Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# An Extended Goal-oriented Development Methodology with Contextual Dependability Analysis

Danilo F. Mendonça

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof. Dr.ª Genaína Nunes Rodrigues

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

**CIP — Catalogação Internacional na Publicação**

Mendonça, Danilo F..

An Extended Goal-oriented Development Methodology with Contextual Dependability Analysis / Danilo F. Mendonça. Brasília : UnB, 2015.
61 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2015.

1. LaTeX, 2. metodologia científica

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília

Instituto de Ciências Exatas
Departamento de Ciência da Computação

# An Extended Goal-oriented Development Methodology with Contextual Dependability Analysis

Danilo F. Mendonça

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Brasília, 30 de janeiro de 2015

# Dedicatória

# Agradecimentos

# Resumo

A static and stable operation environment is not a reality for many systems nowadays. Context variations impose many threats to systems safety, including the activation of context specific failures. Goal-oriented software-development methodologies adds the 'why' to system requirements, i.e., the intentionality behind system goals and the means to meet then. Contexts may affect what requirements are needed, which alternatives are available and the quality of these alternatives, including dependability attributes. In order to allow a formal and probabilistic analysis of systems affected by context variation and elicited with Goal-Oriented Requirements Engineering (GORE) approach, we have proposed an extension to the TROPOS methodology to associate dependability constraints to goals and to provide a more precise and formal non-functional requirements (NFR) verification by translating a contextual goal model (CGM) annotated with a behavioural regular expression into a probabilistic model to be checked against properties defined with the Probabilistic Computation Tree Logic (PCTL). We evaluated the proposed TROPOS extension with a case study of a Mobile Personal Emergency Response System (MPERS).

**Palavras-chave:** LaTeX, metodologia científica

a

# Abstract

**Keywords:** LaTeX, scientific method

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introduction

## 1.1 Problem Definition

Among the different causes that lead a system to fail, some can be tracked back to design decisions in early system development process. A systematic requirements engineering process may improve the quality and precision of the delivered documents, models and specification. Nonetheless, many methodologies for system development do not further investigate or verify if the designed system-to-be conforms to non-functional constraints.

Dependability requirements are NFR related to the correct system operation. Dependability defines the ability of delivering a service that can justifiably be trusted. Fail forecasting should provide a qualitative and/or a quantitative evaluation of the system behaviour in respect to fault occurrence or activation, while fault removal includes the verification, diagnosis and correction [AVIZIENIS]. For some systems, late fault correction may be very costly and, in the worse case, may only happen after the manifestation of a catastrophic failure [DEPENDABILITY].

In traditional Goal-oriented requirements engineering (GORE) methodologies [GORE CAC], contribution analysis is based on domain knowledge about the positive, neutral (implicit) or negative impact of a given system alternative to one or more system goals, generally a qualitative goal. By comparing the overall contribution of two or more alternatives, a decision is made about which one should be adopted for the system-to-be. For instance, if one goal is to communicate with a remote user mobile, alternative means for the notification agent could be to send a SMS, an internet based message or a voice call. These alternatives may contribute with different values for qualitative goals such as 'reliable delivery', 'fast delivery', 'convenient delivery', etc.

In TROPOS, a contribution analysis aims to select the best alternative based on positive or negative contribution to softgoals. This analysis is limited in many ways. First, it is based on domain knowledge information that may not exist or may not be precise and reliable. As a consequence, the decision of which alternative to use may be biased and lead to unexpected violations at runtime - the selected alternative may prove to be inappropriate. From this, our first research question emerges:

**Research Question 1** Is it possible to improve the precision of the non-functional verification in TROPOS methodology?

Second, contribution analysis is based on a static representation without any runtime specification required for estimating non-functional metrics that depend on system behaviour and its many nuances such as temporal order, cardinality and priority. From this, our second research question arises:

**Research Question 2** Is it possible to complement a goal model with a behaviour specification concerning, e.g., the temporal order, cardinality and priority of the achievement of goals and the execution of tasks?

Third, contribution links are deterministic. As such, the design decisions based on contribution analysis are reduced to a simple sum comparison of the concurring alternatives contributions with no support for probabilistic verification. Accordingly, our third research question is:

**Research Question 3** Is it possible to use a probabilistic approach for the verification of non-functional requirements?

Fourth, TROPOS provides no support for the estimation of non-functional metrics in temporal frames such as “the percentage of success for a given goal in a month”. This leads us to our fourth research question:

**Research Question 4** Is it possible to estimate time-bounded non-functional metrics related to the achievement or failure of system goals?

Additionally to the contribution analysis limitations, the context in which systems operate may not be static. Mobile and pervasive computing, among others, are examples of new computer paradigms for which the environment is dynamic. Battery, signals strength, components availability and the quality of physical resources and relevant information such as the user geographic location may vary through time, posing a new challenge to the development of socio-technical systems based on these paradigms. The contextualization of the informations gathered at RE phase becomes imperative once its validity may be threatened by changing environment conditions [CGM]. From this, our fifth and last research question arises:

**Research Question 5** Is it possible to consider the context effects described by the CGM in the verification process?

## 1.2 Proposed Solution

Static model-based verification is a formal approach to evaluate metrics over a finite-state model representing the system behaviour. The advantage of this approach is to reduce the correction costs by anticipating failures at early stages of the development cycle using not the system itself, but its representation. As long as the system model is precise, this method may provide a reliable evaluation of non-functional metrics such as dependability attributes.

Goal models are not restricted to strategical goals. Through AND/OR-decomposition, goals are further detailed and means-end tasks are responsible for the operationalization of leaf-goals. Thus, tasks may be directly mapped to activities that composes the system behaviour. A probabilistic, model-based verification of a goal model, as it will be explained in later sections, addresses our RQ1.

As the goal model proposed in TROPOS is static, no information regarding achievement/execution order, cardinality and priority of goals/tasks is available, except the activity diagram for the detailing of an agent's single capability behaviour and the sequence diagram for agents interaction. Nonetheless, this problem was tackled by Dalpiaz et al. with a regular expression language to specify the behaviour in a goal model, e.g., how many times the same goal should be achieved and the execution order of different system tasks [RGM]. This answers our RQ2.

A probabilistic model checking (PMC) approach has already been explored and is supported by tools such as PRISM model checker[genaína PMC, PRISM]. PMC is a formal method for static verification of different properties, including non-functional metrics like reliability. In order to verify non-functional requirements for the system-to-be, a corresponding probabilistic model of the system behaviour must be built. In this work, the RGM proposed by Dalpiaz et al. is used as input for the generation of a DTMC model and non-functional requirements are verified with the PMC technique proposed by Uriel [URIEL]. This probabilistic approach addresses our RQ3.

PRISM is a model checker tool that supports PCTL property verification. PCTL provides the syntax for time-bounded probabilistic properties such as "what is the probability of failure of a given goal in the next x time units?". This is useful for scenarios with restrictions affected by time, like the battery of a mobile device. The use of PCTL properties addresses our RQ3 (probabilistic) and RQ4 (temporal).

Finally, to address the problem of a dynamic context of operation, the context effects over goals, means and metrics should be treated by the verification model. Context variables may be parametrized to produce a formula that can check the system and its alternatives for different contexts. The feasibility of this approach, as presented by later chapters of this work, answers RQ5.

Runtime self-adaptation is beyond the scope of this work. However, based on the contextual analysis provided by the CGM and the enriched non-functional and dependability analysis provided by the verification of different alternatives using the PMC technique, it should not be difficult to extend the approach with the additional monitoring, planing and execution capabilities of a self-adaptation loop and have a self-adaptive architecture and mechanism reflected upon its runtime goal model requirements. These concerns should be addressed in future work.

## 1.3 Evaluation

This proposal was evaluated with the application of the extended TROPOS methodology to the development of a Mobile Personal Emergency Response System (MPERS). This system may be seen as a body area network (BAN) with extended functionalities related to ubiquitous emergency response running in a mobile device [BAD]. Instead of static environment, the MPERS is conceived to allow patients with different health risk degrees to maintain mobility while they are monitored and assisted. MPERS features were based on real emergency response systems available at the industry and also at the BAN explored in previews work [Fernandes].

The evaluation process was focused in revealing the major benefits and limitations of the extended TROPOS proposal. Time to market is an important aspect for any software development methodology. Also, the soundness and precision of the proposed probabilistic verification is crucial and must be evaluated as they should not result in mislead decisions about which alternatives should be used by the system. Instead, they must anticipate any violation that could lead to a system failure, specially severe or catastrophic failures, giving analysts valuable information about where the system requirements and specification should be tailored and improved.

## 1.4 Contributions Summary

This section summarizes the contributions of this proposal.

1. A formal approach for the verification of non-functional requirements as an extension of the TROPOS Goal-oriented software development methodology.

    - A new syntax for the specification of non-functional constraints as qualitative hard goals.
    - Use of a probabilistic model checking technique for the estimation of non-functional metrics and for the conformance verification of a given system alternative to non-functional constraints.
    - Inclusion of context effects over goals, means and metrics in the verification model using appropriate constructs and parameters for each case.

2. An automated generation of the PRISM probabilistic model representing activities from a runtime goal model annotated with the runtime regex and graphically modelled using the TAOM4E tool that supports TROPOS methodology.

    - A parser implementation for the regular expression (regex) language used in runtime goal models with support for execution order, cardinality, alternative execution, optional execution, conditional execution and multiple executions of the same task.
    - Definition of conversion rules between different decomposition types and behaviours rules in a runtime goal model to a probabilistic model in PRISM language.
    - Implementation of a RGM-to-DTMC generator integrated to the TAOM4E Eclipse plugin that supports the TROPOS methodology with a graphical environment for goal modelling and analysis.

