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

Coordenadora: Prof.ª Dr.ª Alba Cristina Magalhaes Alves de Melo

Banca examinadora composta por:

Prof. Dr.ª Genaína Nunes Rodrigues (Orientadora) — CIC/UnB
Prof.ª Dr.ª Vander Alves — CIC/UnB
Prof. Dr. Luciano Baresi — Politecnico di Milano

**CIP — Catalogação Internacional na Publicação**

Mendonça, Danilo F..

An Extended Goal-oriented Development Methodology with Contextual Dependability Analysis / Danilo F. Mendonça. Brasília : UnB, 2015.
59 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

Prof. Dr.ª Genaína Nunes Rodrigues (Orientadora)
CIC/UnB

Prof.ª Dr.ª Vander Alves
CIC/UnB

Prof. Dr. Luciano Baresi
Politecnico di Milano

Prof.ª Dr.ª Alba Cristina Magalhaes Alves de Melo
Coordenadora 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

# Abstract

**Keywords:** LaTeX, scientific method

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Among the different causes that lead a system to fail, some can be tracked back to design decisions in early system development process, others are caused by variations in the context of operation. Dynamic contexts increases even further the complexity of the development activities, as different contexts may change what the system should accomplish and the available means to do it. Also, some failures are only activated in specific contexts, posing a new threat to the development of dependable systems.

According to Finkelstein et al. [2], contexts are the reification of the system environment and the environment is whatever over which the system has no control and surrounds its operation - battery status, signals strength, sensors availability, infrastructure restrictions, user characteristics, physical environment conditions, etc. A self-adaptive system should be able to adapt to different context conditions to avoid deviations from its specified behaviour, i.e., to avoid failures.

Dependability defines the system ability of delivering a service that can justifiably be trusted. Different dependability attributes are used to characterize the correct system behaviour, its ability to undergo modifications and the consequence of failures. In dependability analysis, fail forecasting should provide an evaluation of the system behaviour in respect to fault occurrence or activation, while fault removal includes the verification, diagnosis and correction of faults [3].

A systematic requirements engineering process aims to improve the quality the delivered systems. However, not many methodologies and frameworks investigate the conformance of the system-to-be to qualitative goals and metrics related to system failures, nor provide adequate means for the monitoring and analysis of these metrics as part of a self-adaptive loop. We argue that, despite the robustness of self-adaptation architectures, a faulty or biased analysis may result in severe or catastrophic system failures.

## 1.1 Problem Definition

In recent works, goal-oriented requirements engineering (GORE) has been adopted for the development of self-adaptive systems. Among others, goal models are used for deriving a self-adaptive architecture [17] and a high-variability design [19]. Also, goals become live requirements entities that can be self-adapted according to the context [5, 6] or are complemented with a special class of meta-requirements that refers to their success/failures and other metrics that must be satisfied through self-adaptation [16].

Goal models are not restricted to higher-level, 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. Dalpiaz et al. proposed a paradigm shift for the goal modelling and analysis. Instead of a static model for design-time analysis, a regular expression for behaviour specification over goals and tasks composes the runtime goal model (RGM). RGM is suitable for verifying the conformance of the execution and fulfilment of monitored tasks and goals instances to their class specification [8].

Despite its contribution to the runtime conformance verification, detailed measurement of the success and failure rates over temporal frames are not yet supported by the RGM framework. Also, RGM relies on measurements over the past execution like 'the percentage of success for a given goal over the past month' and 'the trend for failure of a given goal in the last week'. Due to this limitation, the original RGM approach is not appropriate for proactive self-adaptation in which systems should avoid violations by detecting or estimating the probability of failures in future executions.

Given the fact that for many systems the ability to avoid failures before they take place is crucial and considering the benefits of a goal-oriented approach for requirements engineering (and for architecture and design, according to [7]), in this work we investigate the feasibility of a formal verification for the goal model representation of a system in order to estimate the probability of different goals in being fulfilled as part of a proactive self-adaptation analysis. Accordingly, the first and more general research question emerges:

> **Research question 1 (RQ1):** Given a correct and consistent goal model for which goals are ultimately satisfied by system tasks, is it possible to analyse the probability of different goals in being fulfilled?

