		11.11		88.89		
3			55.56	44.44		
4			11.11	44.44	44.44	
5		33.33	22.22	44.44		
6		22.22	22.22	55.56		
7	11.11	33.33	22.22	33.33		
8			11.11	55.56	33.33	
9			22.22	66.67	11.11	
10			22.22	66.67	11.11	
11			11.11	66.67	11.11	11.11
12			11.11	44.44	33.33	11.11
13			33.33	11.11	44.44	11.11
14			11.11	55.56	22.22	11.11
15				44.44	44.44	11.11

than the number of the experimental group. The number of warnings, on the other hand, was 64.29% higher in experimental group.

Moreover, the structural complexity of control group is higher (mean of 68 elements) than experimental group (62.33 elements) corresponding to a reduction of 8.33%. In addition, the mean of behavioral similarity of experimental group was higher (75.56%) than control group (54.07%) corresponding to a difference of 21.49%. Besides, the time spent by the experimental

group to construct the statecharts was higher (119.67 minutes) than the control group (108.56 minutes) with a difference of 11.11 minutes (9.29%).

The nine subjects that applied our process also filled out a post-questionnaire in which they expressed their opinion about the process using the following scale Totally Disagree, Disagree, Indifferent, Agree and Totally Agree. We asked their opinion about easiness to apply the process and all subjects agreed (66.67% agreed and 33.33% totally agreed) that the process is understandable.

The results of our experiment are discussed in more details in Chapter 6.

5.6 Threats to Validity

This section discusses how valid the results are and if we can generalize them to a broad population. According to WOHLIN et al. (2012), there are four kinds of validity: internal, conclusion, construct and external as discussed in the following subsections.

5.6.1 Internal Validity

Internal validity analyzes if the collected data in the study are result of the dependent variables and not from an uncontrolled factor. We tried to mitigate the selection bias (i.e. there are differences between the subjects' expertise) by performing a random assignment of the subjects to the control group and the experimental group.

Despite being separated in two groups, one that used the GO2S process and the other that did not use, both groups received the same goal model and system specification. Therefore, we did not expect the subjects to be unhappy or discouraged in performing or not the treatment, since the resulting statechart should be behaviorally equivalent. Finally, given that the experiment was performed in one day related to a domain that they had no contact before, we mitigate the history and maturation effects by making observation at a single time point.

5.6.2 Conclusion Validity

Conclusion validity is related to the ability to reach a correct conclusion about the collected data, as well as the reliability of measures and the collected data. We tried to improve the reliability of treatment implementation by using the same treatment and providing the same training, with the same instructor for all subjects of the experimental group. If the training was provided by different instructors, it would not possible to infer whether the results were influenced by the training or they were derived from the GO2S process. Thus, we mitigate this possible threat to validity. We also attempted to improve the conclusion validity by randomly choosing the subjects of both groups, thus promoting heterogeneous groups that were not correlated with the dependent variables.

5.6.3 Construct Validity

Construct validity is concerned to the relationship between the concepts and theories behind the experiment as well as what is measured and affected. In order to mitigate threats of this nature, we carefully designed our study. We chose objective measurements that did not depend on who was administering the test. Therefore, the subjects applied the treatment to a specification of a *Smart Home System* using an execution plan, which explained how to apply the treatment. In addition, they performed a dry run to make clear how the treatment should be implemented and how data should be collected.

5.6.4 External Validity

External validity is concerned with the ability to generalize the results to an industrial environment. One expected result of this study is to guide software engineers on when to use (or not) the GO2S process. As we used randomization to separate the subjects in two groups, we expect a decrease on the confounding factors (factors that can influence the results of the experiments), since the most important is the subjects' expertise.

Even though we had different types of students (undergraduate, master's and doctoral's degree), the limited number of subjects does not allow to generalize outside the scope of the study. On the other hand, we expect that the results, including the subjects' feedback, can be used as guidelines to improve our process.

Although the results are limited by the narrow scope, we believe that the process and the study design are considerable contributions. This experiment can guide other studies in order to evaluate the proposed process with more general and conclusive results and can also support other kind of studies.

5.7 Ethics

In our controlled experiment, we were committed to make our research ethical. Therefore, we addressed the ethical principles that form the core of several research ethics guidelines and codes: informed consent, beneficence, confidentiality (VINSON; SINGER, 2008).

The principle of informed consent stipulates that potential subjects should be informed of all relevant facts about a study before making an explicit, free and well-considered decision about whether to participate (VINSON; SINGER, 2008). Therefore, we provided to subjects all the information necessary to understand how the research would affect them: the purpose of the research, its procedure, the risks to the subjects, the anticipated benefits to the subjects, alternatives to participation, the treatment of confidential information, the voluntary nature of participation, and a statement offering to answer the subjects' questions.

The degree of beneficence results from a weighted combination of risks, harms, and benefits to the subjects and society from participation in a study (VINSON; SINGER, 2008).

In our experiment, the risks of harm were minimized by protecting the confidentiality. The principle of confidentiality refers to the subjects' right to expect that any information they share with researchers will remain confidential (VINSON; SINGER, 2008).

Accordingly, we executed procedures in order to maintain confidentiality and reduce the risks of harm. During the experiment, we randomly assigned a number to each participant and they answered all questionnaires anonymously. Therefore, the data anonymity was preserved since the analysis of the data could not reveal the identity of the subjects.

5.8 Final Considerations

In this section, we described the empirical method used to evaluate our process. We conducted a controlled experiment to investigate if the GO2S process could improve the derivation of statecharts of context-sensitive systems from requirements models. Since experimentation is a labor-intensive task, we followed the framework proposed by WOHLIN et al. (2012) for performing experiments in software engineering in order to make sure that the proper actions were taken to ensure a successful experiment. In the next chapter we discuss the results obtained in this dissertation, we summarize the main contributions, and we indicate some future works that are required to improve our approach.

6

Conclusions

This chapter discusses the results obtained in this dissertation. Moreover, it summarizes the main contributions, scope and limitations found. We indicate some future works that are required to improve our approach. Finally, we list papers related to this dissertation that were published.

6.1 Discussion

Software-development organizations frequently begin their activities with one of these alternative starting points - requirements or architectures - often adopting a waterfall development process. The start from the requirements produces artificially frozen requirements documents for use in the next step in the development life cycle. Starting from architecture, on the other hand, creates systems with constrained architectures that restrict users and handicap developers by resisting inevitable and desirable changes in requirements (NUSEIBEH, 2001).

Software engineers have difficulties to understand and define what to consider as context and how to design context-sensitive systems. A possible reason is the lack of consensus in the literature regarding the terminology, characteristics and specificities necessary to develop them. There is a need for approaches to guide the designers to perform activities related to the specification of the behavior of context-sensitive systems.

Processes provide steps that support an activity and are important as they can be used as checklists and guidelines of what to do and how to do it (WOHLIN et al., 2012). In order to develop context-sensitive systems, several steps are required to derive their behavior. Thus, a systematic process for derivation of the behavior of context-sensitive systems from requirements models is needed.

This dissertation proposes GO2S a systematic process for obtaining the behavior of context-sensitive systems (expressed as statecharts) from requirements models (expressed as goal models) following the twin peaks concept (NUSEIBEH, 2001). Altough the process was presented in a sequential way in Figure 3.1, this is an iterative process centered on the incremental refinement of a goal model, which provides different views of the system (design, contextual,

behavioral). Accordingly, it may be necessary to go back and forth in the activities until the desired level of detail is reached. The iterative characteristic of the GO2S process can be provided by a tool to support it. This tool will allow the software engineer to modify some view of the model, and to reflect these changes in the next views.

The benefits of obtaining the behavior of context-sensitive are manifold: the models can be used as a communication channel among stakeholders during system-development activities; they improve the confidence that the context-sensitive system will be able to achieve its goals; and they are amenable to reasoning (CLEMENTS et al., 2002). Hence, it becomes possible to analyze properties such as system's completeness, correctness, deadlocks as well as the satisfaction of some quality attribute.

In Chapter 3, we described the activities of GO2S process indicating input/output artifacts and a systematic way to execute each activity. The process can be divided into the following main activities: the first activity concerns the construction of design goal model. It is followed by the specification of contextual variation points. In the third activity, the tasks required for the monitoring and adaptation activities are specified. Later, the system behavior is represented in flow expressions. In the next one, a statechart is derived from the behavioral contextual design goal model. Finally, the last activity is the prioritization of variants is performed.

We explored the idea that it is possible to organize the development of CSS by structuring the GO2S process in six activities considering the main steps in order to develop a context-sensitive system: requirements specification, context specification, the adaptation and monitoring, the definition of system's behavior, the statechart derivation and the prioritization of variants. We hope that this organization can aid the maintenance and evolution of CSS, diminishing the complexity on building these applications.

The behavioral view of the context-sensitive architecture was represented in statecharts since it is a popular visual formalism to represent the behavior and this diagram, adopted by the UML, has a powerful graphical notation to describe reactive systems.

The input of GO2S process is a goal model. We adopted this model considering its benefits described in the literature (LAPOUCHNIAN, 2005) (ALI; DALPIAZ; GIORGINI, 2010): goals provide rationale for requirements that operationalize them; goals provide a precise criterion for sufficient completeness of a requirements specification; a single goal model can capture variability in the problem domain through the use of alternative goal refinements; they provide a natural mechanism for structuring complex requirements documents, and goals offer a very intuitive way to elicit and analyze requirements.

This dissertation also proposed the GO2S metamodel that describes the concepts of the behavioral contextual design goal model, their properties and the valid connections between the elements. This metamodel relates the requirements, architectural design, context and behavior annotations in a unified metamodel.

Besides, many authors such as PENSERINI et al. (2007), MORANDINI et al. (2009), PIMENTEL et al. (2014), and ALI; DALPIAZ; GIORGINI (2010) agree that goal models have

been used as an effective means to capture the interactions and information-related requirements of adaptive systems and context-sensitive systems. The reason is that they incorporate the space of alternatives of a set of operations, i.e. variants, which gives more flexibility to meet stakeholders' goals in a dynamic environment (ALI; DALPIAZ; GIORGINI, 2010).

Flow expressions were used to specify the system behavior and thus, help the derivation of statecharts. These expressions are useful because they can aid in the design, analysis, and understanding of software systems (SHAW, 1978). Since every goal and task should be present in the flow expression, excepting the ones assigned to users, we can check the completeness of the derivation. Hence, every element should correspond to a state in the statechart.

In comparison with the work of PIMENTEL et al. (2014), the main difference is that we address the system's context, the operationalization of the NFRs, the specification of monitoring and adaptation in the same model and prioritization of variants. This prioritization was performed using the AHP method. This method uses a scale [1...9] that is based on psychological theories and experiments that points to the use of nine unit scales as a reasonable set that allows humans to perform discrimination between preferences for two items. Each value of the scale can be given a different interpretation allowing a numerical, verbal or graphical interpretation of the values (SAATY, 1987).

The prioritization of variants 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 most critical NFR from the point of view of 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 support for this method.

The AHP method has many benefits (BRITO et al., 2007): it is a well-known and accepted method; it is appropriate for handling conflicting concerns problems; it has the ability to quantify subjective judgements; it is capable of comparing alternatives in relation to established criteria; and it provides means to guarantee the logical consistency of the judgements. Hence, the AHP has proven to be an effective method for prioritizing objectives. In industrial projects, this method has been reported as being effective, accurate and also to yield informative and trustworthy results (KARLSSON, 1996). However, since all unique pairs must be compared, the required effort can be substantial.

An experiment to study the scalability of statechart generation algorithm was previously conducted by PIMENTEL et al. (2014). 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.