## 1.5 Document Organization

This dissertation is organized as follows. Chapter 2 presents the base concepts of this work. Chapter 3 describes the most important related works. Chapter 4 details the motivation for this work and the requirements for the proposed TROPOS extension. Chapter 5 describes the TROPOS extension for NFR verification using a PMC technique. Chapter 6 presents the rules for the automatic generation of a DTMC model in PRISM language from the contextual goal model with runtime regex. Chapter 7 evaluates the proposal and describes its benefits and limitations. Finally, Chapter 8 concludes this work with final considerations about the current proposal and our future work.

# Capítulo 2

# Baseline

## 2.1 Goal-oriented Requirements Engineering

Goal-oriented requirements engineering brings forward the intentionality behind system requirements. More than just presenting the *what* and the *how* of a system-to-be, it provides the justification for each requirement, that is, they also present the *why*. Through a directed graph tree that begins with a root goal, goals are connected trough decomposition links. Root and higher level goals are related to strategical concerns, while lower level and leaf-goals are related to technical and operational features of the system.

The main purpose of a goal model is to support the early process of RE, including the elicitation of social needs and dependencies, the actors involved in delivering functionalities and resources, the decomposition of higher-level goals into more granular and detailed requirements chunks, the operationalization through means-end tasks and finally the comparison between different alternatives for the system-to-be. A goal model is said to be valid and complete if it follows all its syntactic rules and if all system goals are either decomposed, delegated to other actors or fulfilled by operational system tasks.

Three frameworks/methodologies, namely KAOS, i* and TROPOS, represent the foundations for the goal model analysis used by a variety of other proposals [KAOS, i*, TROPOS]. Despite some differences among their syntax, they all share a set of core concepts:

Entities

- **Actor:** an entity that has goals and can decide autonomously how to achieve them. They represent a physical, social or software agent. E.g.: A patient, an emergency center, a doctor and a Mobile Personal Emergency System running in patient's smartphone.

- **Goal:** are actors' strategic interests. A goal with a clear-cut criteria for its satisfaction is called a hard goal. In opposition, softgoals has no clear-cut criteria for deciding whether they are satisfied or not and are usually associated to non-functional requirements of an actor. E.g.: vital signs are monitored, emergency is detected, emergency center is notified (hard goals) and system availability, detection precision, emergency awareness (softgoals).

- **Task:** an operational means to satisfy actors' goals. E.g.: monitor temperature sensor, persist vital signs data, request emergency assistance.

<u>Relations</u>

- **AND/OR Decomposition:** a link that decomposes a goal/task into sub-goals/sub-tasks, meaning that all (at least one) decomposed goal(s)/task(s) must be fulfilled/executed in order to satisfy its parent entity.

- **Means-end:** a means to fulfil an actor's goal through the execution of an operational task by the same actor.

- **Contribution link:** a positive or negative contribution between a given goal/task to a softgoal. Contribution links are used for deciding between alternative goals/tasks at design time (contribution analysis).

## 2.2 TROPOS Goal-oriented Software Development Methodology

TROPOS is a GORE methodology based on the i* framework [TROPOS]. Its main improvement is the addition of new phases of requirements engineering and system design, namely:

- Late requirements engineering: Beyond the social dependency modelling with actors diagrams representing stakeholders and their needs in early requirements phase, a late requirements phase focuses on the system actor analysis. In this phase, system goals are inherited from stakeholders needs and represent both functional and non-functional requirements. Each goal has to be further decomposed in more granular sub-goals, delegated to other actors or to be fulfilled by means-end tasks.

- Architectural design: In this phase, new actors representing sub-systems are created to fulfil different system goals. The idea is to shape the solution using a multi-agent architecture style instead of a monolithic system approach. Data and control interconnections are represented as dependencies.

- Detailed design: The last phase is characterized by the specification of agent capabilities and interactions though UML activity and sequence diagrams. Also, the implementation platform and other specific implementation details are addressed in order to directly map the design to system code.

Implementation phase is also specified by TROPOS methodology, but it is out of the scope of this work as our objective is to improve the analysis and the solution that will be later implemented.

## 2.3 Contexts

Context may be defined as the reification of the environment that surrounds the system operation [FINKElSTEIN]. Contexts, as already stated, may not be static, but dynamic. A system has no control over its context of operation. Accordingly, a system must be able to support different contexts of operation without violating its goals. Moreover, systems should be able to monitor the state of its surrounding environment and decide which alternative will be used regarding both the availability of that alternative and the optimization of non-functional requirements.

In GORE, dynamic contexts may affect what goals a system have to reach, the means available to meet them and also the quality achieved by each alternative[CGM]. Root goal and higher level strategical goals are not contextualized as they represent the main purpose of a system [Finkelstein]. As these goals are decomposed in more granular sub-goals, a context condition may dictate:

1. If the goal is required for that context, limiting 'what' a system should do;

2. If a sub-goal or task is adoptable, limiting the 'means' to fulfil a required goal;

3. The positive, neutral or negative contribution of using some goal or task to another goal, usually a qualitative softgoal;

The third effect is the main focus of this work, as it is related to the GORE contribution analysis that we aim to improve. In our proposal, we extend the concept of the context variation effect to non-functional metrics of goals and tasks - as the components used for concrete task execution may also be affected by context variations, e.g., the reliability of a sensor in different temperatures.

## 2.4 Variability in GORE

Given the possibility of an OR-decomposition in a goal model, more than one alternative can exist in terms of which subgoal should be achieved to satisfy its upper goal, which subtask should be executed to satisfy its upper task or which task should be executed to satisfy its upper goal. Accordingly, multiple paths may lead to the satisfaction of the root goal. They are called alternative behaviours, or alternatives.

Solving the variability problem with goal models have different meanings according to the life cycle phase it takes place. At design time, multiple alternatives are elicited, but not all are selected to be part of the system-to-be. At runtime, system inherits variability in order to adapt to different contexts of operation. For each context, one adoptable alternative must be selected.

Our proposal aims to solve the variability in goal models at design time through static verification of system models for multiple contexts of operation. For this, analysed contexts are iterated and alternatives are selected based on their contextual conformance to non-functional requirements in terms of softgoals and non-functional metrics.

### 2.4.1 Different paths, one context, one solution

Despite the elicitation of more than one path, only one alternative is kept at design time. Softgoals are generally used as criteria for the alternative selection [i*, TROPOS, KAOS]. We extend TROPOS with a model-based verification of non-functional constraints. Verification should point out at design time:

1. Which alternative conforms to the non-functional metrics associated to the goal model.

2. The best alternative given one or more metrics. In the last case, a multi-criteria approach should decide which alternative will be selected for the system-to-be.

### 2.4.2 Different paths, multiple contexts, multiple solutions

A system affected by context variation may have to select different alternatives for different contexts, justifying variability in the system-to-be design. Solving variability becomes more complex as for each context a different alternative may provide the best contribution to softgoals and achieve higher values for non-functional metrics.

In our work, variability solving for multiple contexts involves an iterative verification in which contexts are analysed for all adoptable alternatives. At least one alternative must be able to fully satisfy both functional and non-functional requirements of the system.

## 2.5 Dependability Analysis

The concept of dependability is related to dependence and trust as well as the ability of a system to avoid failures that are more frequent and more severe than certain threshold [AVIZIENIS]. According to Avizienis et al., dependability encompasses the following attributes:

- Availability: readiness for correct service.
- Reliability: continuity of correct service.
- Integrity: absence of improper system alterations.
- Safety: absence of catastrophic consequences on the user(s) and the environment.
- Maintainability: ability to undergo modifications and repairs.

A holistic dependability specification has to include not only the software operation, but also the requirements for which that operation is meant. Non-functional metrics are an important factor to decide the acceptable frequency and severity of a software or hardware failure.

A failure is a perceived deviation from system expected behaviour that may have variable degrees of consequence on the user(s) and the environment. These failures are caused by specification faults or specification violations. In the first case, requirements model fails to describe the system: either the goals or the means to fulfil then are incorrect or incomplete. In the second case, software or hardware behaviour did not follow its specification due to a natural phenomena, a human-made fault, a malicious fault or an interaction fault [AVIZIENIS].

The scope of this work is restricted to specification violations, i.e., we assume that a system specification is complete and consistent. Failures are restricted to anomalous behaviour of the components participating in the execution of system tasks, including technical components and human actors. Regarding the different means to attain dependability, our proposal consists of a fault forecasting as part of the Validation & Verification phase of RE by estimating metrics related to dependability and assuring their conformance to the dependability constraints associated to the goal model.

## 2.6 PRISM Probabilistic Model Checker

A model checking is a formal method that aims to automatically verify if a system model meets its specification for defined properties. Probabilistic and state based model checking supports the verification of finite-state probabilistic models such as discrete-time Markov chain (DTMC), continuous-time Markov chain (CTMC) and Markov decision process (MDP). Different types of properties enables the verification of a vast range of non-functional metrics.

The PMC technique used in this approach is supported by the PRISM model checker tool [PRISM]. PRISM allows the modelling and analysis of systems which exhibit random or probabilistic behaviour. The decision of using PRISM as the probabilistic state-based model checker was due to the number of successful case studies that have used this tool, indicating its maturity [PRISM CS].

PRISM is suitable for many different kinds of model evaluations depending on the abstraction level, the type of probabilistic model and the PCTL properties to be analysed. PRISM language offers a rich set of constructs that may represent system modules and components, among others architectural and design configurations. Both qualitative and sensitive analysis are available.

As it will be explained in later sections, goal models may extended with the behaviour specification required for the verification of some important dependability attributes. The objective is to anticipate non-functional dependability violations and to support the decision of which alternatives to use in the system-to-be. A model checking technique should be used for dependability analysis as long as:

- A formal system model may be built;
- Properties representing dependability attributes may be defined;
- The analysis overhead is justified, e.g., by its criticality.

Finally, PRISM also supports a parametric model verification. That is, instead of providing the final evaluation for a given property, models may use parameters instead of initialized variables and the verification will output a parametric formula whose evaluation will estimate or verify the model for any valid combination of parameters values.