Model checking is a formal technique to automatically verifying properties over a finite-state model that can represent different system aspects, e.g., its behaviour in the form of components interaction or a higher-level activities workflow. The advantage is to provide, through an appropriate verification model and properties defined with linear, temporal

order or even probabilistic logic, complex querying capabilities about the correctness of the system model and its conformance in fulfilling both functional and non-functional requirements.

In specific, the probabilistic model checking (PMC) has been largely explored and is supported by tools such as PRISM model checker. As long as the verification model built from a behaviour specification is precise, this method provides a trusted estimation for metrics as those related to dependability attributes. For instance, PMC can estimate the probability of a system to reach its final success state in a DTMC model based on the reliability of the components involved in the execution, which defines the global reliability of this system or for the analysed activity [4, 15].

Following a similar principle of the component-based verification, the investigation focus has been narrowed to the feasibility of a goal-oriented PMC approach for which the leaf-tasks representing high-level system activities are mapped to a probabilistic model to provide both qualitative and quantitative analysis for the probability of successfully fulfilling different system goals. Accordingly, our first research question was refactored to:

> **Research Question 1 (RQ1'):** Given a correct and consistent goal model for which goals are ultimately satisfied by system tasks and a set of leaf-tasks that fulfils some specific system goal, is it possible to analyse the probability of achieving this goal through a probabilistic model checking technique?

As described by the contextual goal model [1], the contextualization of the informations gathered at RE and at design phases becomes imperative once its validity may be threatened by changing environment conditions. Context variation may relativize the need for a goal, restrain the adoptability of alternative means and also affect the quality of available means in fulfilling their goals. Hence, a goal-oriented verification must consider these effects in the analysis results. From this, our second research question arises:

> **Research Question 2 (RQ2):** Is it possible to consider the context effects over what goals are required, what alternatives are adoptable and the quality of each alternative in the probabilistic verification process?

As discussed earlier, self-adaptation must rely on an adequate analysis that conforms to the complexity and criticality of the metrics being analysed. Reliability is an important dependability attribute and a first class requirement for self-adaptation, as it defines the continuity of a correct service, i.e., the absence of failures. In goal models, runtime variability solving is based on runtime inputs and non-functional criteria measured or

estimated for each alternative [19]. Considering a self-adaptation analysis for solving variability, our third research question is defined as:

> **Research Question 3 (RQ3):** Is it feasible to employ a goal-oriented reliability verification based on parametric PMC as part of a self-adaptation analysis for solving variability?

Finally, considering the complexity and the analysis overhead caused by the manual generation of a probabilistic verification model from a goal model as required by our previous research questions, a fourth and final research question arises:

> **Research Question 4 (RQ4):** Is it possible to automatically generate the probabilistic verification model for the reliability analysis defined in RQ1-3 from a system goal model modelled and stored by some GORE tool?

## 1.2 Proposed Solution

To the best of our knowledge, a goal-oriented approach for dependability analysis based on a formal verification method have not yet been proposed. In order to achieve the probability estimation of the fulfilment of different system goals, it is important to realize that the primitive goal models as proposed in [7, 9, 18] are designs artefacts and, as argued by Dalpiaz et al., not tailored for runtime analysis. Nonetheless, the RGM filled this gap by adding a behavioural specification to the static goal model [8].

In contrast to previous approaches for dependability analysis through PMC, this work proposes a verification that is directly mapped to a system runtime goal model. Instead of building the probabilistic verification model from behaviour models like UML activity and sequence diagrams, we benefit from the RGM syntax that specifies the behaviour for goals and tasks to build a high-level DTMC model for reliability verification of different system goals (RQ1). This model, comparable to a high-level UML activity diagram, should also include the context effects over goals, means and qualitative metrics as defined by the contextual goal model [1] (RQ2).