The design of context-sensitive systems entails more work in comparison to applications that do not consider context since they must care for context-related tasks, such as the acquisition, processing, storage and presentation of contextual information. Hence, it is important to note that the monitoring required to assess the context may have a significant impact on the system under development. The context monitoring 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.

Chapter 4 presents an illustration of the application of the GO2S process in the Znn exemplar. Znn is a news service, based on real sites, that serves multimedia news content to its customers through a website. This problem is available in the repository of examples and challenge problems that the software engineering for self-adaptive systems community can use to motivate research, exhibit solutions and techniques, and compare results.

In Chapter 5, we described a controlled experiment conducted to evaluate our process. The objective of this experiment was to compare the output of the GO2S process (a statechart) with the statecharts produced by the control group through five dependent variables: time to implement, syntactic correctness, structural complexity, behavioral similarity and cognitive complexity.

The subjects were students of different course types (graduate and undergraduate students) having the mostly studied/is studying computer science course. The subjects in our experiment stated that they had previously modeled the behavior of systems (see Figure 5.2). Moreover, they had more experience with use case diagram and class diagram. Besides, the majority of the subjects agreed that they have proficiency in modeling languages.

The majority of the subjects of experimental group said that they are proficient in statecharts or other system modeling languages. This was not a threat to validity of the experiment since this is a subjective assertion and only two subjects in the control group made syntactic mistakes in the statecharts. The subject that made 8 errors agreed that he/she is proficient and the subject that made 4 errors did not agree. Accordingly, we can infer that the level of proficiency said by the subjects in statecharts and other system modeling languages does not have a direct impact on the results of syntactic correctness.

From the statecharts produced by the subjects, we can conclude in relation to the syntactic correctness, the number of syntactic errors of control group was 66.67% higher than the number of process group (see Table 5.2 and Table 5.3). These errors were made by two subjects in the control group and one subject in the experimental group. The other participants did not make syntactic error. If we consider only the subjects that modeled the functionalities as described in the requirements document (the subjects that modeled correctly at least 50% of the functionalities - Table 5.6 and Table 5.7), the mean of syntactic errors in the control group still remains higher than experimental group.

The number of warnings was 64.29% higher in process group (see Table 5.2 and Table 5.3). These warnings were present in the statecharts of two subjects in the experimental group

and two of the control group. If we perform the same analysis with the subjects that modeled correctly at least 50% of the functionalities) described in Table 5.6 and Table 5.7, the average of warnings in the experimental group still remains higher than experimental group. However, one of these two subjects of the experimental group, whose statecharts had warnings, modeled all functionalities correctly (100%) and the other one modeled 66.67% correctly. Besides, these two participants did not make any syntactical errors.

Moreover, the structural complexity of control group (see Table 5.4) is higher than process group (see Table 5.5). The average of elements in the control group was 68 elements and the mean of the experimental group was 62.33 elements corresponding to a reduction of 8.33%. If we consider only the subjects that modeled correctly at least 50% of the functionalities, the difference between the control and experimental groups increases to 11.16%.

In addition, the mean of behavioral similarity of process group was higher (75.56%) than control group (54.07%) corresponding to a difference of 21.49%. If we consider only the subjects that modeled correctly at least 50% of the functionalities, the difference between the control and experimental groups decreases to 10.56%.

Besides, the time spent by the process group to construct the statecharts was higher (119.67 minutes) than the control group (108.56 minutes) with a difference of 11.11 minutes (9.29%). If we consider only the subjects that modeled correctly at least 50% of the functionalities), the difference between the control and experimental groups increases to 12.83 minutes.

Thereby, the experiment results allow us to reject the null hypotheses (H01..4) that there is no difference between using or not the GO2S process and accept the alternative hypotheses (H12..4). We can not reject the H11 since the time to implement the statechart was higher in the experimental group.

Therefore, there are some evidence that the number of functionalities that behaved as specified in requirements document was higher in process group models. Besides, the structural complexity was lower in the process group models. However, the time spent to implement, on the other hand, was slightly higher (9.29%) in the process group.

The nine subjects that applied our process also filled out a post-questionnaire in which they expressed their opinion about the process using the following scale Totally Disagree, Disagree, Indifferent, Agree and Totally Agree. We asked their opinion about easiness to apply the process and six subjects (66.67%) agreed that the process was easy to use, while 3 subjects (33.33%) said they Totally Agree. Hence, we can reject the H05 hypothesis that the GO2S process is not easy to understand and accept the H15 hypothesis.

Eight subjects (88.89%) agreed that the step 1 (Construction of design goal model) is easy and only one subject disagreed (11.11%) as illustrated in Table 5.11. Considering the easiness of understanding Step 2 (Specification of contextual variation points), three subjects (33.33%) disagreed, two subjects (22.22%) marked indifferent, and four agreed (44.44%) as shown in Table 5.11. These results indicate that some subjects had some difficulties to identify

and specify the context.

In relation to the easiness of Step 3 (Specification of monitoring and adaptation), three subjects agreed (33.33%), two subject marked indifferent (22.22%), three disagreed (33.33%) and one totally disagreed (11.11%) as demonstrated in Table 5.11. We expected these results since Step 3 is the most critical activity of the GO2S process that has many steps to specify all the tasks required to monitor the context and adapt the system according to it.

The Step 4 (Specification of flow expressions) was also evaluated about its easiness of understanding (see Table 5.11). The majority agreed (55.56%) and totally agreed (33.33%) that this step was easy of understand. Only one subject marked as indifferent (11.11%). We also asked the subjects if the Step 5 (Statechart derivation and refinement) was easy to understand. Six subjects (66.67%) agreed that the step 5 was easy, one totally agreed (11.11%), one marked indifferent (11.11%) and one subject did not answered this question (11.11%) as illustrated in Table 5.11.

Therefore, since the subjects did not have much difficulties to use the GO2S process, the results of the experiment indicate that it is possible to reproduce the process and it is understandable. Although the results are limited by the narrow scope we have, we believe that the process and the study design are considerable contributions. This controlled experiment can guide other studies in order to evaluate the GO2S process with more general and conclusive results and can also support other kind of studies.

Nevertheless, despite the encouraging results obtained, they must be considered as preliminaries. Further replication is necessary and also new experiments must be carried out with software engineers who develop context-sensitive systems.

6.2 Limitations of our experiment

The context of our experiment is students of undergraduate, master's and doctor's degree from a Requirements Engineering course at an university. The subjects were eighteen students enrolled in the course. The experiment was run off-line (not in an industrial software development environment).

Many authors, such as BASILI; SELBY; HUTCHENS (1986), FENTON (1993) and SJOBERG et al. (2002) point out the difficulties of conducting controlled software engineering experiments in realistic environments. Accordingly, these environments can also be a weakness, because there are an enormous number of factors that differ across environments, in terms of desired cost/quality goals, methodology, experience, problem domain, constraints, etc (BASILI; SELBY; HUTCHENS, 1986). Hence, it may be too costly or impossible to manipulate an independent variable or to randomize treatments in real life (SJOBERG et al., 2002).

The particular individuals examined in an empirical study can make an enormous difference (SJOBERG et al., 2002). Someone could argue that we should use professionals in experiments because there may be many differences between students and professionals. These

differences are related to their experience and skill levels, use of professional methods and tools, and team work versus individual work (SJOBERG et al., 2002). However, the lack of professionals in software engineering experiments is due to the conception of high costs and large organisational effort.

SJOBERG et al. (2002) apud Warren Harrison (2000) state that professional programmers are hard to come by and are very expensive. Thus, any study that uses more than a few professional programmers must be very well funded. Even if we can somehow gather a sufficiently large group of professionals, the logistics of organizing the group into a set of experimental subjects can be daunting due to schedule and location issues. According to SJOBERG et al. (2002), empirical software engineering research departments should have particular budgets for paying students and software professionals for taking part in experiments. Unfortunately, this was not our case.

A non-controversial use of student experiments is to use them to test experimental design and initial hypotheses, before conducting experiments with professionals, as recommended by SJOBERG et al. (2002). We accept these arguments and assume that we tried to mitigate these effects by performing a careful experiment in which we tried to use an appropriate assessment criterion.

The experiment we conducted was specific, since it focused on the following metrics to evaluate the generated statecharts: time to implement, syntactic correctness, structural complexity, behavioral similarity, and cognitive complexity. These metrics were already used in the literature to evaluate behavioral models (DIJKMAN et al., 2011) (MIRANDA; GENERO; PIATTINI, 2005).

Moreover, formal designs and the resulting statistical robustness are desirable, but we should not be driven exclusively by the achievement of statistical significance. Common sense must be maintained, which allows us, for example, to experiment just to help develop and refine hypotheses (BASILI; SELBY; HUTCHENS, 1986). Besides, the number of subjects in our experiment is too small to conduct hypothesis testing.

We agree that the experiment results can be different according to the subjects and environmental conditions where the experiment is conducted. However, increasing the realism of software engineering experiments also requires an increase in the resources needed to conduct such experiments. Using professionals as subjects usually means that they must be paid. Development of necessary supporting tools is costly. Attracting experts to take part in the design, management and data analysis of realistic experiments also requires resources (SJOBERG et al., 2002).

In the next sections, some conclusions, considerations and future works are presented.

6.3 Contribuitions

The main contributions of this work consist in the proposal of:

- **A systematic process for deriving the behavior of context-sensitive systems, expressed as statechart, from requirements models, specified as goal models**. The GO2S process consists of six activities to guide the software engineer: *Construction of design goal model; Specification of contextual variation points; Specification of monitoring and adaptation; Specification of flow expressions, Statechart derivation and refinement*. The process was modeled using the BPMN language to capture the sequence of activities to be performed indicating the input/output artifacts. The process is useful both for guiding a context-sensitive systems development team on designing a new application and also as a conceptual foundation to support academic teaching activities on context and context-sensitive systems. This GO2S process for deriving the behavior of context-sensitive systems from requirements models is original.

- **Specification of monitoring and adaptation tasks in a contextual design goal model.** We propose the specification of monitoring and adaptation tasks in a single model, the contextual goal model. This is an important contribution since context-sensitive systems should provide three characteristics: monitoring, awareness and adaptation. Therefore, it does not require any additional notation or extension, the specification is performed using the elements already defined in this model.

- **The behavioral contextual design goal model.** This model allows to express, in a single model, information about requirements, architectural design, operationalization of NFRs, context, behavior, adaptation and monitoring tasks.

- **The GO2S metamodel.** This dissertation proposed the GO2S metamodel that describes the concepts of the behavioral contextual design goal model, their properties and the valid connections between the elements. This metamodel relates the requirements, architectural design, context and behavior annotations in a unified metamodel. To the best of our knowledge, there is no metamodel that addresses all those elements.

- **A running example to demonstrate the process application.** In order to illustrate our process we considered the popular Znn.com system, aiming closely follow the defined process.

- **A controlled experiment in order to evaluate our process.** The experiment results allowed us to reject hypotheses that there is no difference between using or not the proposed process. The results of the experiment indicate that there are some evidence that the number of functionalities that behaved as specified in requirements document was higher in the group that used the GO2S process. Besides, the structural complexity was lower. However, the time spent to implement, on the other hand, was slightly higher in the group that followed our approach.

6.4 Future Works

For future works, the following activities can be undertaken:

- **Develop a case tool to implement the process.** This tool could be used to produce the goal model and guide the software engineer to apply the GO2S process generating the different views (design, contextual and behavioral) of our process to implement the statechart derivation. Such tool can be developed using the metamodel proposed in this dissertation and it could use OCL rules to constrain the relationships between the metaclasses present in this unified metamodel.

- **Apply the process in complex systems**. The use of the process for more complex systems, especially in the context of the industry, would help to assess if it is suitable in different domains. Equally important is to conduct further investigation to assess its costs and benefits. This analysis would help to identify points of the process that require improvements.