## 2.7 Mobile Personal Emergency Response System

The MPERS case study will be further detailed in later chapter as the proposed TROPOS extension is described with the MPERS goal models of previous requirements engineering phases. As such, this section will cover some relevant aspects of this system that justify the use of our formal verification approach.

An emergency response system is a mission-critical system for which failures in achieving its main goals by the time they are required may lead to catastrophic consequences on users, i.e., on patients monitored by the system expecting to be promptly assisted in case of a medical emergency. Accordingly, any stakeholder that wishes to offer a service based on this system will have both ethical and contractual obligations regarding the safety of its product, that is, it must use appropriate means to prevent system failures.

MPERS is expected to have a high availability - as it must be ready to respond to an emergency that may happen at any time - and a high reliability - as an incorrect emergency response may lead to death or to costly false-positives. Integrity is a less critical attribute in this case, but must also be addressed as patient privacy may not be violated by disclosing his personal health or geolocation info to unauthorized persons. Maintainability is addressed, among others, by the use of a software development methodology and by the ability to update emergency rules remotely at runtime.

Reliability verification of an Ambient Assisted Living System also based on body-area networks though PCM technique was explored by Fernandes [Fernandes, 2012]. Reliability estimation demonstrates the non-determinism in the verification model that will result in an non-deterministic evaluation result. Moreover, PRISM cost/reward structures could be used for the verification of non-functional metrics such as power consumption. In this work, we limit the analysis to the reliability verification as part of a dependability analysis.

## 2.8 Antlr Language Recognition Tool

ANTLR or Another Tool for Language Recognition is a open source parser generator for reading, processing, executing or translating structured text or binary files. The main purpose is to automatically generate a parser for a custom language defined in a specific grammar language supported by the tool. The parser can then be imported in any version compatible JAVA project to build and walk parsed trees.

As a result, any domain-specific language may be specified and then parsed using JAVA methods that will manipulate primitive attributes and objects according to what each parser rule and lexical term means for that language. In our proposal, ANTLR was successfully used to generate the parser for the regular expression language that specifies the behaviour of a runtime goal model (RGM). Further details of the grammar with both parser rules and lexical terms is given in later section.

# Capítulo 3

# Related Work

## 3.1 Contextual Goal Model

The Contextual Goal Model (CGM) [CGM] proposes the contextualization of required goals, adoptable means (goals/tasks) and contribution links values. The main benefit of this work is to enrich the original goal model with the contextualization of entities and relations affected by context variations and to provide a rationale for context analysis. In contrast, the main problem tackled by the our work is the verification of non-functional attributes such as dependability attributes that needs a more precise and less biased approach instead of the existing contribution analysis that is based on analysts direct evaluation of the forward impact between goals/tasks and softgoals.

In this regard, the CGM provided more realistic and precise contribution analysis contextualized by environment conditions, but did not change the nature of the contribution analysis process. Our work has benefited from the CGM conceptual model and has extended the non-functional GORE analysis with a context-dependent formal verification, i.e., that includes different context effects in the probabilistic model used by the PMC to contextually estimate the values of required non-functional attributes of the system and provide a reliable decision criteria for the selection of concurring alternatives in the goal model given different contexts.

## 3.2 Awareness Requirements

Souza et al. [AwaReq] proposed the Awareness Requirements (AwReq) as a meta-requirement in a goal model, i.e., AwReq specify the success/failure rate and temporal constraints for other requirements in the model, including goals, tasks, domain assumptions and other AwReqs (*-meta-requirement). The objective is to enrich the original goal model and provide constraints for system behaviour and to support self-adaptation, as runtime AwReq violations should be addressed by corrective actions. AwReq are formalized by a temporal logic formula, namely the Object Constraints Logic with Temporal Message (OCLtm).

Despite its contribution to the specification of meta-requirements in goal models, AwReq do not provide an approach to analyse and validate its meta-requirements before system implementation and monitoring. Original GORE contribution analysis could be used to define the impact of a given alternative to some attribute or value composing one or more AwReqs. However, the paper have only mentioned the formalization and monitoring through code instrumentation. In contrast, our approach relies on model based verification of meta-requirements similar to AwReq through probabilistic model checking technique that can be performed at design time and provide alternative design decision criteria and anticipate violations that must be treated before implementation.

## 3.3 Runtime Goal Model

Despite the use of goal models to support the runtime monitoring and adaptation, Dalpiaz et al. argued that these works are 'using design artefacts for purposes they are not meant to, i.e., for reasoning about runtime system behaviour'. As such, they proposed a conceptual distinction between the static goal model, named Design Goal Models (DGM), and the Runtime

Goal Model (RGM) that extend DGM with ‘additional state, behavioural and historical information about the fulfilment of goals’ [RGM].

The main purpose of the RGM approach is to provide the proper specification of behaviour information among system goals. RGM defines a class model, while the Instance Goal Model (IGM) captures instance states of runtime monitored goals that must conform to its class specification. IGM are useful to have an instance representation of the RGM provided by the monitoring of the activities involved in fulfilling system goals. If the monitored IGM violates the RGM, then a corrective action should take place. Our work has benefited from the RGM specification language by using the proposed regular expression to have a behaviour specification and generate the probabilistic model. Instance representation is out of the scope of our proposal.

## 3.4 Dependability Contextual Goal Model

The current work has been preceded by another proposal concerning goal-oriented requirements engineering, dependability analysis and dynamic contexts, namely the Dependability Contextual Goal Model (DCGM) [DCGM]. The contribution was focused on both dependability requirements and estimations based on declarative fuzzy logic rules and a variable context of operation.

In DCGM, a failure classification scheme was used to classify the consequence level and domain of failures in achieving system goals. This process lead to the definition of dependability constraints that must be achieved by the means-end tasks used to fulfil leaf-goals in specific contexts of operation, i.e., to the specification of contextual dependability requirements. These requirements inherited the same concept of the AwReq, but instead of being static, they could be associated to a context condition. Another facet of the DCGM is the contextual failure implication, which consisted of a dependability specific GORE contribution analysis supported by fuzzy logic to define IF-THEN rules between context conditions and the level of a dependability attribute, e.g., availability and reliability.

The main drawback of this proposal was the lack of scalability, as declarative rules must be provided for different goals, attributes and contexts, proving to be a time-consuming task for the analysts. A second problem was the subjectivity of the rules, as they were based in domain knowledge that was also used to shape membership functions. This problem, as much as in GORE contribution analysis, lead to the idea of coupling a more precise and reliable verification approach such as the PMC technique. Still, the idea of a failure classification and the specification of contextual non-functional requirements were kept.

## 3.5 Formal TROPOS

The idea behind the formalization of a goal model, as proposed by Formal TROPOS [FTROPOS], is to provide a verifiable specification of sufficient and necessary conditions to create and achieve intentional elements like goals, tasks and dependencies in the model and invariants for each of these elements. In addition to this, new *prior-to* links describe the temporal order of intentional elements. Also, cardinality constraints may be added to any link in the model. Finally, Formal TROPOS uses a first-order linear-time temporal logic as a specification language.

The nature of the verification proposed by Formal TROPOS is different from the PMC used by our work. Formal TROPOS aims to provide the information required for a consistency verification of the goal model. The verification is not only for the abstract TROPOS syntax of intentional elements and relations, but also to domain specific information of how each element is created and fulfilled in time. Once the model starts to have more elements and relations, its consistency checking becomes non-trivial, justifying the use of a formal specification that can be verified by a model checker tool.

In contrast to Formal TROPOS, our proposal uses a probabilistic model checking technique for the verification of properties that depend on how activities in the model are organized in terms of time, cardinality, combination, non-determinism and how each activity individually contributes to the property being evaluated. For instance, if power consumption is to be checked, each activity has to be associated to a power consumption unit and the global consumption value is evaluated considering any non-determinism specified in the execution workflow. Thus, even if the Formal TROPOS language also provides behaviour specification in a goal model, it is tailored for consistency checking of requirements and not for the probabilistic verification of dependability and other non-functional requirements.

# Capítulo 4

# Modelling the Problem

## 4.1 Motivation

Goal-oriented requirements engineering (GORE) has gained the attention of both academic and industrial practitioners due to its ability to systematically model the intentionality behind system requirements. More than just presenting the 'what' and the 'how', goal models also express the 'why' of different requirements to exist. Its simple graphical notation allows non-technical stakeholders to take part in the analysis process and have a clear view of the system-to-be. Finally, automated model verification should avoid violations of the requirements specification.

TROPOS is a GORE methodology that also includes architectural and detailed design phases for the development of socio-technical, multi-agent systems. Socio-technical systems provide and control a wide range of daily used services. Often, these systems are responsible for important and even critical requirements whose failures would cause undesirable or intolerable consequences. This requires developers to take dependability into consideration as a first class requirement.

In TROPOS, as in other GORE frameworks, there is no coupling to any specific verification approach for dependability attributes and other non-functional requirements. Contribution analysis is used for the comparison and selection of alternative design solutions based on how each alternative contribute to one or more system goals, usually qualitative softgoals. This approach, however, is not tailored for metrics depending on system behaviour or more complex analysis techniques, i.e., it is not appropriate for dependability analysis.

Raian et al. proposed a context analysis and notation to the TROPOS goal modelling [CGM]. In a previous work, we tackled the effect of a variable context of operation to dependability attributes using fuzzy logic [SEAMS'14]. We considered that a more scalable and precise approach for the verification of critical NFRs such as dependability attributes affected by context variation was needed. Probabilistic model checking emerged as a potential option to cope the TROPOS methodology with a formal method able to estimate and verify different non-functional metrics in a goal model.

In dependability analysis, probabilistic fault-forecasting aims to derive probabilistic estimates about measures related to the behaviour of a system in the presence of faults. Dependability benchmark enables the characterization of the dependability and security of a system and the comparison of alternative solutions according to one or several attributes [AVIZIENIS 37]. This benchmark may be achieved by a probabilistic model checking technique coupled to the TROPOS methodology, enabling the dependability benchmark of a goal model system representation with additional behaviour specification. To the best of our knowledge, PMC have not yet being used for NFR verification of contextual goal models.