PRISM tool provides a rich analysis environment for PMC. However, a runtime self-adaptation analysis must be automatic, i.e., based on computable processes without human intervention. The parametric PMC answers this requirement by generating a parametric formula for a given probabilistic model and property to be analysed with parameters in the place of constant variables. Thus, different runtime analysis may be performed by just initializing the parameters with values corresponding, e.g., to a specific context of operation, and evaluating the resulting formula. In our proposal, we investigate the

feasibility of a parametric PMC as the formal method for the goal-oriented reliability analysis in self-adaptive systems (RQ3).

Finally, to reduce the analysis overhead and to improve the scalability and the feasibility of our goal-oriented reliability analysis, a JAVA implementation was integrated to a tool that supports TROPOS goal modelling and analysis, namely TAOM4E. This extension provides automatic generation of a DTMC model in PRISM language from a RGM with or without additional context effect notations (RQ4).

## 1.3 Evaluation

The proposal has been evaluated with the application of the goal-oriented reliability analysis for self-adaptive systems 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 [12, 13]. Instead of a static environment, the MPERS is conceived to allow patients with different health risk degrees to preserve their mobility while they are monitored and assisted. For being a mobile system, MPERS is affected by context variations and self-adaptation becomes a mandatory feature to avoid failures, as safety is of paramount importance for this system.

The MPERS features were based on real emergency response systems available at the industry and also at the BAN explored in previews works [12, 15]. In this evaluation, we focus on the development process of the MPERS and demonstrate the feasibility of the goal-oriented approach for the dependability analysis with a probabilistic model checking technique. In specific, our main goal is to provide reliability analysis capability for the MPERS system with a verification model mapped to the contextual/runtime goal model of the system-to-be, empowering both the analysts (design-time) and the system itself (runtime self-adaptation) with a precise method for reliability verification coped to the goal model specification.

## 1.4 Contributions Summary

This section summarizes the contributions of this proposal.

1. An goal-oriented approach for reliability analysis based on probabilistic model checking.

    - Use of a probabilistic model checking whose verification model is built from a runtime goal model with leaf-tasks as system activities.

- Inclusion of the context effects over goals and tasks in the verification model as defined by the contextual notations in the runtime goal model.
- Probabilistic and temporal querying capabilities for different metrics related to the reliability of one or more goals in the system goal model.

2. An automated generation of the DTMC model in PRISM language from a contextual/runtime goal model.
   - 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 parallel or sequential executions of the same task.
   - Definition of conversion rules between different decomposition types and behaviours rules in the contextual/runtime goal model to a DTMC model in PRISM language.
   - JAVA implementation of DTMC generator integrated to the TAOM4E tool that supports the TROPOS methodology using the Eclipse plugin architecture.

## 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 goal-oriented reliability analysis. Chapter 5 describes and evaluates the proposal with the MPERS cases study. Chapter 6 presents the rules for the automatic generation of a DTMC model in PRISM language from the runtime goal model with additional context effects notations. Chapter 7 validates the proposal with some relevant metrics. Finally, Chapter 8 concludes this work with final considerations about the current proposal and our future work.

# Chapter 2

# Background

## 2.1 Goal-oriented Requirements Engineering

Goal-oriented requirements engineering captures 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.

Frameworks and methodologies like the i*, KAOS and TROPOS represent the foundations for the goal model analysis used by a variety of other proposals [7, 9, 18]. Despite some syntax differences, most goal-oriented approaches share a set of common and more important concepts:

### 2.1.1 Intentional 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 represent non-functional requirements. E.g.: vital signs are monitored, emergency is detected, emergency center is notified (hard goals) and emergency awareness, precise assistance, feel supported (softgoals).

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

- **Resource:** an information data or a physical resource that is generated or required by an actor.

### 2.1.2 Relations

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

- **Means-end:** a relation that indicates 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).