- **Perform new controlled experiments.** Despite the encouraging results obtained, we consider them as preliminaries. Further replication is necessary and also new experiments must be carried out with software engineers who develop context-sensitive systems.

- **Develop mechanisms to perform the reasoning of context-sensitive systems from the generated statecharts.** Statecharts allow the reasoning through the analysis of properties such as system's completeness and correctness. This reasoning could be added to the GO2S process to improve the quality of generated statecharts.

- **Incorporate other architectural views in our process**. Software architecture can be composed of four views: structural, behavioral, deployment, and configuration. The structural view was already addressed in the work of PIMENTEL et al. (2012) and our process addressed the behavioral view. It is important to derive systematically the other architectural views, for example deployment and configuration in order to obtain a complete system specification.

6.5 Summary of publications

In this section we list papers related to this dissertation that were published in international venues.

VILELA, J.; CASTRO, J.; PIMENTEL, J.; SOARES, M.; LIMA, P.; LUCENA, M. Deriving the behavior of context-sensitive systems from contextual goal models. 2015. 30th ACM/SIGAPP Symposium On Applied Computing (SAC). April 2015. In press.

VILELA, J.; CASTRO, J.; PIMENTEL, J.; LIMA, P. On the behavior of context-sensitive systems. 2015. 18 Workshop em Engenharia de Requisitos (WER 2015). April 2015. In press.

DERMEVAL, D.; VILELA, J.; BITTENCOURT, I.; CASTRO, J.; ISOTANI, S.; BRITO, P.; SILVA, A. Applications of ontologies in requirements engineering: a systematic review of the literature. In: Requirements Engineering journal, 2015, pp.1-33.

DERMEVAL, D.; VILELA, J.; BITTENCOURT, I.; CASTRO, J.; ISOTANI, S.; BRITO, P. A Systematic Review on the Use of Ontologies in Requirements Engineering. In: Simpósio Brasileiro de Engenharia de Software (SBES), 2014, pp. 1-10.

REFERENCES

ABOWD, G. D. et al. Towards a better understanding of context and context-awareness. In: HANDHELD AND UBIQUITOUS COMPUTING. **Anais...** [S.l.: s.n.], 1999. p.304–307. (Lecture Notes in Computer Science, v.1707).

ALI, R. **Modeling and Reasoning about Contextual Requirements**: goal-based framework. 2010. Tese (Doutorado em Ciência da Computação) — Universita degli Studi di Trento.

ALI, R.; DALPIAZ, F.; GIORGINI, P. A goal-based framework for contextual requirements modeling and analysis. **Requirements Engineering**, v.15, n.4, p.439–458, 2010.

ALI, R.; DALPIAZ, F.; GIORGINI, P. Reasoning with Contextual Requirements: detecting inconsistency and conflicts. **Information and Software Technology**, v.55, n.1, p.35–57, 2013.

ANGELOPOULOS, K.; SOUZA, V. E. S.; MYLOPOULOS, J. Dealing with multiple failures in zanshin: a control-theoretic approach. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 9., New York, NY, USA. **Proceedings...** ACM, 2014. p.165–174. (SEAMS 2014).

ANGELOPOULOS, K.; SOUZA, V. E. S.; PIMENTEL, J. a. Requirements and Architectural Approaches to Adaptive Software Systems: a comparative study. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 8. **Proceedings...** IEEE Press, 2013. p.23–32. (SEAMS '13).

BASILI, V. R.; SELBY, R. W.; HUTCHENS, D. Experimentation in software engineering. **IEEE Transactions on Software Engineering**, v.SE-12, n.7, p.733–743, 1986.

BAZIRE, M.; BRÉZILLON, P. Understanding context before using it. In: DEY, A. et al. (Ed.). **Modeling and using context**. [S.l.]: Springer, 2005. p.29–40. (Lecture Notes in Computer Science, v.3554).

BRITO, I. S. et al. Handling Conflicts in Aspectual Requirements Compositions. In: RASHID, A.; AKSIT, M. (Ed.). **Transactions on Aspect-Oriented Software Development III**. [S.l.]: Springer Berlin Heidelberg, 2007. p.144–166. (Lecture Notes in Computer Science, v.4620).

CASTRO, J.; KOLP, M.; MYLOPOULOS, J. Towards requirements-driven information systems engineering: the tropos project. **Information systems**, v.27, n.6, p.365–389, 2002.

CHALMERS, D. **Contextual mediation to support ubiquitous computing**. 2002. Tese (Doutorado em Ciência da Computação) — University of London, Imperial College of Science, Technology and Medicine.

CHENG, S.-W.; GARLAN, D.; SCHMERL, B. Evaluating the effectiveness of the Rainbow self-adaptive system. In: ICSE WORKSHOP ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS (SEAMS '09). **Anais...** [S.l.: s.n.], 2009. p.132–141.

CHENG, S.-W.; SCHMERL, B. **Model Problem**: znn.com. 2014.

CHUNG, L. et al. Non-functional requirements. **Software Engineering**, 2000.

CLEMENTS, P. et al. **Documenting software architectures**: views and beyond. United States: Pearson Education, 2002.

DIJKMAN, R. et al. Similarity of Business Process Models: metrics and evaluation. **Information Systems**, v.36, n.2, p.498–516, 2011.

FENTON, N. How effective are software engineering methods? **Journal of Systems and Software**, v.22, n.2, p.141–146, 1993.

FIDALGO, R. N. et al. EERMM: a metamodel for the enhanced entity-relationship model. In: ATZENI, P.; CHEUNG, D.; RAM, S. (Ed.). **Conceptual Modeling**. [S.l.]: Springer Berlin Heidelberg, 2012. p.515–524. (Lecture Notes in Computer Science, v.7532).

GLASS, R. L. The software-research crisis. **IEEE Software**, v.11, n.6, p.42–47, 1994.

HAREL, D. Statecharts: a visual formalism for complex systems. **Science of computer programming**, v.8, n.3, p.231–274, 1987.

KARLSSON, J. Software requirements prioritizing. In: SECOND INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING. **Proceedings...** [S.l.: s.n.], 1996. p.110–116.

KLEIN, C. et al. A Survey of Context Adaptation in Autonomic Computing. In: AUTONOMIC AND AUTONOMOUS SYSTEMS, 2008. ICAS 2008. FOURTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.106–111.

LAPOUCHNIAN, A. Goal-oriented requirements engineering: an overview of the current research. **University of Toronto**, 2005.

LIU, Y.; MA, Z.; SHAO, W. Integrating Non-functional Requirement Modeling into Model Driven Development Method. In: ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE (APSEC), 17. **Anais...** [S.l.: s.n.], 2010. p.98–107.

LUCKEY, M. et al. Adapt Cases: extending use cases for adaptive systems. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 6., New York, NY, USA. **Proceedings...** ACM, 2011. p.30–39. (SEAMS '11).

MELLOR, S. J.; CLARK, T.; FUTAGAMI, T. Model-driven development: guest editors' introduction. **IEEE software**, v.20, n.5, p.14–18, 2003.

MIRANDA, D.; GENERO, M.; PIATTINI, M. Empirical Validation of Metrics for UML Statechart Diagrams. In: ENTERPRISE INFORMATION SYSTEMS V. **Anais...** [S.l.: s.n.], 2005. p.101–108.

MORANDINI, M. et al. A Goal-Oriented Approach for Modelling Self-organising MAS. In: ALDEWERELD, H.; DIGNUM, V.; PICARD, G. (Ed.). **Engineering Societies in the Agents World X**. [S.l.]: Springer Berlin Heidelberg, 2009. p.33–48. (Lecture Notes in Computer Science, v.5881).

MURATA, T. Petri nets: properties, analysis and applications. **Proceedings of the IEEE**, v.77, n.4, p.541–580, apr 1989.

NICOLA, R. d. Extensional equivalences for transition systems. **Acta Informatica**, v.24, n.2, p.211–237, 1987.

NUSEIBEH, B. Weaving together requirements and architectures. **Computer**, v.34, n.3, p.115–119, 2001.

OMG. **Object Management Group. Business Process Model and Notation**. 2014.

PENSERINI, L. et al. High Variability Design for Software Agents: extending tropos. **ACM Transactions on Autonomous and Adaptive Systems (TAAS)**, New York, NY, USA, v.2, n.4, Nov. 2007.

PIMENTEL, J. et al. Deriving software architectural models from requirements models for adaptive systems: the stream-a approach. **Requirements Engineering**, v.17, n.4, p.259–281, 2012.

PIMENTEL, J. et al. From requirements to statecharts via design refinement. In: ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING: 24-28 MARCH 2014; GYEONGJU, KOREA, 29. **Proceedings...** [S.l.: s.n.], 2014. p.995–1000.

RANJITA, K. S.; PRAFULLA, K. B.; DURGA, P. M. Minimal TestCase Generation for Object-Oriented Software with State Charts. **CoRR**, 2012.

SAATY, R. W. The analytic hierarchy process - what it is and how it is used. **Mathematical Modelling**, v.9, n.3, p.161–176, 1987.

SANTOS, E. B. **Business Process Configuration with NFRs and Context-Awareness**. 2013. Tese (Doutorado em Ciência da Computação) — Federal University of Pernambuco, Centers of Informatics.

SANTOS, V. V. dos. **CEManTIKA**: a domain-independent framework for designing context-sensitive systems. 2008. Tese (Doutorado em Ciência da Computação) — Federal University of Pernambuco, Centers of Informatics.

SHAW, A. Software Descriptions with Flow Expressions. **IEEE Transactions on Software Engineering**, v.SE-4, n.3, p.242–254, 1978.

SJOBERG, D. I. et al. Conducting realistic experiments in software engineering. In: INTERNATIONAL SYMPOSIUM IN EMPIRICAL SOFTWARE ENGINEERING, 2002. **Proceedings...** [S.l.: s.n.], 2002. p.17–26.

VAN LAMSWEERDE, A.; DARIMONT, R.; MASSONET, P. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In: SECOND IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING. **Proceedings...** [S.l.: s.n.], 1995. p.194–203.

VIEIRA, V. et al. Uso e Representação de Contexto em Sistemas Computacionais. **Cesar AC Teixeira, Clever Ricardo G. de Farias, Jair C. Leite, and Raquel O. Prates.(Org.)., Tópicos em Sistemas Interativos e Colaborativos**, p.127–166, 2006.

VIEIRA, V.; TEDESCO, P.; SALGADO, A. C. Designing context-sensitive systems: an integrated approach. **Expert Systems with Applications**, v.38, n.2, p.1119–1138, 2011.

VINSON, N. G.; SINGER, J. A Practical Guide to Ethical Research Involving Humans. In: GUIDE TO ADVANCED EMPIRICAL SOFTWARE ENGINEERING. **Anais...** [S.l.: s.n.], 2008. p.229–256.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer, 2012.

YAKINDU. **Yakindu Modeling Tool**. 2014.

YU, Y. et al. From Goals to High-Variability Software Design. In: AN, A. et al. (Ed.). **Foundations of Intelligent Systems**. [S.l.]: Springer Berlin Heidelberg, 2008. p.1–16. (Lecture Notes in Computer Science, v.4994).

APPENDIX

A

Pre-Experiment Questionnaire

We applied a pre-experiment questionnaire in portuguese in order to know the subjects profile and experience in system modeling. The file that subjects received is presented in next page.

Derivation of statecharts from goal models to context-sensitive systems
Experiment
Date: 03/12/2014
Group: ( ) **Using the process** ( ) **Control**
Anonymous questionnaire

Pre-Experiment – Questionnaire

The aim of this questionnaire is to obtain information about your background on systems modeling and software engineering. Your answers do not affect the other activities of this experiment, they simple provide us a context for the interpretation of results. Feel free to write beyond the designated lines in order to explain or detail your answers, if needed be.