## 4.2 Requirements

Based on the identified gap of a probabilistic verification of non-functional requirements in GORE methodologies, we defined the following requirements that must be addressed by our proposal:

R.1 **Backward compatibility:** Extended TROPOS runtime regex and contextual notation added to the goal model must not conflict to the existing syntax and semantic of the original TROPOS methodology.

R.2 **Optionality:** The use of the probabilistic model checking as a formal verification approach of runtime goal models as an extended TROPOS phase should be optional and not mandatory.

R.3 **Verification scope:** The verification scope may be restricted to a part of the system or encompass the whole system, according to analysts decision.

R.4 **Model generation:** The probabilistic model representing the activities of runtime goal models should be automatically generated from a runtime goal model created in TROPOS RE phases.

R.5 **Tool integration:** The TAOM4E tool/plugin used for TROPOS modelling and analysis activities should be extended with the runtime regex, context notations and verification model generation.

R.6 **Static syntax support:** The verification model should be coherent to the AND/OR decomposition of goals/tasks and to the goal-task means-end relation of the original/static goal model syntax.

R.7 **Dynamic syntax support:** The verification model should be coherent to the goal/tasks achievement/execution order, to cardinality and to alternative, optional and conditional achievement/execution syntaxes in a runtime goal model.

R.8 **Contextual syntax support:** The verification model should be coherent to the context effects over the activation of goals, the adoptability of sub-goals/tasks and over the individual quality metric of leaf-tasks selected for analysis.

# Capítulo 5

# TROPOS Methodology with Probabilistic Requirements Verification

This chapter describes the extended TROPOS methodology applied to the MPERS case study. Both TROPOS early and late requirements analysis phases are fully presented, as well as the verification process that requires a behaviour specification and the additional contextual notation regarding context effects. Further details about the TROPOS methodology may be found in the reference literature [TROPOS].

## 5.1 TROPOS Requirements Engineering Phases

### 5.1.1 TROPOS Early Requirements Phase

In Early Requirements phase, stakeholders are modelled as as well as their needs. Each actor may be a depender or a dependee of a goal, task or resource dependency. In this phase, only the main system actor and the application domain stakeholders are analysed, leaving the detailed system analysis to later development phases.

The MPERS sytem and its social dependencies are presented by the actor model in Figure 5.1. System actors and social actors are displayed in different colors. Among the stakeholders, the emergency center represents a private or public organization interested in providing an emergency response service to patients. Patient and doctor represent, respectively, the assisted person and the medical responsible for defining and evolving the emergency detection rules as part of an evolutionary approach for personal emergency response. Finally, sensors retailer should provide the vital signs sensors required for monitoring.

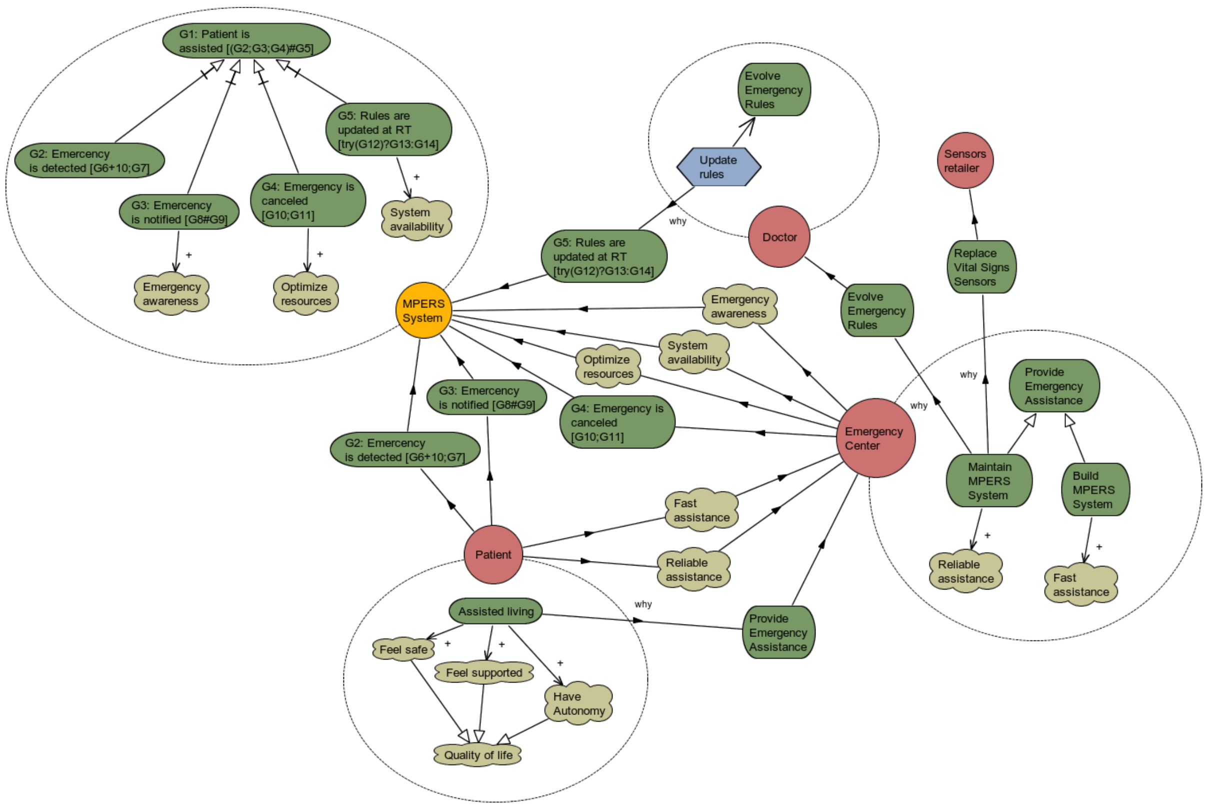


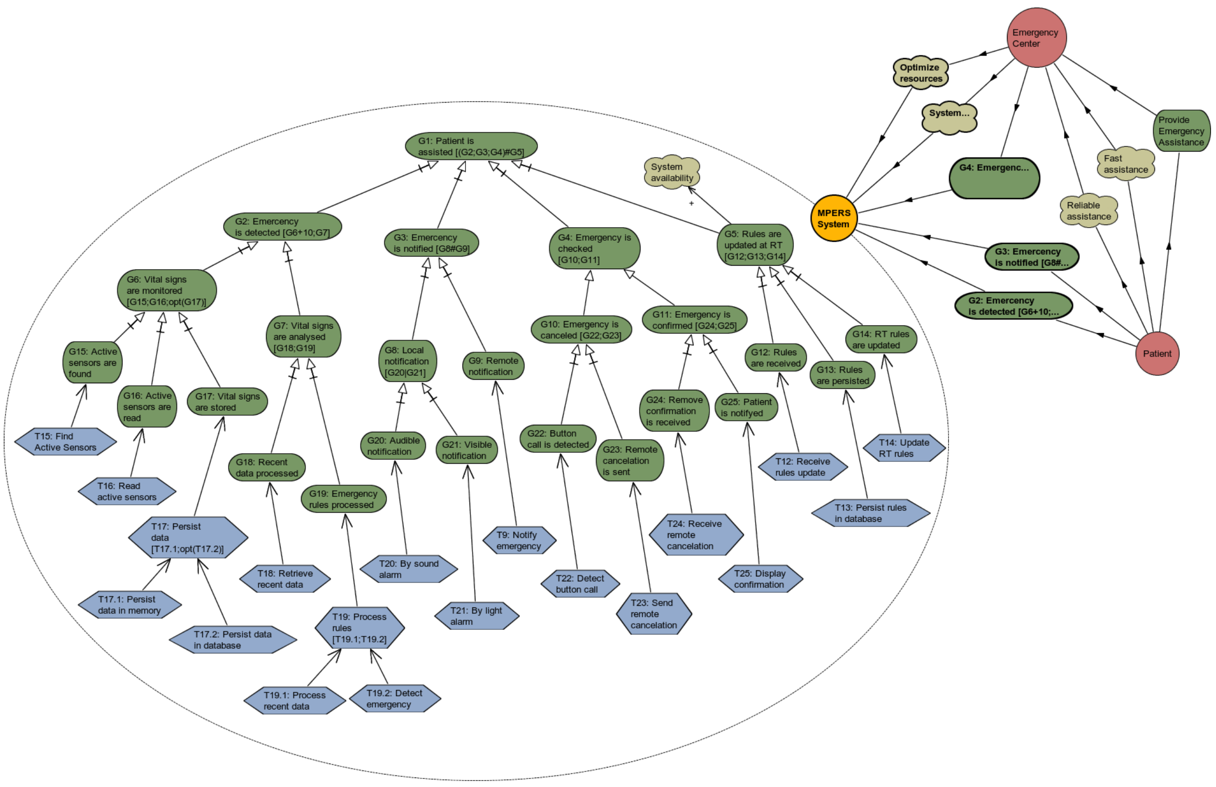
Figura 5.1: MPERS at TROPOS early requirements phase

From the diagram in Figure 5.1 it is possible to have a first view of the MPERS system-to-be. Main goals are divided in detecting, notifying and checking an emergency. Also, the ability to update the emergency rules at runtime (RT) is the fourth and last mandatory goal (AND-decomposition) that fulfils the 'Patient is assisted' root goal. System goals are directly or indirectly related to stakeholders functional and non-functional needs.

The yellow circle indicates that MPERS is a system actor. MPERS goals can be seen with a regex indicating its dynamic behaviour as part of the runtime goal model specification required by the proposal. This notation is a reflex of the late requirements phase, as the TAOM4E tool supporting TROPOS methodology shares unique entities and relations among different development phases. The regex syntax is enclosed by brackets to differentiate then from goal name. In future work, a specific modelling compartment should receive the values for the runtime regex.