- **Dependency link:** a delegation of a goal, task or resource (*dependum*) from an actor (depender) to another (dependee).

## 2.2 TROPOS Methodology

TROPOS is a GORE methodology based on the i* framework [7]. Its main improvement is the addition of new phases of requirements engineering, architecture 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.

## 2.3 Goals, Means and Contexts

Context may be defined as the reification of the environment that surrounds the system operation [2]. Contexts, as already stated, may not be static, but dynamic, and a system has no control over the context variation. Accordingly, a system must be able to support different contexts of operation without violating its functional and non-functional goals. To achieve this, systems must be able to monitor the state of its surrounding environment and take adaptive actions regarding the alternatives used for fulfilling its goals.

In a contextual goal model (CGM), 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 [1]. Root goals and higher-level strategical goals are generally not contextualized as they represent the main purpose of a system [2]. As these goals are decomposed in more granular sub-goals, a context condition may affect:

1. If one or more goals are required for that context, limiting 'what' a system should do. For instance, the goal 'track person's location' ceases to be required in the context 'patient is at home'.

2. If a sub-goal or task is adoptable, limiting the 'means' to fulfil a required goal. For instance, 'track by GPS' may not be used in the context 'battery is low' (stakeholder preference) or 'no GPS signal' (technical impediment).

3. The positive, neutral or negative contribution of one alternative to a qualitative softgoal or to a non-functional metric. For instance, the precision of each geolocation

method - voice call, mobile triangulation and GPS variate according to contexts like the health condition of the person and the strength of mobile and GPS signals.

The third effect is the main focus of this work, as it is related to non-functional requirements such as dependability attributes. Also, the other effects must also be considered by the verification model, including the variability at what goals are required at each context and what means are adoptable, resulting in a more realistic and precise verification of corresponding metrics.

## 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 in goal models have different meanings according to the development phase it takes place.

### 2.4.1 Design-time analysis

At design time, multiple alternatives are elicited trough goal-oriented analysis, but not all are selected to be part of the system-to-be. In traditional GORE, contribution analysis is used for the the comparison of how each alternative contributes for one or more softgoals. Usually, only one alternative with the more positive contribution sum is selected for the system-to-be. Or, as in the simplistic example of Figure 2.1, the decision relies on the softgoals priorities [10].

### 2.4.2 Runtime analysis

In contrast to the design-time variability in goal models, runtime variability depends on runtime input and must be preserved at runtime, i.e., variability is inherited by the solution design [19]. The purpose is to select the valid alternative according to stakeholders preferences and the current context of operation.

The contextual goal model tackles the influence of the context on the autonomous decision of which alternative should be selected [1]. For instance, if the monitored traffic condition is 'jammed' and the patient's condition is 'critical', an emergency chopper is selected for assistance in the place of an ambulance. We call this 'direct context implication'.

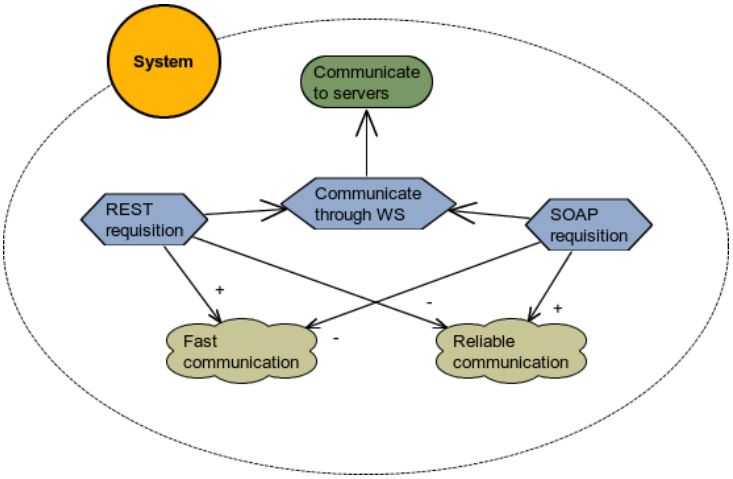



Figure 2.1: Contribution analysis in TROPOS GORE.