1) On which academic level are you enrolled?

   ( ) Undergraduation ( ) Master ( ) Doctoral

2) What is/was your undergraduation course?

   _____

3) Do you have professional experience on software engineering?
   ( ) No ( ) Yes, for how long? _____

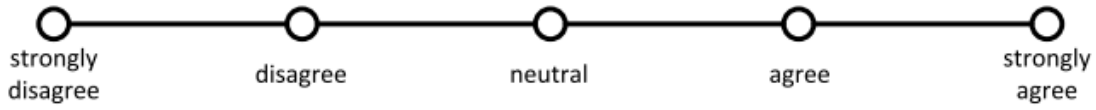
   If yes, which activities have you performed professionally?

   _____
   _____
   _____

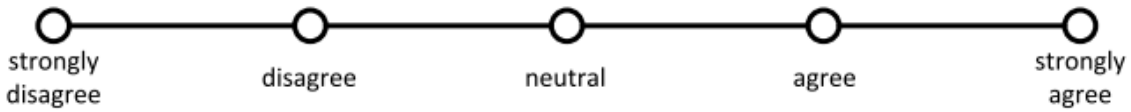
4) Have you ever used a modeling language for describing systems behavior before this training??
   ( ) No ( ) Statemachines ( ) Petri Nets ( ) Statechart
   ( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram
   ( ) Goal Model ( ) Process Diagram
   ( ) Other: which one(s)?

   _____

5) Choose one of the alternatives with respect to the following statement:
   "I am proficient in statecharts or other kinds of state diagrams".



6) Choose one of the alternatives with respect to the following statement:
   "I am proficient in a systems modeling language **different** of statecharts or other kinds of state diagrams".



B

Activity of Control Group

The control group received the specification in portuguese of simplified version of a *Smart Home System* described in natural language (portuguese), its goal model, and the activities the subjects had to perform in the experiment.

Statecharts Derivation
Experiment
Group: **Control** Data: 03/12/2014

Start Time: _____
End Time: _____

1) Consider the system for a smart home whose requirements and goal model are presented below.

The smart home system has been designed to make life easier for people with dementia problems and provide continuous care to them to ensure their safety and comfort. In order to achieve that, the smart home has to act in response to the context. In this experiment, we will use a small version of the system adapted from description available at [ALI, 2010][1].

The main goal of the system is to control the home for the patient: people with dementia suffer from serious problems with memory. As a result of these problems, a patient may forget to maintain healthy family environment. One of the features of the system is the temperature setting, in order to achieve this, the patient should be able to open/close the windows or turn on/off the ventilator as often as desired.

Protect the house against potential thieves is another goal that the smart home should be responsible. The house should provide the illusion that there is always someone at home when the patient is out. Thus, the system should turn on/off the lights when the patient is away from home for a long time and it is night; when sunrise should turn off the lights. This would help prevent a thief enter the house.

Furthermore, the smart home should at the same time act against any potential thief when he is inside the house area. Thus, the smart home should always monitor the rooms of the house and check if a person is acting suspiciously (by entering foran unusual entry, for example). If this occurs, the smart home should many times as necessary to close all doors and windows and then call the police.

While the system manages the temperature and protects the house from thieves, the system must provide patient entertainment in various ways, such as watching movies (for this he should log into a site of film and then the film is shown on television) or view a TV channel or send messages to friends/relatives of the patient to visit him when they do not do more than a month.

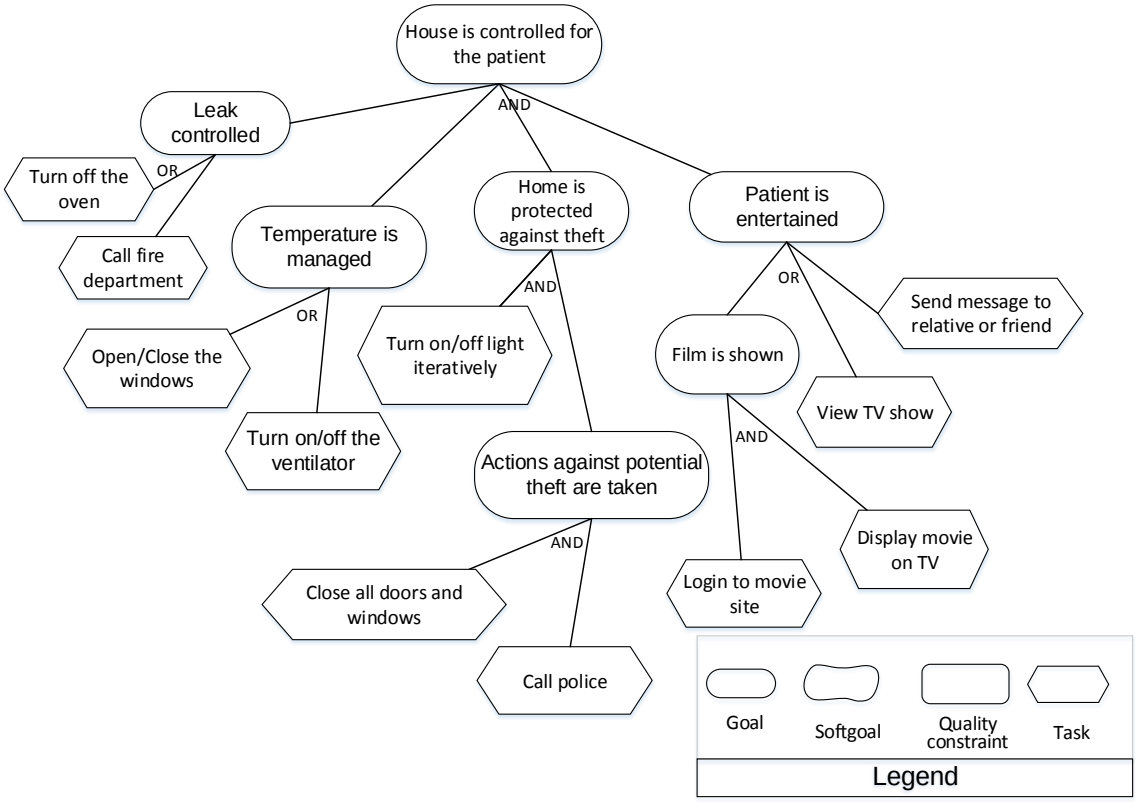
A critical requirement for the smart home is the gas leakage control, so the use of the oven should be controlled. Thus, the possible actions for the system are turn off the oven (when the patient finished using it) or notify the fire department (when it detects a leak).

For all these actions are taken by the system, it should perform the necessary monitoring.

2) From the specification presented above, create a statechart in Yakindu tool that represents the desired system behavior. This statechart should not have any added functionality or less than is described in the specification.

[1] Raian Ali. Modeling and Reasoning about Contextual Requirements: Goal-based Framework. Doctoral thesis, 2010.

The goal model of the smart home is presented below.



C

Activity of Experimental Group

The experimental group also received the specification in portuguese of simplified version of a *Smart Home System*. Besides the specification and its goal model, the activity also described the activities the subjects had to perform in the experiment.

Derivation of statecharts from goal models to context-sensitive systems
Experiment
Group: **Using the Process** Date: 03/12/2014

Start Time: _____
End Time: _____

1. Consider the system for a smart home whose requirements and goal model are presented below.

The smart home system has been designed to make life easier for people with dementia problems and provide continuous care to them to ensure their safety and comfort. In order to achieve that, the smart home has to act in response to the context. In this experiment, we will use a small version of the system adapted from description available at [ALI, 2010][1].

The main goal of the system is to control the home for the patient: people with dementia suffer from serious problems with memory. As a result of these problems, a patient may forget to maintain healthy family environment. One of the features of the system is the temperature setting, in order to achieve this, the patient should be able to open/close the windows or turn on/off the ventilator as often as desired.

Protect the house against potential thieves is another goal that the smart home should be responsible. The house should provide the illusion that there is always someone at home when the patient is out. Thus, the system should turn on/off the lights when the patient is away from home for a long time and it is night; when sunrise should turn off the lights. This would help prevent a thief enter the house.

Furthermore, the smart home should at the same time act against any potential thief when he is inside the house area. Thus, the smart home should always monitor the rooms of the house and check if a person is acting suspiciously (by entering foran unusual entry, for example). If this occurs, the smart home should many times as necessary to close all doors and windows and then call the police.

While the system manages the temperature and protects the house from thieves, the system must provide patient entertainment in various ways, such as watching movies (for this he should log into a site of film and then the film is shown on television) or view a TV channel or send messages to friends/relatives of the patient to visit him when they do not do more than a month.

For all these actions are taken by the system, it should perform the necessary monitoring.

The goal model of the smart home system is presented in next page.

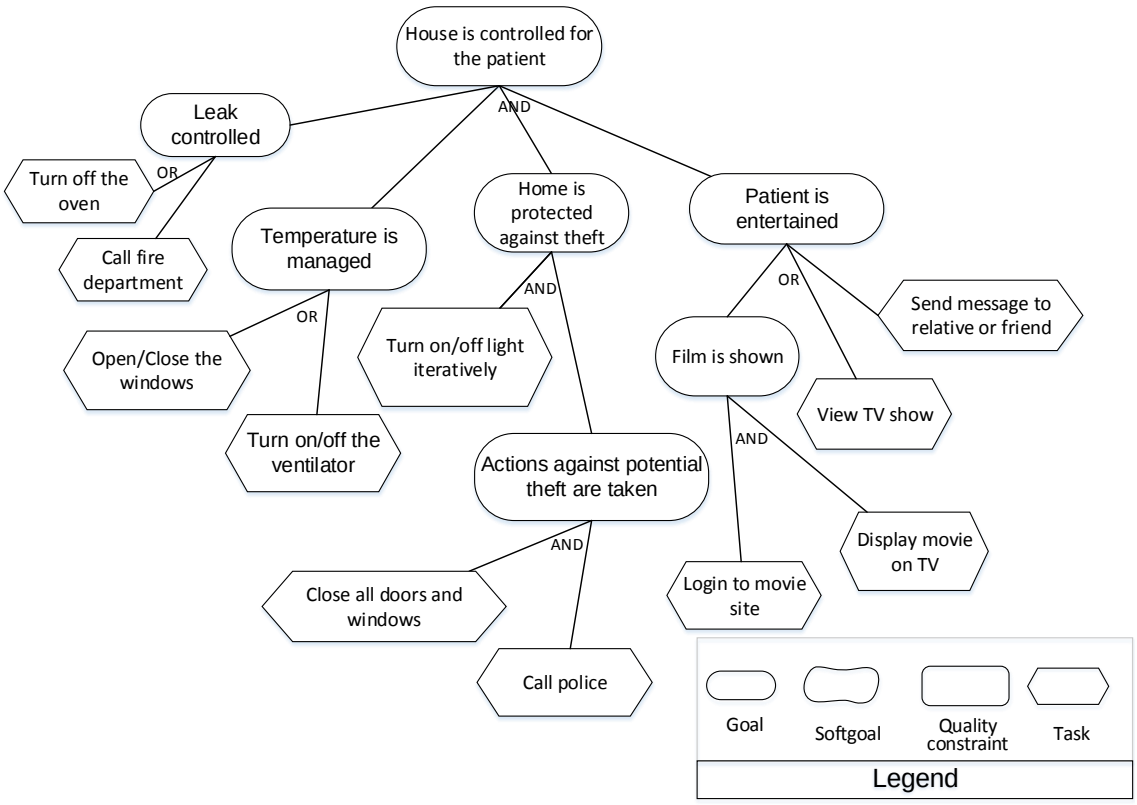
2. Specification of requirements identified in the design phase in the goal model:

2.1. Include the requirements below in the goal model:

The system must provide patient entertainment in various ways, such as watching movies (for this he should log into a site of film and then the film is shown on television) or view a TV channel or send messages to friends/relatives of the patient to visit him when they do not do more than a month.