### 5.1.2 TROPOS Late Requirements Phase

Later requirements phase concentrates the analysis in the system-to-be and its operational environment. The MPERS goal model occupies the most part of the diagram and each of its main goals are further decomposed through AND/OR decomposition. Also, means-end tasks defines how leaf-goals are fulfilled and the runtime regex across goals and tasks specifies dynamic properties of the system-to-be behaviour. Figure 5.2 illustrates the late requirements diagram for the MPERS.



Emergency Center
Optimize resources
System...
Provide Emergency Assistance
G4: Emergenc...
Fast assistance
Reliable assistance
MPERS System
G3: Emercency is notified [G8#...
G2: Emercency is detected [G6+10;...
Patient
System availability
+
G1: Patient is assisted [(G2;G3;G4)#G5]
G2: Emercency is detected [G6+10;G7]
G3: Emercency is notified [G8#G9]
G4: Emergency is checked [G10;G11]
G5: Rules are updated at RT [G12;G13;G14]
G6: Vital signs are monitored [G15;G16;opt(G17)]
G7: Vital signs are analysed [G18;G19]
G8: Local notification [G20|G21]
G9: Remote notification
G10: Emergency is canceled [G22;G23]
G11: Emergency is confirmed [G24;G25]
G12: Rules are received
G13: Rules are persisted
G14: RT rules are updated
G15: Active sensors are found
G16: Active sensors are read
G17: Vital signs are stored
G18: Recent data processed
G19: Emergency rules processed
G20: Audible notification
G21: Visible notification
G22: Button call is detected
G23: Remote cancelation is sent
G24: Remove confirmation is received
G25: Patient is notifyed
T15: Find Active Sensors
T16: Read active sensors
T17: Persist data [T17.1;opt(T17.2)]
T17.1: Persist data in memory
T17.2: Persist data in database
T18: Retrieve recent data
T19: Process rules [T19.1;T19.2]
T19.1: Process recent data
T19.2: Detect emergency
T20: By sound alarm
T21: By light alarm
T9: Notify emergency
T22: Detect button call
T23: Send remote cancelation
T24: Receive remote cancelation
T25: Display confirmation
T12: Receive rules update
T13: Persist rules in database
T14: Update RT rules

Figura 5.2: MPERS at TROPOS early requirements phase

At this stage of the methodology, the system is represented as a monolithic actor and its goal model may be extended with the runtime specification. This extended model merges multiple views in the same diagram: goals and tasks represent the requirements view for the system-to-be as well as the intentionality behind then, while the runtime specification provides a dynamic representation in terms of goal achievement and task execution.

To evaluate our proposal, we have first explored the use of the PMC technique for the verification of dependability attributes in the system model of the TROPOS late requirements phase. The idea is to initially evaluate the approach in a monolithic representation without the additional complexity of a multi-agent architecture. The evaluation involving later TROPOS phases should be explored in future work. The remaining of this section will focus on the extended verification phase proposed by this work.

## 5.2 TROPOS Extended Verification Phase

In this section, we present our proposed TROPOS extension for verification of NFRs. First, non-functional constraints are associated to the MPERS goal model. Then, the runtime regex is further detailed and compared to a UML activity diagram. After this, the high-level DTMC model is built and details about the conversion from a RGM to a DTMC are provided. Finally, we demonstrate how to estimate reliability of the RGM based on a high-level DTMC model composed of leaf-tasks selected for analysis.

Figure 5.3 illustrates the extended TROPOS methodology that starts with original phases involving goal-oriented modelling and ends with the generation of a high-level DTMC model used for the verification of NFRs.

### 5.2.1 Non-functional requirements specification

TROPOS goal model also provides rationale for NFR analysis, as it originally inherited the softgoal analysis from the NFR framework [NFR]. In our

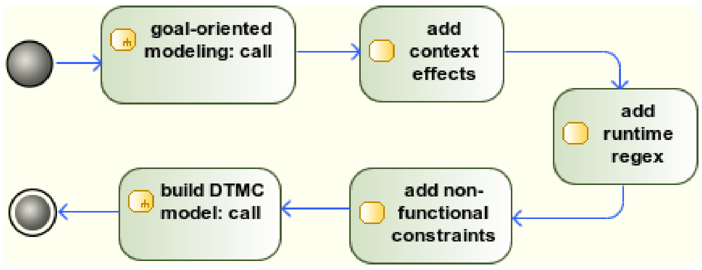


Figura 5.3: Building a DTMC model from a contextual/runtime goal model.

approach, non-functional constraints are modelled as qualitative hard goals with a clear cut value for its satisfaction, complementing the other NFRs modelled as softgoals. As a benefit of a goal-oriented modelling for NFRs, the elicitation of a given NFR may be justified by its relation to other elements in the model. Figure 5.4 presents some NFRs for the MPERS system actor.

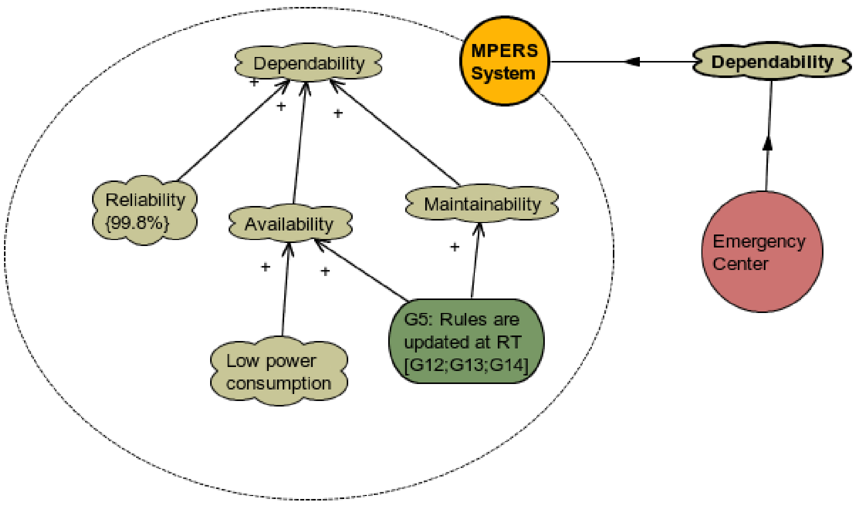


Figura 5.4: MPERS non-functional requirements.

Similarly to the Awareness Requirements by Souza et al. [AWREQ], some NFR define metrics over other requirements. These meta-requirements are not directly fulfilled by system functionalities like 'emergency awareness' is fulfilled by 'notify emergency' or 'confidentiality' could be fulfilled by 'user authentication', but by how these functionalities will perform. Reliability, for instance, is inversely proportional to the likelihood of failures. Hence, the reliability metric depends on the probability of system functionalities to fail.

Conformity to these metrics is not implicit in the model, it requires some verification technique.

Each requirement in a goal model must come from another requirement through decomposition, means-end or contribution links, or it must be directly mapped to stakeholder needs through dependency links. Emergency center attended the patient's needs by providing and maintaining the MPERS system itself and by assuring other NFR for the system. Reliability was selected as metric over the system execution (meta-requirement), while availability and maintainability are partially satisfied by the 'low power consumption' softgoal and MPERS ability to update emergency rules at runtime, respectively. This evaluation will focus on the reliability metric and, for the sake of simplicity, will omit other NFRs like availability as metric.

### Specifying non-functional constraints

The specification of non-functional constraints is a sensible task that depends on expertise and on domain knowledge. For instance, an analyst or a reliability engineer should be aware of what does it mean for a system to be 99.99% reliable, as this level may not be achieved by any alternative solution and must be coherent to the system criticality - a catastrophic failure should be avoided by all means. In some cases, the system will have to comply to some industry standards or contract based constraints. Table 5.1 summarizes two possible non-functional constraints for the MPERS.

| NFR | Constraint | Target |
|---|---|---|
| **Reliability** | 99.8% | **G1** |
| **Power consumption** | 100 p.u. | **G2** |

Tabela 5.1: Non-functional metrics for the MPERS system.

As indicated by the *Target* column, each NFR constraint may be associated to a root level goal or to any of its subgoals. The corresponding probabilistic verification based on the execution of a set of leaf-tasks in the RGM is defined as:

- *Global*, if the activities set is a minimum set composed of the tasks that satisfies the chain of subgoals up to the root goal Groot. For instance, in Table 5.1 reliability is associated to root goal G1.
- *Local*, if the activities set is a minimum set composed of tasks that satisfies the chain of subgoals up to a goal Gx, where Gx != Groot. For instance, in Table 5.1, power consumption is (locally) associated to goal G2.

Other MPERS NFR are 'emergency awareness' and 'resources optimization'. These softgoals are addressed by system functionalities. The former receives a full contribution (double positive sign) from goal 'emergency is notified', meaning it is fully satisfied by this goal. The later is just assumed to be partially satisfied by the 'emergency is checked' functional goal.

### 5.2.2 Behaviour specification

In this section, further details about the runtime regex syntax and semantic will be explained as they are a central part of this proposal. Table 5.2 provides a textual description of each RGM notation with corresponding meaning in terms of what behaviour it specifies and also an example from the MPERS RGM. A formal and detailed description can be found in the RGM reference publication [RGM].

A small variation of the original regex was employed for the E+ and E# rules. Instead of an undetermined number of goal/task instances, analyst should provide the exact number of instances for goals achievement and tasks execution. This information is required for the generation of the verification model from a RGM.

| **Expression** | **Meaning** | **Example (MPERS)** |
|---|---|---|
| skip | No action. Useful for conditional ternary expressions involving two elements. | try(G10)?skip:G11 |
| E1;E2 | A goal/task E1 must be fulfilled/executed before E2. | G1;G2;G3 |
| E1\|E2 | Fulfillment/execution of goal/task E1 is alternative with respect to E2. | T9.1\|T9.2 |
| opt(E) | Fulfillment/execution of goal/task E is not mandatory. | opt(T17.2) |
| E+n | Goal/task E must be fulfilled/executed n times, with n >0. | G22+2 |
| try(E)?E1:E2 | If goal/task E succeeds, E1 must be fulfilled/executed; otherwise, E2. | try(G10)?skip:G11 |
| E1#E2 | Interleaved fulfillment/execution of goal/task E1 and E2. | G8#G9 |
| E#n | Interleaved fulfillment/execution of n instances of E, with n >0. | - |

Tabela 5.2: Description of RGM textual notations used by the proposal.