In other cases, alternatives should be monitored and analysed in terms of non-functional metrics to decide which one is suitable or for selection. Here, the context of operation does not directly defines the adoptable alternative, but its effect on the quality of each alternative is considered as a decision criteria. For instance, a more complex analysis must estimate the probability of the ambulance to reach the patient in less than X time units given a traffic condition. We call this 'indirect context implication'.

In Figure 2.2, the availability of the emergency resources are contexts that directly restricts the adoption of corresponding means to reach a patient. In contrast, the traffic condition affects the time to reach the patient, i.e., this context indirectly affects the selection of an ambulance and requires further analysis to estimate the assistance time in different traffic conditions. Also, the context 'patient health' affects the time restriction in the 'fast assistance' quality goal. This scenario leads to the following question: Among the available functional alternatives, which also fulfils the quality goal in the current context?

## 2.5 Dependability Analysis

The concept of dependability is related to dependence and trust as well as to the ability of a system to avoid failures that are more frequent and more severe than certain threshold [3]. 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.

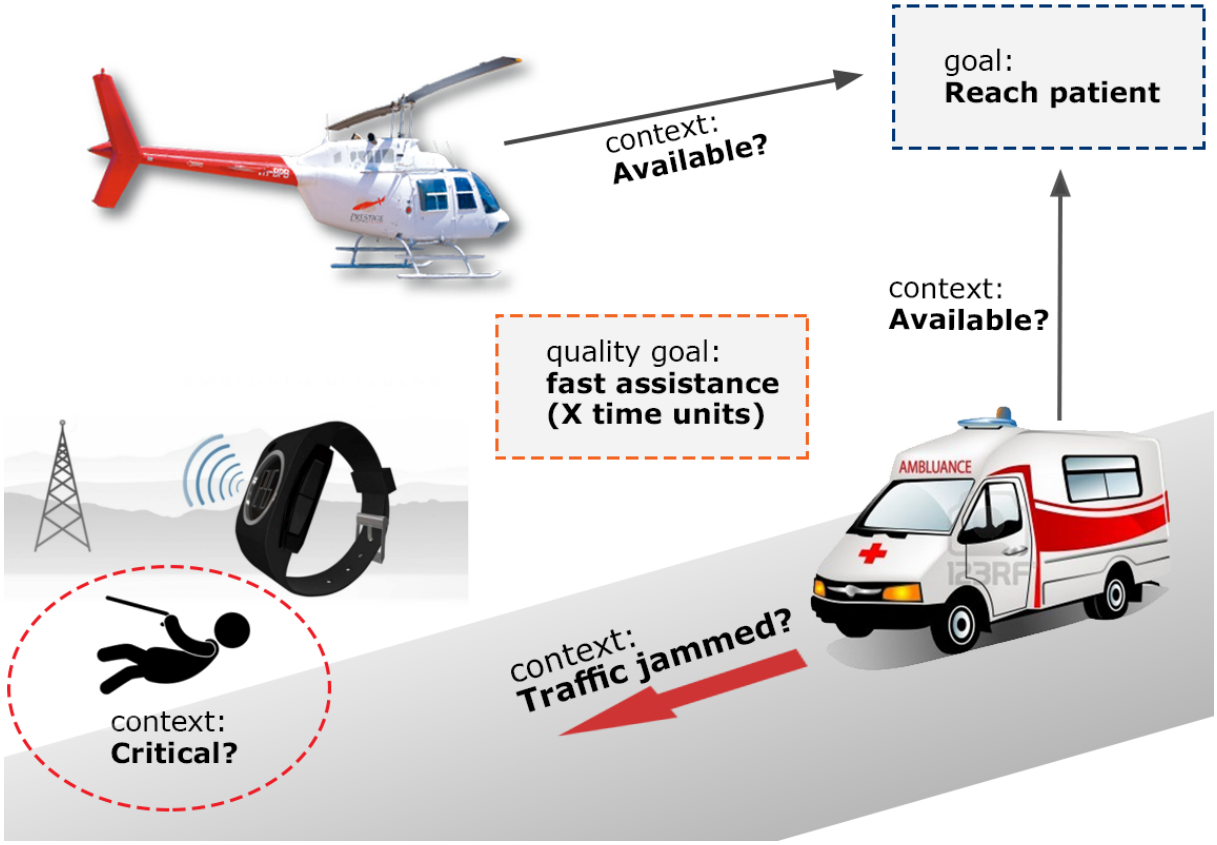



Figure 2.2: Contribution analysis in TROPOS GORE.

- Safety: absence of catastrophic consequences on the user(s) and the environment.
- Maintainability: ability to undergo modifications and repairs.

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 and behaviour models fails to describe the system: either the goals or the means to fulfil then are incorrect, inappropriate 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 [3].

The scope of this work is restricted to specification violations, i.e., we assume that a system specification is complete and consistent. Failures are limited 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 analysis in a self-adaptation feedback loop.

## 2.6 Self-adaptation

## 2.7 Probabilistic Model Checking

A model checking is a formal method that aims to automatically verify the correctness of a system model. Probabilistic 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.

### 2.7.1 PRISM tool

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 [11].

PRISM is suitable for different kinds of model evaluations depending on the abstraction level, the type of probabilistic model and the PCTL properties to be analysed. Both qualitative and quantitative analysis are available features in the simulation/verification environment. Other environments for modelling and PCTL property specification are also available in the tool.