2.2. Identify the contextual variation points from the conditions for the execution of tasks listed in the specification of question 1.

[1] Raian Ali. Modeling and Reasoning about Contextual Requirements: Goal-based Framework. Doctoral thesis, 2010.



3. Include in the goal model the tasks necessary for the adaptation of the critical objective of the system with their contexts and monitoring as described in the step 3 of the reference guide.

A critical requirement for the smart home is the gas leakage control, so the use of the oven should be controlled. Thus, the possible actions for the system are turn off the oven (when the patient finished using it) or notify the fire department (when it detects a leak).

4. Observe the order of execution of the tasks in the specification of question 1 and determine the system flow expressions, writing the expression in the objective model.

5. Using the flow expressions, create a statechart in Yakindu tool that represents the desired behavior of the system with the appropriate transitions. This statechart should not have any added functionality or less than is described in the specification. Consider the idle states for wait the context hold or the user entries.

D

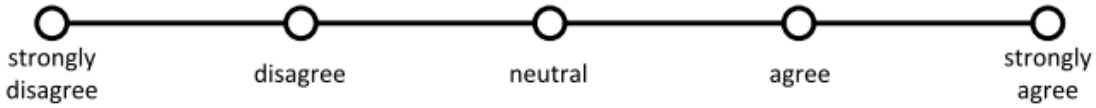
Post-Questionnaire Experiment

We applied a pos-experiment questionnaire in portuguese to the experiemtnal group. The aim of this questionnaire was to detect the subjects opinions about each process activity of GO2S process. The file that subjects received is presented in next page.

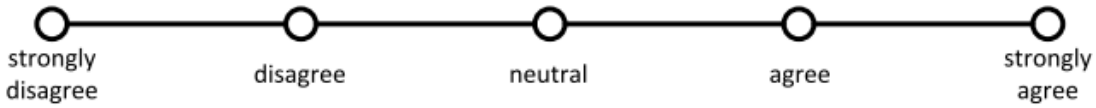
Derivation of statecharts from goal models to context-sensitive systems
Experiment
Group: **Using the process** Date: 03/12/2014
Anonymous questionnaire

For each one of the statements below, select whether you strongly disagree, disagree, is neutral, agree, or strongly agree.

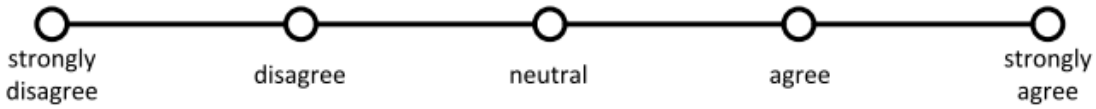
1. The process for statecharts derivation from goal models is understandable.



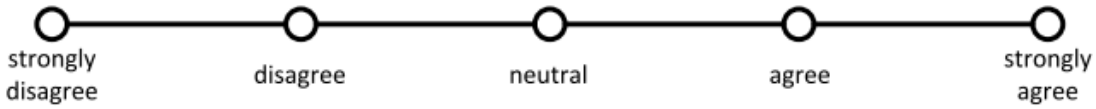
2. Step 1 (Construction of design goal model) is easy to understand.



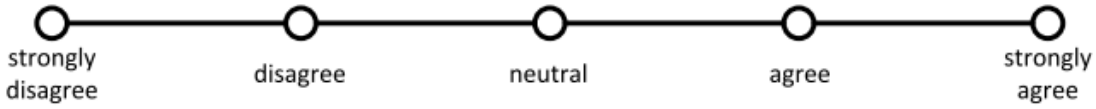
3. The notation of goal model is easy to understand.



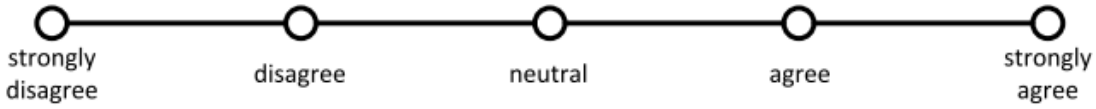
4. The use of goal models facilitates the creation of statecharts.



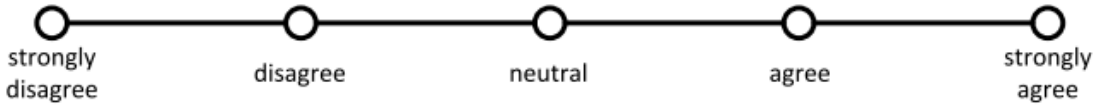
5. Step 2 (Specification of contextual variation points) is easy to understand.



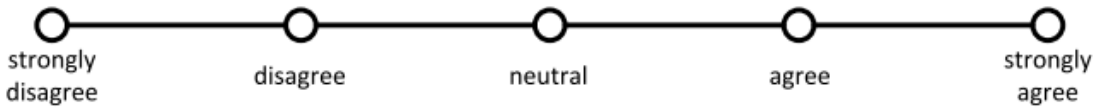
6. The notation for context specification is easy to understand.



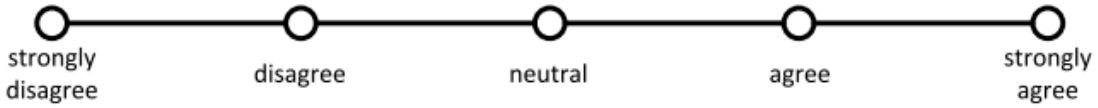
7. Step3 (Specification of monitoring and adaptation) is easy to understand.



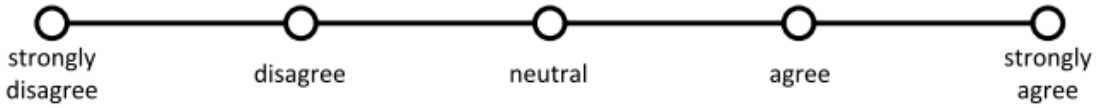
8. Step4 (Specification of flow expressions) is easy to understand.



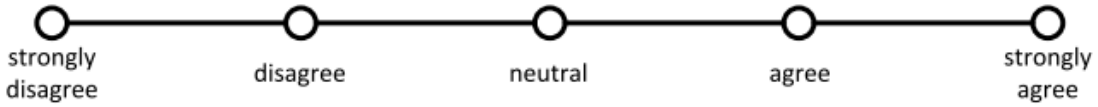
9. The use of flow expressions facilitates the creation of statecharts.



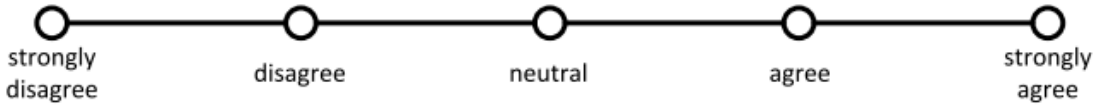
10. The use of flow expressions makes the creation of statecharts more systematic.



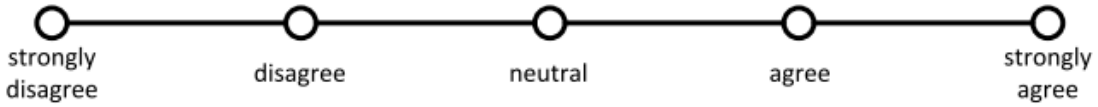
11. Step5 (Derivation of Statechart and refinement) is easy to understand.



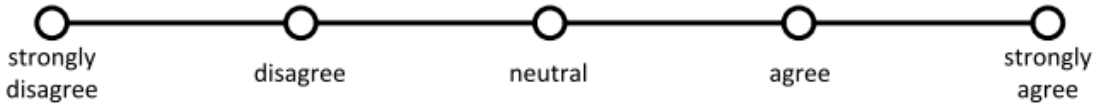
12. Statecharts makes easy to understand the system's behavior.



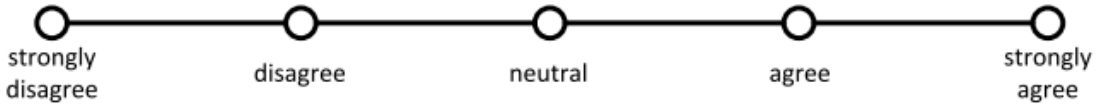
13. The creation of statecharts contributes to a more complete system specification.



14. The mapping of tasks to states facilitates the creation of statecharts.



15. The mapping between goals and super-states improves the organization of the statechart.



Do you like to perform some comment about the process?

_____
_____
_____
_____
_____

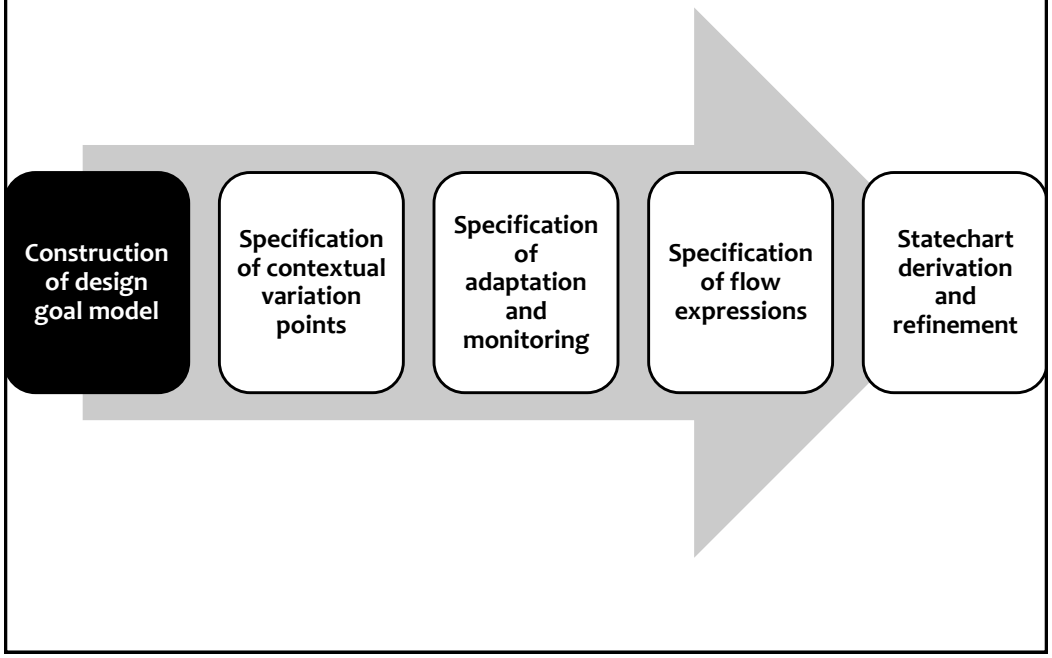
E

Reference Guide

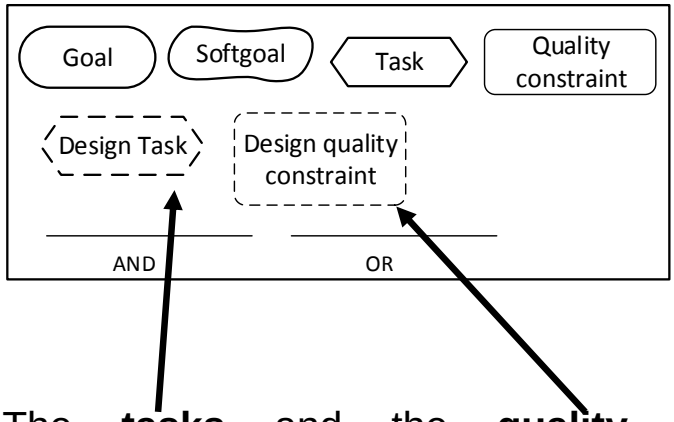
The group that applied the GO2S process received a reference guide in portuguese with a summary of the activities and the notations used by our process. This file is presented in next page.