### RGM - UML activity diagram comparison

A similar specification could be provided by an UML activity diagram with activities as the leaf-tasks of the goal model. However, activity diagrams have an homogeneous abstraction level and do not clearly correlate behaviour to the requirements they are meant to satisfy. In contrast to the RGM, activity diagrams denote behaviour through graphical symbols, while the RGM mixes the original goal model notation with a runtime regex. This simple notation increases the utility of a goal model diagram. Figure 5.5 presents an activity diagram corresponding to the MPERS RGM.

Among its limitations, RGM does not express that an emergency has to be confirmed after a time or signal event as the UML activity diagram does. Both necessary and sufficient conditions for the triggering and fulfilment of goals, tasks and dependencies are provided by Formal TROPOS

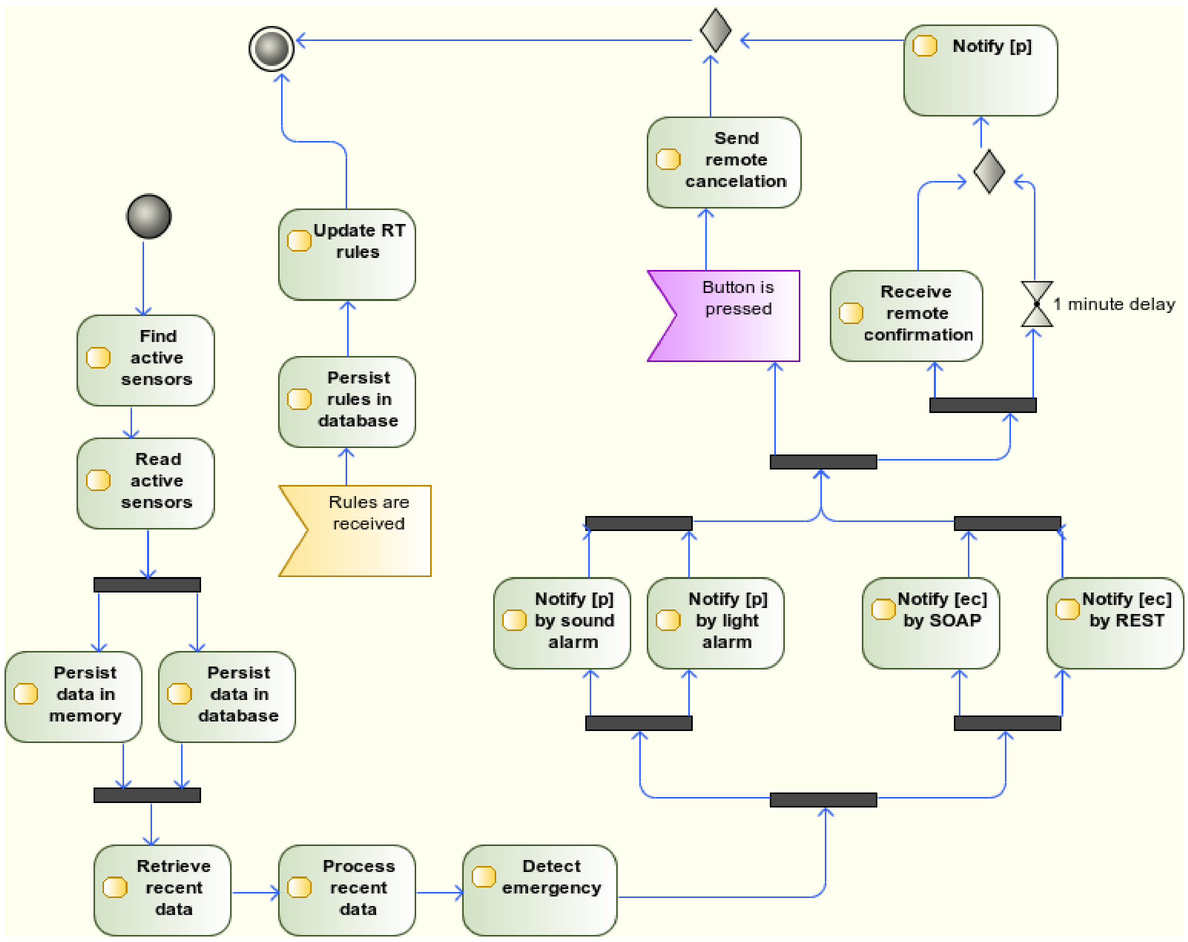


Figura 5.5: MPERS tasks represented by a UML activity diagram

language. Still, sequential, parallel, alternative, optional and conditional execution flows as long as multiple executions of the same task can be expressed by the RGM, providing a rich behaviour specification that could be checked for non-functional requirements such as dependability attributes.

The idea of a runtime goal model is not to replace UML activity diagrams, but to complement the static goal model with a clear runtime syntax that could be used for communication and for conformance verification at both design time and during system execution - as the execution monitoring originally proposed by Dalpiaz et al [RGM]. Depending on the complexity of the behaviour specification, a more robust runtime syntax would have to be employed or complemented by UML behaviour diagrams.

## 5.2.3 Non-functional requirements verification

This section describes the application of a PMC technique to verify the conformance of the RGM to defined non-functional constraints and also to solve the variability problem at design time considering both cases explained in Section 2.4, i.e., for static context and dynamic contexts.

#### Leaf-tasks as DTMC modules

The state-based verification of non-functional constraints that specifies how different functionalities should perform is a complex task that involves a representation of system states and their transitions. A goal model may define system requirements with variable abstraction levels. As such, the feasibility of the verification of a given metric depends on the information provided by the model.

The MPERS RGM in Figure 5.2 expands its main goals in further sub-goals that are ultimately satisfied by operational tasks. Tasks can also be expanded in more granular subtasks. Tasks without outgoing relations are named leaf-tasks. In our proposal, leaf-tasks are mapped to modules in a DTMC model in PRISM language. PRISM modules are containers for variables and commands, i.e., for states and behaviour. Figure 5.6 presents a system task as a PRISM module.

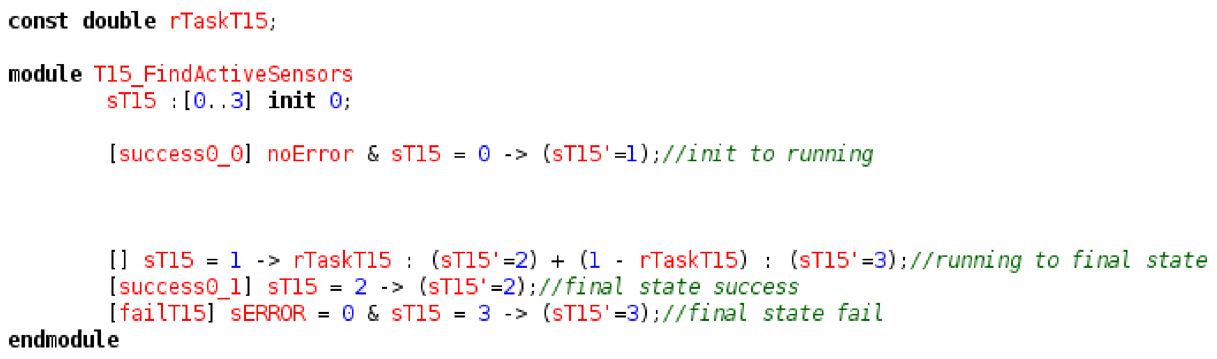

```
const double rTaskT15;

module T15_FindActiveSensors
        sT15 :[0..3] init 0;

        [success0_0] noError & sT15 = 0 -> (sT15'=1);//init to running


        [] sT15 = 1 -> rTaskT15 : (sT15'=2) + (1 - rTaskT15) : (sT15'=3);//running to final state
        [success0_1] sT15 = 2 -> (sT15'=2);//final state success
        [failT15] sERROR = 0 & sT15 = 3 -> (sT15'=3);//final state fail
endmodule
```

Figura 5.6: A PRISM DTMC module representing T15 (mandatory) task.

In the DTMC model, leaf-tasks have their execution state mapped to a variable in the model (sT15 variable in Figure 5.6). From the RGM original set of goal/tasks instance states, we considered the following values:

- Init(sTask=0): corresponds to the initial state of a given leaf-task. From this state, a transition may occur to the running state, if this task is part of a system alternative that will be analysed, or directly to the final success state, if the opposite.

- Running(sTask=1): corresponds to the operational state of a given leaf-task. From this state, a transition to the final success or failure states may occur. A variable ranging from 0 - 1 defines the task's reliability, i.e., the probability of a transition to the success state - and the complementary transition probability to the failure state (variable rTask15 in Figure 5.6).

- Success(sTask=2): corresponds to the absorbing final state of success of a singular task execution.

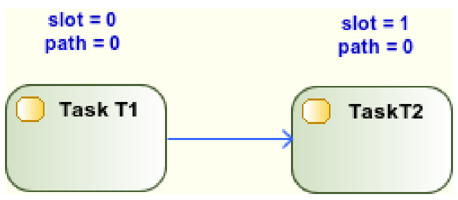
- Failure(sTask=3): the opposite from the final success state.

#### Building the high-level DTMC model from a RGM

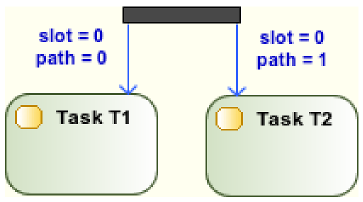
Given a set of leaf-tasks that fulfils a chain of subgoals until a certain goal G, a DTMC model composed of modules for each leaf-task states and transitions is build. This model must represent the same workflow of the corresponding RGM, i.e., it must preserve the behaviour semantic specified by the runtime regex.