### 2.7.2 PRISM language

PRISM language offers a rich set of constructs that may represent system modules, components and others architectural and design abstractions. Modules are the main structure in a PRISM model. Modules are composed of variable and commands. The first describe the finite states a module can be in. The former describe the behaviour of a module, i.e., the actions that may result in state transitions and are guarded by predicates which in turn can be composed of any variable in the model. Finally, labels are used for command naming and synchronization. A DTMC command in PRISM takes the following form:

$$[action/label] < guard > \rightarrow < probability > : < update >;$$

### 2.7.3 PARAM

More recent PRISM versions also supports a parametric model checking, that is, instead of constant variables whose values must be initialized before verification, parameters replace these variables and a parametric formula is generated. Nonetheless, a previous PRISM extension tool named PARAM also provides parametric verification and has been further evaluated in case studies. For this reason, in this work have used PARAM for the generation of parametric formulas corresponding to a given probabilistic model and a PCTL property.

### 2.7.4 Dependability analysis with PMC

As it will be explained in later sections, goal models may be 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 at runtime self-adaptation. 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.

## 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) and for the context effect notations as proposed by the contextual goal model (CGM). Further details of the grammar with both parser rules and lexical terms is given in later section.

## 2.9 Mobile Personal Emergency Response System

The MPERS case study will be further detailed in later chapter as the proposed goal-oriented reliability analysis based on PMC is described with the MPERS goal models of requirements engineering phases of TROPOS methodology. As such, this section will be limited to 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 employ all 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.

Environment changes is an important factor for MPERS, as the following conditions may change:

- The battery of the mobile device;
- The battery of vital signs sensors;
- The disc memory used for storing the vital signs history;
- The mobile signal used for data communication and for geolocation triangulation;
- The GPS signal used for geolocation;
- The health risk of the patient;

For all these environment conditions, a context analysis as proposed by Ali et al. may define the rationale for context monitoring, i.e., for how each context condition may be asserted as true or false through the monitoring of environment information. Some contexts are mapped to single monitorable facts, like 'GPS signal' and 'battery life'. Other like 'patient health risk' are composed of multiple facts in a propositional formula.

# Chapter 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. CGM provides a more realistic and precise contribution analysis contextualized by environment conditions, but does not change the nature of the contribution analysis process.