Reference Guide

Overall Process



Goal Model[Notation]



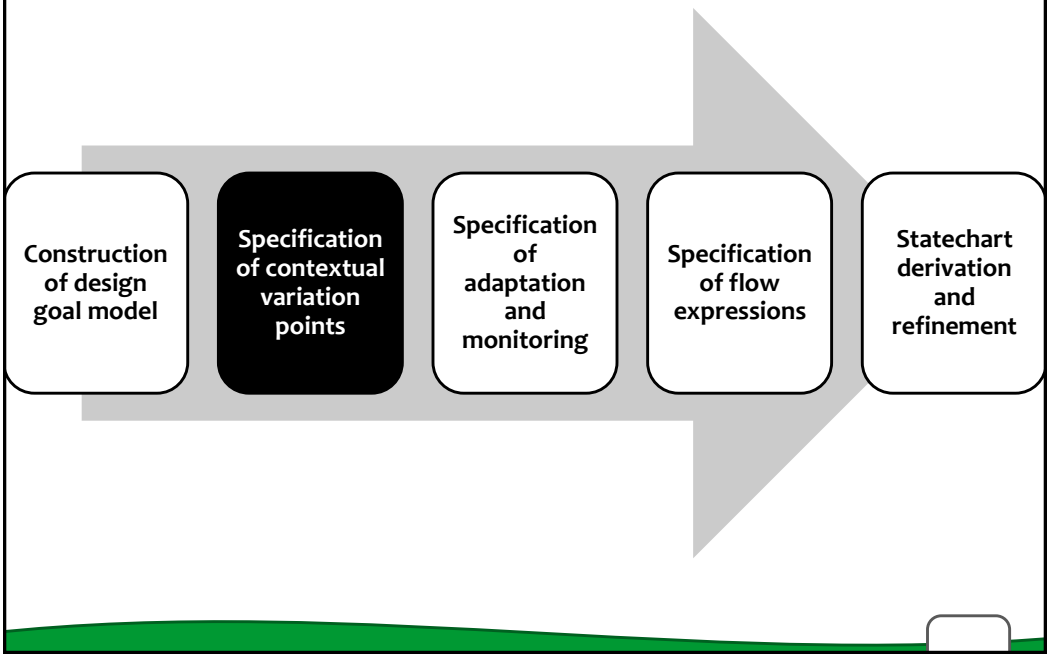
The **tasks** and the **quality constraints** identifified in the **design** phase are represented with the dashed notation**.**

1° Activity of the process

- Steps:
  - Identify the requirements appeared in the design phase such as
    - Non-functional requirements
    - *Design tasks*
    - *Design quality constraints*
  - Assign a design task (that will not be executed by the system) to an user (if there is any task of this type)

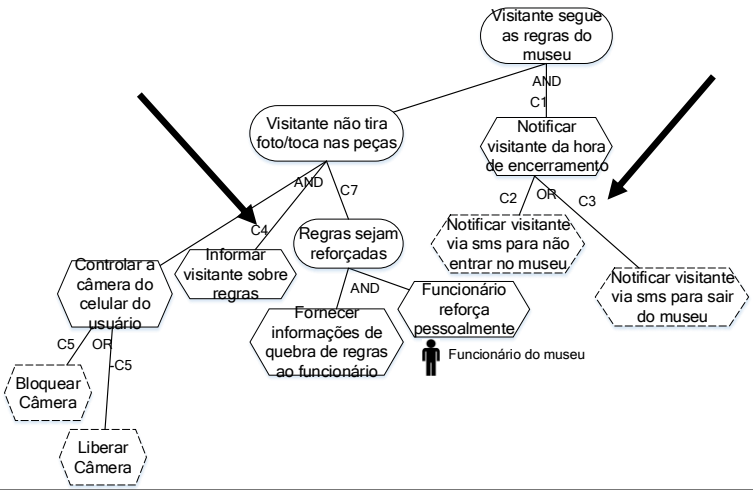
Example:

Overall Process



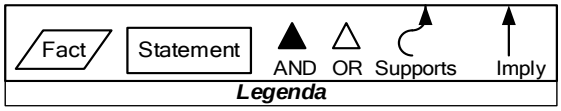
Contextual Goal Model [Notation]

The contextual variation points **can appear in AND/OR** refinements and they are specified through a **label (C1, C2.. Cn)** in the goal model.



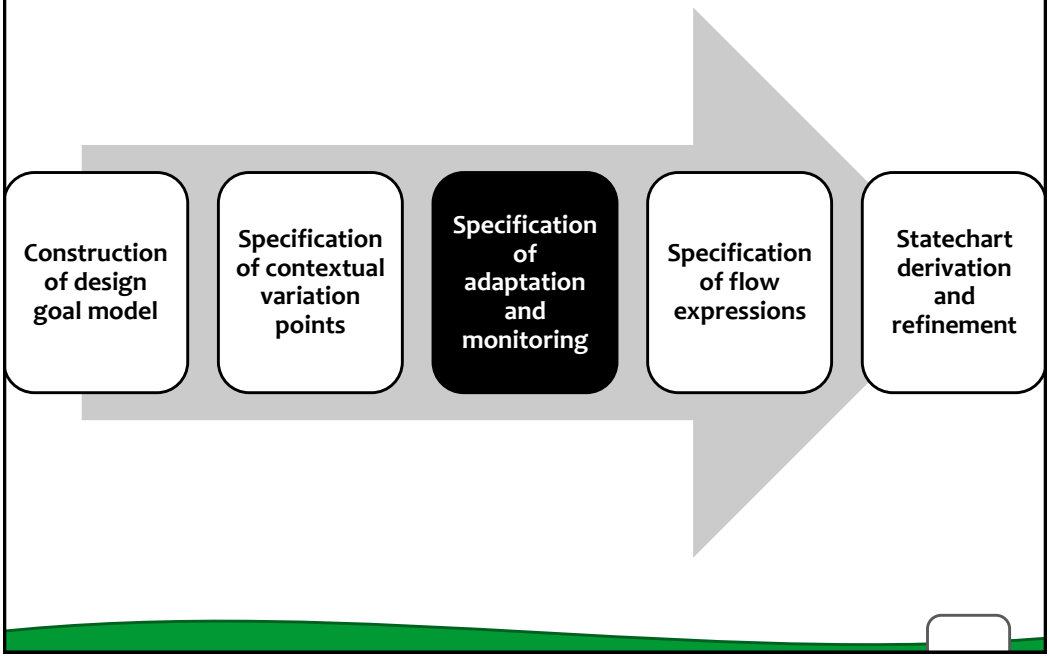
2° Passo do processo

- Steps:
  - Identify and specify the contextual variation points.
    - tip: look for situations that represent conditions for a task to be executed.
  - Refine each context using the following notation



Supports: used to connect *facts* and *statements*.
Imply: used to connects *facts* or *statements* to the context.
Fact: it is possible to check if its value is true ou false directly.
Statement: its value (V or F) is infered through the *facts*.

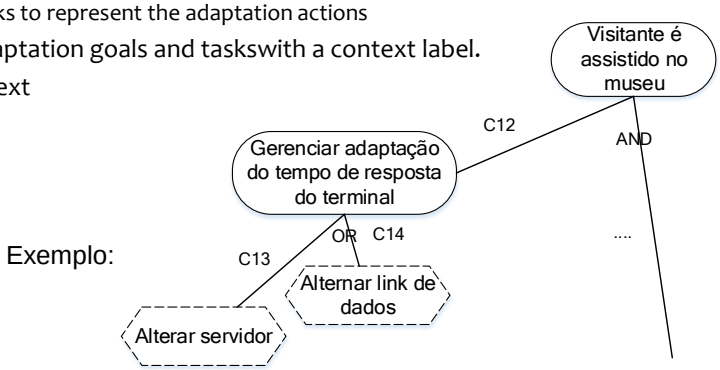
Overall Process



3° Passo do processo

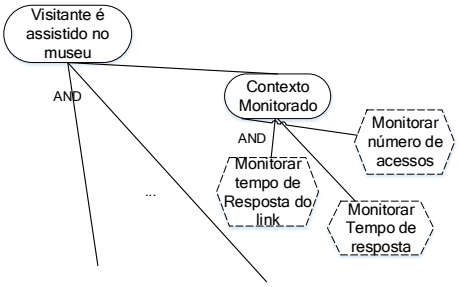
- Steps:
  - Define the requirements that requires na action in case of failure (critical requirements for the system)
  - Represente the adaptation management
    - Adid a new goal in the root node to the management
    - Add subgoals for each critical goal that must be monitored and adapted (in case of more than one critical requirement)
    - Add design tasks to represent the adaptation actions
  - Associate the adaptation goals and taskswith a context label.
  - Refine each context

Exemplo:

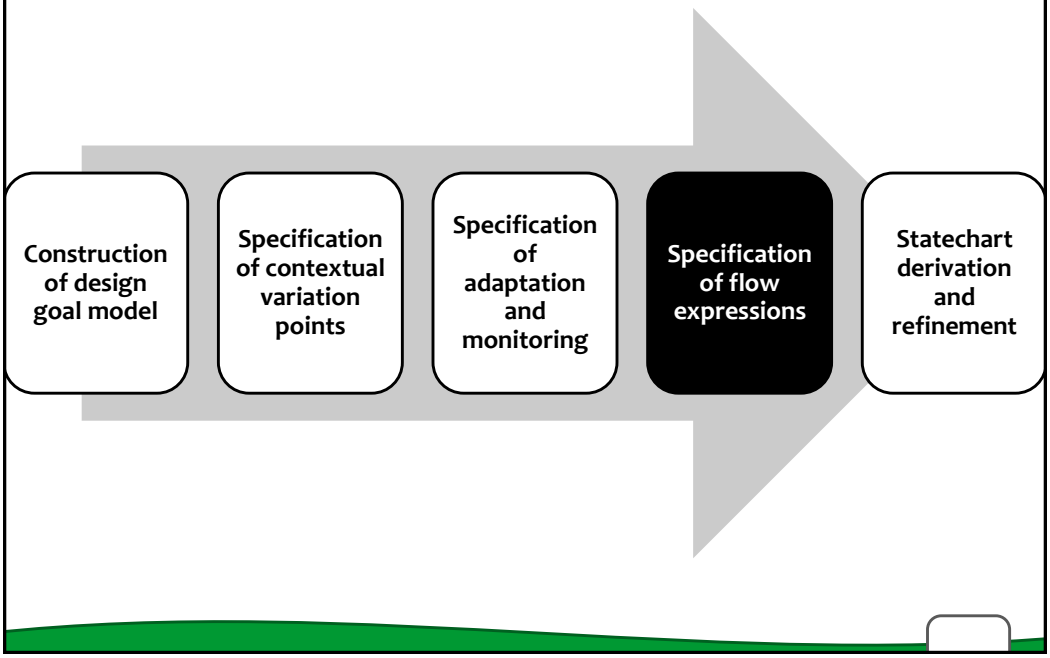


3° Step of the process

- Steps:
  - Identify the dynamic contextual elements (properties of real-world present in the refinements of the contexts facts that need to be monitored all the time from the system because its value constantly changes at runtime)
  - Represent the context monitoring
    - Add a new goal in the root node
    - Adddesign tasks to monitor each dynamic contextual element

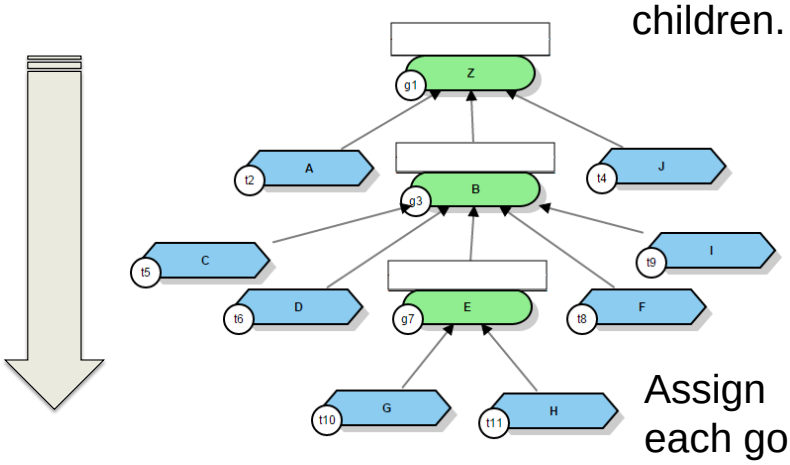


Overall Process



Top-down

Write in the parent the bevhavior of its children.