In this work, we are not interested in checking system instance conformance to its runtime goal model through monitoring as the original RGM proposal. We focus on the estimation of non-functional metrics based on the high-level DTMC model. We call it a high-level DTMC model because leaf-tasks are similar to activities in a UML diagram whose behaviour could be further detailed, e.g., by sequence diagrams.

Each leaf-task in the DTMC model starts at a discrete time slot. Time slots defines the sequence order of tasks executions. Sequential tasks (T1;T2) have subsequent time slots, meaning that T2's initial transition is synchronized to T1's final transition. Interleaved tasks (T1#T2) have their initial state transition synchronized at the same time slot through labels, but occupy different time paths, i.e., following state transitions are interleaved. Figure 5.7a and 5.7b present sequence and interleaved execution cases in UML notation.



(a) Sequential tasks T1 and T2 are in different time slots.



(b) Interleaved tasks T1 and T2 are in different time paths.

Other supported behaviours are: alternative execution (E1|E2), optional execution (opt(E1)) and conditional execution (try(E)?E1:E2). For alternative execution, or XOR, a parameter in the model defines which alternative from a set of two or more is selected for analysis. Figures 5.8 and 5.9 present the UML activities and corresponding PRISM model for alternative tasks T9.00 and T9.01.

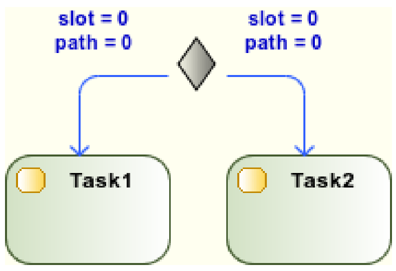


Figura 5.8: Alternative tasks T9.00 and T9.01 can be selected for analysis.

```
const double rTaskT9_00;
const int XOR_T9_00_T9_01;

module T9_00_NotifyBySOAP
        sT9_00 :[0..3] init 0;

        [success0_7] noError & sT9_00 = 0 -> (sT9_00'=1);//init to running


        [success0_7] noError & XOR_T9_00_T9_01=0 & sT9_00 = 0 -> (sT9_00'=1);//init to running
        [success0_7] noError & XOR_T9_00_T9_01!=0 -> (sT9_00'=2);//not used, skip running



        [] sT9_00 = 1 -> rTaskT9_00 : (sT9_00'=2) + (1 - rTaskT9_00) : (sT9_00'=3);//running to final state
        [success2_8] sT9_00 = 2 -> (sT9_00'=2);//final state success
        [failT9_00] sERROR = 0 & sT9_00 = 3 -> (sT9_00'=3);//final state fail
endmodule

const double rTaskT9_01;

module T9_01_NotifyByREST
        sT9_01 :[0..3] init 0;

        [success0_7] noError & XOR_T9_00_T9_01=1 & sT9_01 = 0 -> (sT9_01'=1);//init to running
        [success0_7] noError & XOR_T9_00_T9_01!=1 -> (sT9_01'=2);//not used, skip running



        [] sT9_01 = 1 -> rTaskT9_01 : (sT9_01'=2) + (1 - rTaskT9_01) : (sT9_01'=3);//running to final state
        [success2_8] sT9_01 = 2 -> (sT9_01'=2);//final state success
        [failT9_01] sERROR = 0 & sT9_01 = 3 -> (sT9_01'=3);//final state fail
endmodule
```

Figura 5.9: Alternative tasks T9.00 and T9.01 as DTMC modules with additional integer parameter used for selection.

Similarly to alternative execution, optional tasks are enabled by an additional parameter in the model. Figures ?? and 5.10 illustrate the UML activity and corresponding PRISM model for optional task T17.2.

```
const double rGoalT17_2 = 0.999;
const bool OPT_T17_2;

module T17_2_PersistInDatabase
        sT17_2 :[0..3] init 0;

        [success0_3] noError & OPT_T17_2 & sT17_2 = 0 -> (sT17_2'=1);//init to running
        [success0_3] noError & !OPT_T17_2 -> (sT17_2'=2);//not used, skip running

        [] sT17_2 = 1 -> rGoalT17_2 : (sT17_2'=2) + (1 - rGoalT17_2) : (sT17_2'=3);//running to final state
        [success0_4] sT17_2 = 2 -> (sT17_2'=2);//final state success
        [failT17_2] sERROR = 0 & sT17_2 = 3 -> (sT17_2'=3);//final state fail
endmodule
```

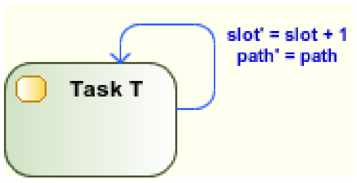
Figura 5.10: An optional task T17.2 as a DTMC module with additional boolean parameter.

Finally, labels are used to condition the execution of tasks to the success and failure of a third task. Figures ?? and 5.11 present the UML activities and corresponding PRISM model for conditional tasks T9.00 and T9.1. In the case of task T9.00, it is, at the same time, alternative to task T9.01 (Figure 5.9) and a failure condition to task T9.1, i.e., task T9.1 will be executed if task T9.00 fails.

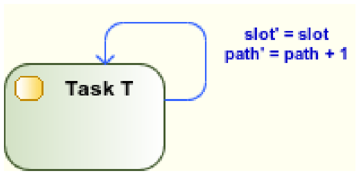
```
const double rTaskT9_00;

module T9_00_NotifyBySOAP
        sT9_00 :[0..3] init 0;

        [success0_7] noError & sT9_00 = 0 -> (sT9_00'=1);//init to running


        [success0_7] noError & XOR_T9_00_T9_01=0 & sT9_00 = 0 -> (sT9_00'=1);//init to running
        [success0_7] noError & XOR_T9_00_T9_01!=0 -> (sT9_00'=2);//not used, skip running


        [] sT9_00 = 1 -> rTaskT9_00 : (sT9_00'=2) + (1 - rTaskT9_00) : (sT9_00'=3);//running to final state
        [success2_8] sT9_00 = 2 -> (sT9_00'=2);//final state success
        [failT9_00] sERROR = 0 & sT9_00 = 3 -> (sT9_00'=3);//final state fail
endmodule

const double rTaskT9_1;

module T9_1_NotifyBySMS
        sT9_1 :[0..3] init 0;

        [failT9_00] noError & sT9_1 = 0 -> (sT9_1'=1);//init to running
        [success2_8] noError & sT9_1 = 0 -> (sT9_1'=2);//not used, skip running


        [] sT9_1 = 1 -> rTaskT9_1 : (sT9_1'=2) + (1 - rTaskT9_1) : (sT9_1'=3);//running to final state
        [success2_9] sT9_1 = 2 -> (sT9_1'=2);//final state success
        [failT9_1] sERROR = 0 & sT9_1 = 3 -> (sT9_1'=3);//final state fail
endmodule
```

Figura 5.11: Conditional tasks T9.00 and T9.1 as DTMC modules.

Cardinality is also supported by the runtime regex specification. In our proposal, execution cardinality is simply represented by n modules with subsequent time slots (E+n) or by modules with synchronized initial transitions and followed by interleaved transitions (E#n). Figures 5.12a and 5.12b respectively illustrate sequential and interleaved task behaviours.



(a) Sequential tasks T1 and T2 are in different time slots.



(b) Interleaved tasks T1 and T2 are in different time paths.

In Table 5.3, the final settings for the time slot, path and other dynamic aspects of leaf-tasks parsed from the RGM and used for the generation of a DTMC model are presents.

| **Task** | **Time path** | **Time slot** | **Optional** | **Conditional** | **Alternative** | Cardinality |
|---|---|---|---|---|---|---|
| **T15** | 0 | 0 | false | false | false | 1 |
| **T16** | 0 | 1 | false | false | false | 1 |
| **T17.1** | 0 | 2 | false | false | false | 1 |
| **T17.2** | 0 | 3 | true | false | false | 1 |
| **T18** | 0 | 4 | false | false | false | 1 |
| **T19.1** | 0 | 5 | false | false | false | 1 |
| **T19.2** | 0 | 6 | false | false | false | 1 |
| **T20** | 0 | 7 | false | false | false | 1 |
| **T21** | 1 | 7 | false | false | false | 1 |
| **T9.1** | 2 | 7 | false | false | T9.2 | 1 |
| **T9.2** | 2 | 7 | false | false | T9.1 | 1 |
| **T22** | 0 | 8 | false | false | T24 | 2 |
| **T23** | 0 | 9 | false | false | T24 | 1 |
| **T24** | 3 | 8 | false | false | T22 | 1 |
| **T25** | 3 | 9 | false | false | T22 | 1 |
| **T13.1** | 4 | 0 | false | false | false | 1 |
| **T13.2** | 4 | 0 | false | false | false | 1 |
| **T13.3** | 4 | 0 | false | false | false | 1 |

Tabela 5.3: Dynamic aspects of the MPERS leaf-tasks in the DTMC model.

### 5.2.4 Gathering individual task metrics

Leaf-tasks are not necessarily atomic system operations and are generally described with a high abstraction level. For instance, the MPERS task 'Find active sensors' could be further decomposed in more granular tasks. As presented in Figure 5.2, MPERS leaf-tasks are not coupled to any platform, architecture and language used for implementation.

The more abstract a task is, the more difficult is to obtain their individual metrics, as the trace between an abstract and the concrete system operation becomes less evident. Each NFR verified by the PMC technique requires corresponding information about individual parts involved in the overall activity, e.g., the reliability, performance, power consumption, cost and other metrics for tasks in a workflow.

This is a key point in this approach, as it may seen too loose to couple a probabilistic verification to a goal model with a high-level operational representation of a system. But its feasibility becomes more clear if the metrics being verified are compatible with the abstraction level of the leaf-tasks in a specific goal model and individual task metrics are available or can be collected.

#### Pure model-based approach

Depending on the purpose of the goal modelling - from early requirements elicitation to detailed design in TROPOS - more granular specification of tasks behaviour can be provided by decomposition and auxiliary UML diagrams. This internal task behaviour specification can be used for individual task analysis of metrics such as reliability and power consumption.

#### Hybrid monitoring approach

In contrast to a pure model-based verification approach, tasks execution instances could be monitored using the original RMG conformance algorithms. This hybrid verification requires a concrete system implementation besides the RGM system model.

### 5.2.5 Reasoning with Ex-Tropos

The probabilistic verification of NFR, performed as part of the Validation & Verification (VV) phase in RE, should anticipate (contextual) violations of non-functional requirements. Treating a detected violation at design time may correspond to actions such as making a different choice for underlying components used by this alternative's tasks, optimizing its behaviour specification or even the disposal of this alternative as a means to satisfy its goal if there is at least one other valid alternative.

PMC technique also allows the identification of system alternatives with more influence on each metric through sensitive analysis.

#### Variability in GORE

The variability in goal models leads to more than one minimum set of tasks capable of fulfilling local or root goals. In OR-decompositions, at least one alternative is required and the maximum number of combinations is defined by $1 + 2^{(n-1)}$, with n the number of OR-decomposed goals/tasks. Therefore, the verification of all alternatives in a goal model with individual models may prove to be inefficient or infeasible if too many variation points exist in the model.

The PMC approach has already been explored for the verification of models with variability. Rodrigues et al. proposed a family-based verification of software product lines (SPL) [RODRIGUES]. The main idea is to reduce the analysis effort and boost the feasibility of the SPL verification. In the proposed family-based verification, parameters in a PRISM probabilistic model generates a parametric formula for all products in the SPL for a given PCTL property.

In our proposal, PCM verification follows the same principle of the SPL family-based verification. A parametric DTMC model should be generated from the RGM. Alternatives are selected by passing values to parameters. For instance, if both GPS and triangulation are available means for identifying the patient location, a parameter with values 0 or 1 will indicate which alternative is enabled for verification.

A family-based PMC is useful for comparing each alternative with non-functional metrics as criteria to decide which one should be used by the system-to-be at design time or by the real system at runtime. At design time, this approach is analogue to the TROPOS contribution analysis. In both cases, a unique parametric formula evaluates a PCTL property corresponding to a non-functional metric.

#### Context selection

As in the contextual goal model, contexts may limit which alternatives are adoptable. This effect must be considered in a realistic verification. As a novelty, our approach for the verification of non-functional metrics through PMC will also include variable contexts of operation and their effects in the verification model. Two different approaches for context selection may be employed:

- *Deterministic context selection, or DCS*: one context is selected by the analyst before the verification. Context effects in the verification model should be activated and cause the evaluation result to correspond to the selected context.

- *Probabilistic context selection, or PCS*: a probability distribution will define the likelihood of a context to be selected and the corresponding context implications in the verification model to be activated.

Both approaches are complementary as the first verifies the selected alternative for one context at a time and the last verifies a realistic scenario with multiple possible contexts.

### Context-alternative selection

Table ?? summarizes each verification approach and possible combinations.

| | | DAS | PAS |
|---|---|---|---|
| | | A single alternative is selected by the analyst. | Alternative selection follows a probabilistic distribution. |
| **DCS** | An unique context is selected by the analyst. | Alternative selection by the analyst is limited by the selected context. | Probabilistic alternative selection is limited by the selected context. |
| **PCS** | Context selection follows a probabilistic distribution. | Alternative selection by the analyst may fail according to the probability of selecting an incomplatible context. | Probabilistic alternative selection may fail according to the probability of selecting an incomplatible context. |

Tabela 5.4: Description of the different approaches for verifying a system with variable alternatives and variable contexts.

If the context selection is deterministic (DCS), there is no reason for verifying an alternative that is known to be incompatible with the selected context. Therefore, only adoptable alternatives must be verified. In opposition, if context selection is probabilistic, that alternative may still be valid in other contexts, hence it must be included in the multi-context evaluation. For instance, the patient location identification through GPS (alternative) will certainly fail if the GPS signal is not available (context). Thus, the DAS-DCS combination checks one compatible context-alternative pair at a time, while DAS-PCS combination leads to the verification of multiple context-alternative pairs at the same time.

The idea behind a probabilistic context selection is to emulate a realistic scenario in which the context of operation varies and the system must avoid requirements violations by having an adoptable alternative for each context. This holistic evaluation provides measures weighted by the probabilistic context distribution. For instance, if GPS signal is available 70% of the time and triangulation is available 90% of the time and if each method has its own reliability, namely rGPS and rTRI, the reliability of high-level task 'identify patient location' is defined by the expression 0.9*rGPS + 0.1*0.7*rTRI, considering that GPS has priority over triangulation.

#### Discrete-time Markov chain model

#### Reliability verification

Transition probability between running to final success state is described by the individual reliability of the corresponding task, namely rTask.

#### Power consumption verification

### 5.2.6 From NFR to PCTL properties

The estimation of attributes through PMC technique is limited to those that a probabilistic model may evaluate. Dependability attributes have an abstract definition that must be associated to a concrete and verifiable PCTL property. To demonstrate our approach, we verify the MPERS model for the following attributes:

- **Reliability**, represented by the probability of a successful execution of all the activities involved in fulfilling leaf-goals of a certain system alternative. It is also know as the *reachability* as the describes the probability of reaching a final and successful system state.

- **Availability**, represented by the power consumption estimation to maximize the time that the system will remain operational depending only on its battery. This attribute is well related to mobile computing.

each task has its own states, including the failure and the success states. Many factors may contribute to the correctness or the failure of system tasks, including internal and external events. The probability of a successful task execution defines its reliability.

In a complex workflow of tasks with different rel

In order to be successful, tasks depend on the proper interaction among the components

seen as an activity diagram and be used to generate a probabilistic model in PRISM language. This allows the model checking of the corresponding goal model as a set of activities for which temporal and other behaviour aspects are specified by the runtime regex of the RGM.

## 5.3 Treating NFR Violations

- Making a different choice for underlying components: In some cases the replacement of a technical component for another of the same class can improve the quality of how they achieve their goal. For instance,

- Behaviour optimization: The quality may also depend on the pattern used for the activities execution. The specification of a different pattern may eliminate the non-functional violation.

- Contextualizing the alternative: An alternative may only violate a NFR in specific contexts. In this case, different valid alternatives may be used according to the context of operation.

- Alternative disposal: If the alternative is in absolute violation or if its validity is restricted to contexts that have at least one other valid alternative, this branch can be eliminated from the model.

Dependability analysis is used to provide information about different dependability attributes related to system failures. These metrics may be specified as non-functional requirements for isolated system functionalities or for the whole system. Instead of softgoals, we use meta-requirements over functional goals with clear-cut quantitative criteria such as '99.999%' reliable - a probabilistic value to make it compatible with the PMC estimation results.

To perform the , we focus on dependability related metrics that should be estimated and compared to their required constraint values through quantitative analysis. Sensitive analysis to reveal how different system parts contribute to the overall value of those attributes. Sensitive analysis may be considered analogous to the original GORE contribution analysis.

# Capítulo 6

# Automatic PRISM generation

As we wanted to automate the code generating process for the verification model, the graphical modelling environment that supports TROPOS methodology and the code generation for multi-agents was extended to also generate probabilistic models for the PMC technique.

To reduce the effort of codifying the verification model, an automated generation of the PRISM probabilistic model was implemented based on an existing open source tool for TROPOS development support named TAOM4E[citation]. TAOM4E provides a graphical environment for goal modelling with TROPOS methodology based on the well known Eclipse Modelling Framework (EMF) and Graphical Editing Framework (GEF). The GORE to PRISM generator was implemented as a Eclipse plugin and integrated to the TAOM4E environment.

The purpose of the automated code generation for the probabilistic PRISM model is to optimize the formal verification step by abstracting the PRISM language from the analysts and reduce the overhead and time of the model verification. This should increase the feasibility of adopting the extended TROPOS methodology by keeping analysts with their original responsibility of modelling and analysing the system, its social environment and its different contexts of operation.

In terms of a high level system behaviour, each activity has its own states space including success and failure. Our probabilistic verification approach requires not only a formal specification of the system behaviour, but also metrics related to how individual components involved in system activities will perform in respect to the analysed metric. In reliability verification, each component has an individual probability of successfully performing its functional task. Analysts must obtain these values by consulting their manufacturer, by individually analysing each component reliability based on their behaviour specification until the atomic level or by monitoring these components in a testing or production environment. Further details of how individual metrics may be obtained for the PCM may be found at the literature and are out of the scope of this work.

In the PMC technique that has been adapted by this proposal, a behavioural specification, usually provided by UML activity and sequence diagrams, are manually converted to a probabilistic model in PRISM language. This tends to be a costly and error-prone process with a complexity proportional to the number of components, actions and interactions causing state transitions.

As a goal model is traversed from strategical root goal to operational leaf-goals, and each leaf-goal is reachable by a delegation to other actor or by a operational task, then a behaviour specification as proposed by the RGM may have enough information to be consumed as input for the generation of probabilistic models in PRISM language and for the verification of some important NFRs. However, the manual generation from RGM is still a costly task.

Depending on the abstraction level and the nature of the verification, PRISM models may either very complex or may follow a clear pattern. For instance, PRISM modules can be used to represent leaf-tasks of a runtime goal model. Considering a DTMC model, a task workflow can be modelled as a sequence of probabilistic state transitions according to the behavioural specification parsed from the RGM.

This probabilistic model follows a pattern that motivated the implementation of an automatic generation of DTMC models representing leaf-tasks execution directly from a runtime goal model.

# Capítulo 7

# Validation

# Capítulo 8

# Conclusion