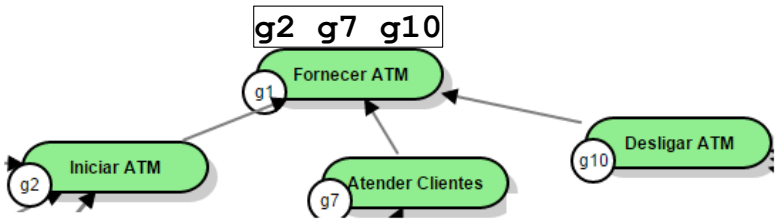
Assign a code for each goal and task.

Flow Expressions [Notation]

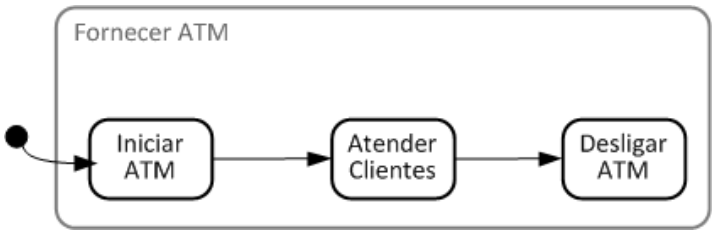
- **A B** ➔ sequence (first A, then B)
- **A | B** ➔ alternative (A or B)
- **A?** ➔ A is optional
- **A+** ➔ repetition (1 or more times)
- **A*** ➔ repetition (zero or more times)
- **A – B** ➔ A and B in parallel (ortogonal)
- **i1** ➔ idle state

1) Sequence (result)

- **Goal model:**

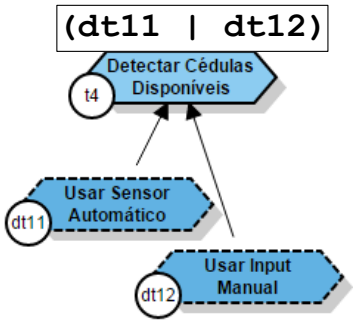


- **Statechart:**

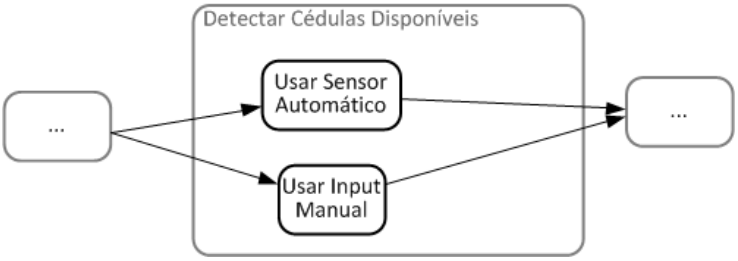


2) Alternatives (result)

- Goal model:

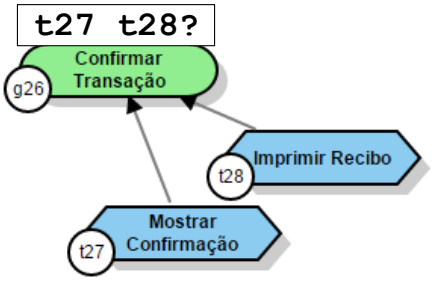


- Statechart:

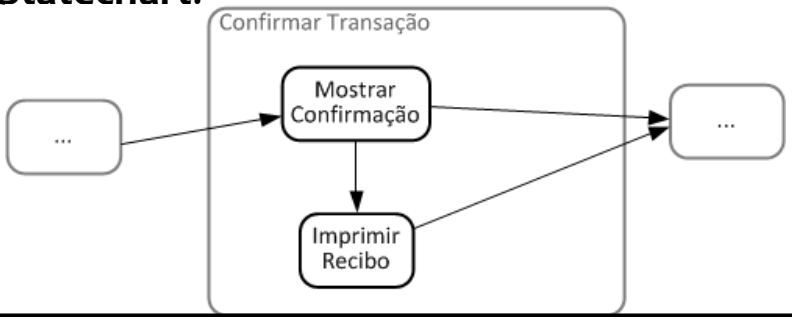


3) Optional (result)

- Goal model:

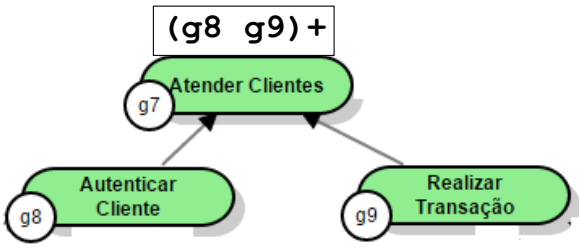


- Statechart:

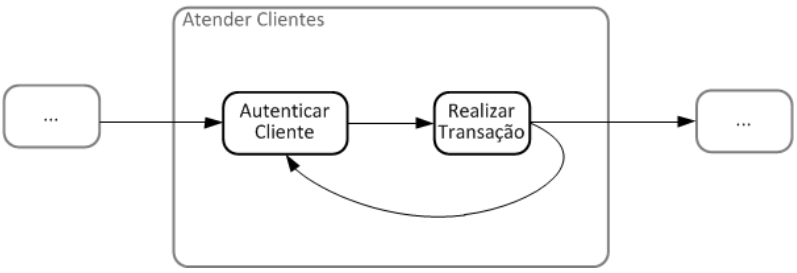


4) Repetition 1 or more (result)

- Goal model:

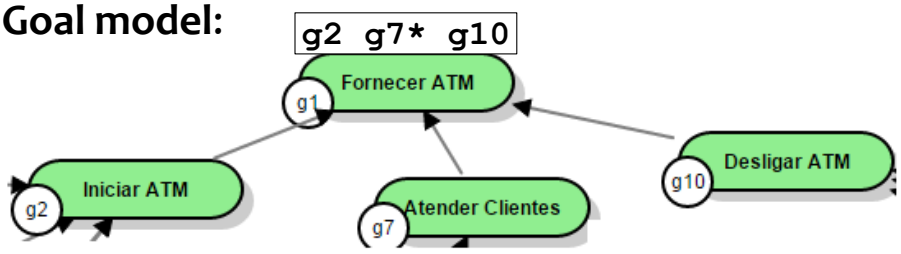


- Statechart:

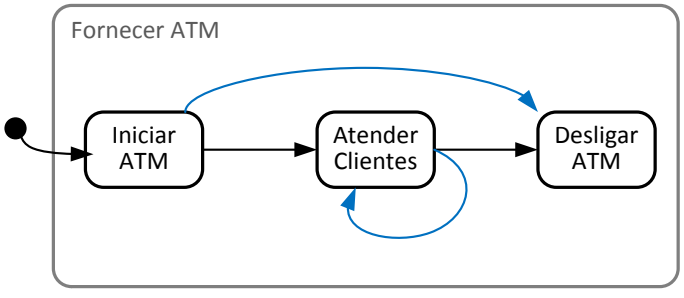


5) Repetition 0 or more (result)

- Goal model:

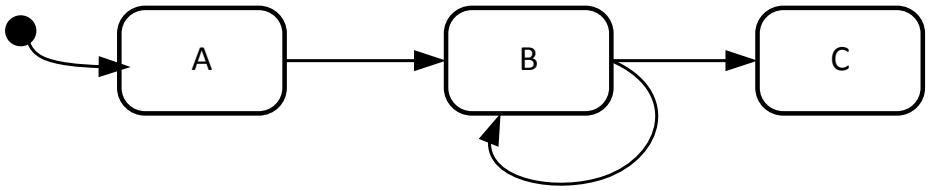


- Statechart:

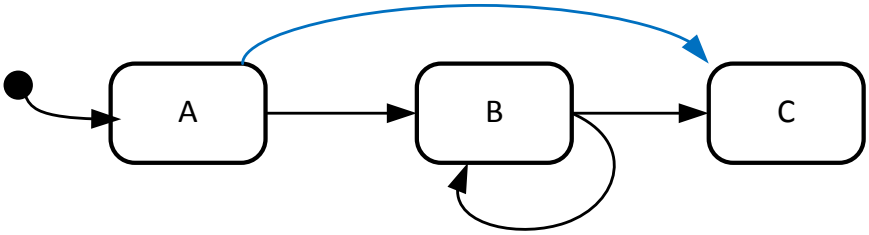


Comparison between 1+ e 0+

- A B+ C (1 or more times)

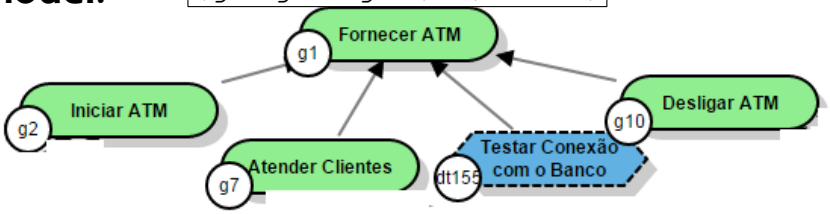


- A B* C (zero or more times)

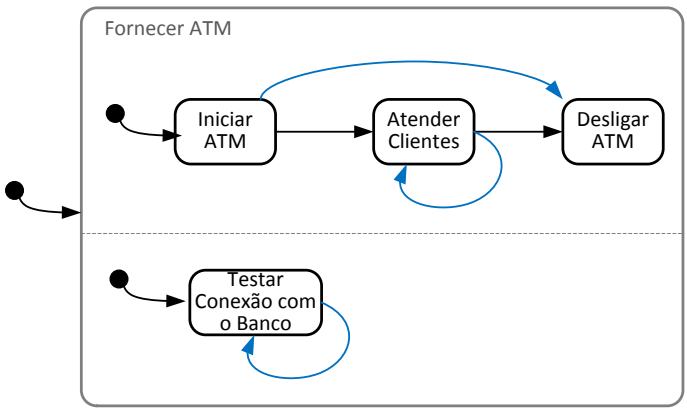


6) Paralell (result)

- **Goal model:** `(g2 g7* g10)-(dt155)`

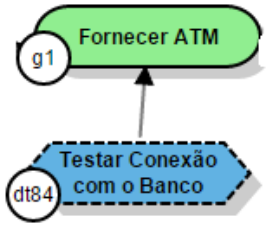


- **Statechart:**



7) Idle states

- **Case:** when a state in which nothing will be done is needed - the system is just waiting for the occurrence of any event or context to be true.
- **Example:**

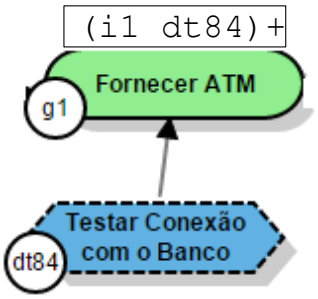


- **Expression:**
  - Use *iX* (where X is a number) to add a idle state:

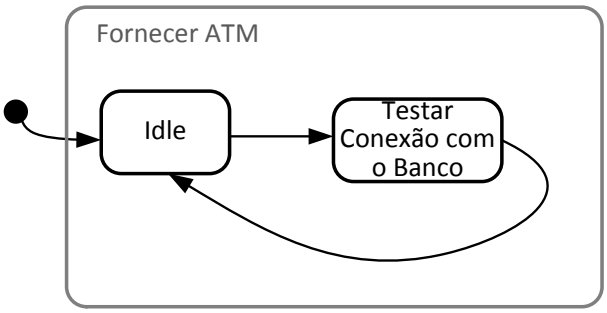
```
(i1 dt84)+
```

7) Idle states (result)

- **Goal model:**



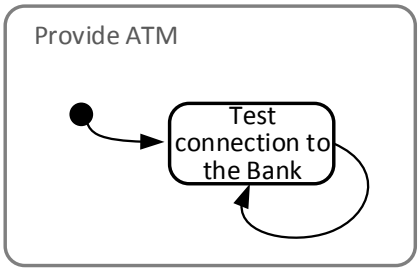
- **Statechart:**



Comparison...

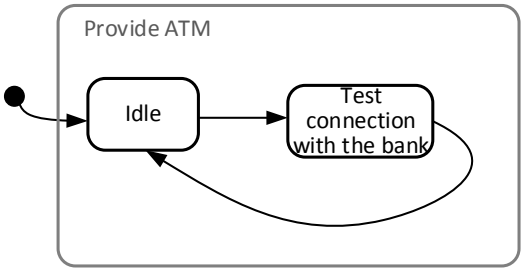
- Without *idle*

`dt84+`

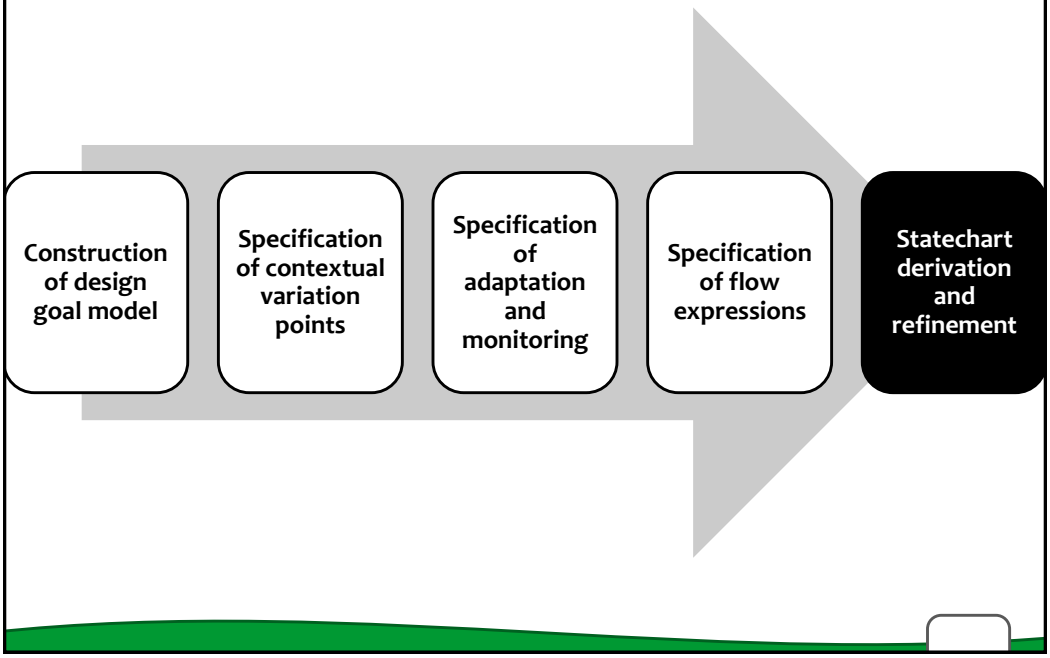


- With *idle*

`(i1 dt84)+`



Overall Process

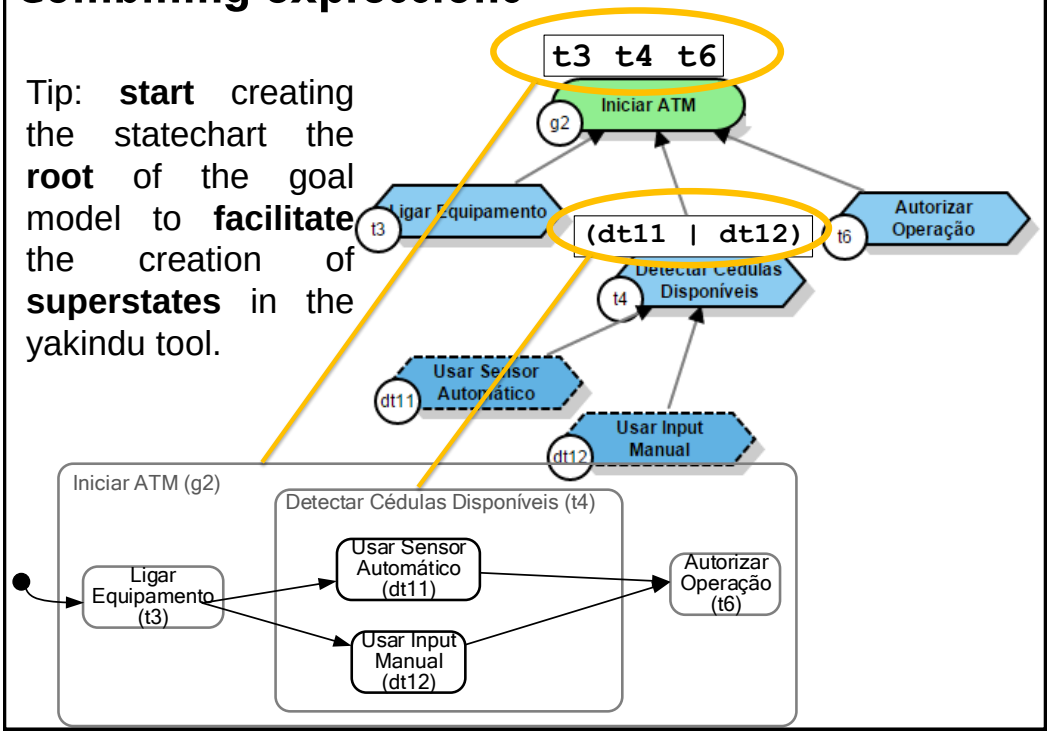


5° Step of the process: Derivation strategy

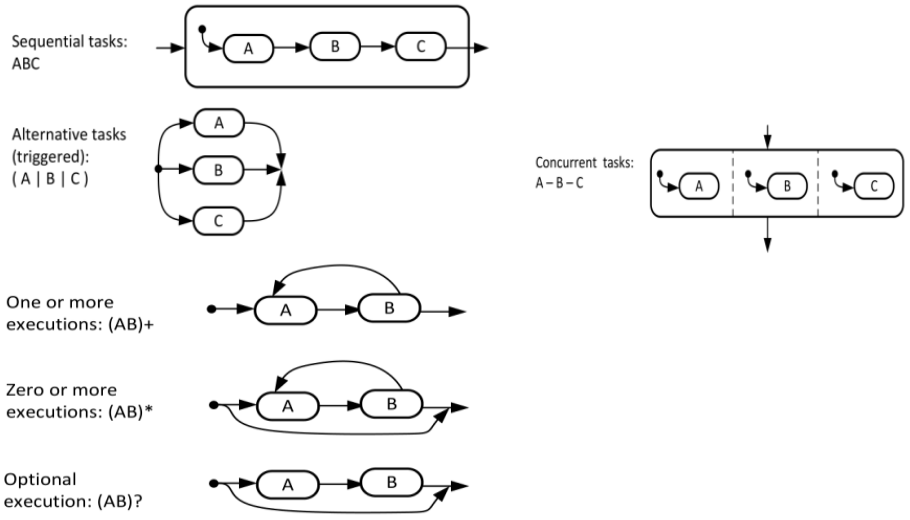
- Create statechart from the goal model:
  - Model the **goals/tasks** in **refinements** as **super-states**
    - The children elements of these elements should be sub-states of the parent.
  - Model the **tarefas states**
  - Criate **idle states** to wait until the context be true to occur state change.

Combining expressions

Tip: **start** creating the statechart the **root** of the goal model to **facilitate** the creation of **superstates** in the yakindu tool.



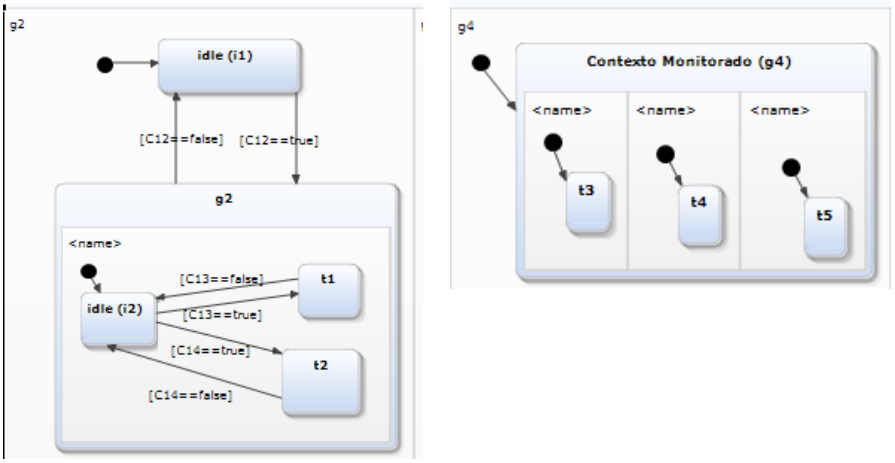
5° step of the process: statechart derivation patterns



5° step of the process: events and conditions

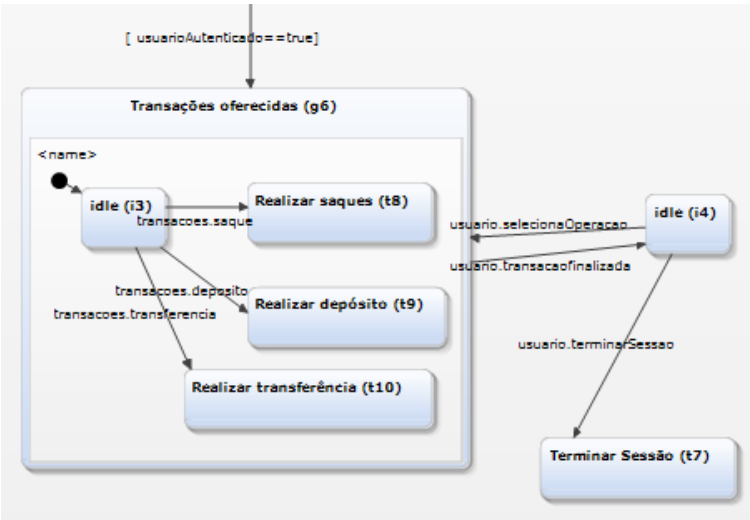
- After generating the statechart, it is necessary to write the events and conditions of the transitions.
- The context monitoring and the management of adaptation actions should be represented in parallel.

Example:



5° step of the process: events and conditions

Exemplo:



Sources of inspiration for defining events

- Context is true/false
- Task accomplished
- Task request by a user
- Timer