In contrast to CGM, the main contribution of our work is the goal-oriented probabilistic verification of reliability related metrics that needs a more robust and less biased approach instead of the existing contribution analysis that is based on the direct evaluation of the forward impact between goals/tasks and softgoals. Our work has also 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 to contextually estimate the probability of different system goals in being fulfilled as part of a more precise analysis.

## 3.2 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.

## 3.3 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.

## 3.4 Awareness Requirements

Souza et al. [AwaReq] proposed the Awareness Requirements (AwReq) as a meta-requirements class in a goal model, i.e., AwReqs specify the success/failure rate and other constraints for requirements in the model, including goals, tasks, domain assumptions and even other AwReqs (*-meta-requirement). The objective is to enrich the original goal model with constraints for the system behaviour and to enable 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.

## 3.5 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.

# Chapter 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 measures depending on system behaviour or more complex analysis techniques.

In a previous work and following, we have extended the contextual goal model to tackled the context variation effects on dependability attributes based on declarative fuzzy logic rules [14]. From our experience, we have considered that a more scalable and precise approach for the verification of dependability metrics in dynamic contexts

was needed. Probabilistic model checking emerged as a potential option to cope a goal-oriented methodology with a formal method for dependability analysis.

To the best of our knowledge, formal methods have not yet being used for a goal-oriented dependability analysis. As a fault forecasting approach, the purpose would be to characterize the dependability and the security of a system and to compare concurring alternative solutions - in this case, goal model alternatives - according to one or more attributes. In specific, the reliability attribute should be the focus of this proposal and the resulting verification model should estimate and verify reliability related metrics as a design-time, manual analysis and specially as part of a runtime automated analysis as part of a self-adaptation loop. Finally, the context effects over goals, means and metrics as described by the CGM should be considered in the analysis.

## 4.2 Requirements for the Goal-oriented and Contextual Dependability Analysis

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 **Verification scope:** The verification scope may be restricted to a part of the system (local goals) or encompass the whole system (root goal), according to analysts decision.

R.3 **Model generation:** The probabilistic model representing the activities of runtime goal models should be automatically generated from a runtime goal model created in TROPOS late requirements engineering phase.

R.4 **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.5 **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.6 **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.7 **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.

# Chapter 5

# Goal-oriented PMC for Reliability Analysis of Self-adaptive Systems

This chapter present our proposal for goal-oriented contextual dependability analysis applied to the MPERS case study. Both TROPOS early and late requirements analysis phases are fully presented, as well as the probabilistic verification process that requires a behaviour specification and the additional contextual notation regarding context effects.

The next sections are structured as follows. First, a goal modelling and analysis as described by TROPOS methodology is performed [7]. Next, the RGM behaviour rules are further explained and then compared to UML activity diagrams. After this, non-functional constraints are associated to the MPERS goal model. Then, the high-level DTMC model for MPERS is built and details about the conversion from a RGM to a DTMC are provided. Next, PCTL properties representing the reliability of fulfilling system goals are defined. Finally, we demonstrate how our goal-oriented PMC approach may be employed in reliability analysis.

Figure 5.1 illustrates the goal-oriented verification process that starts with original phases involving goal-oriented modelling and ends with the generation of a high-level DTMC model used for reliability analysis.

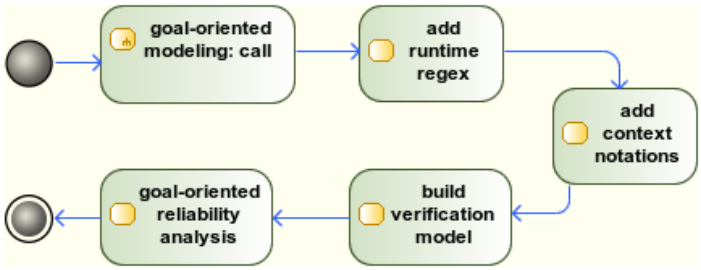



Figure 5.1: Building a DTMC model from a contextual/runtime goal model.

## 5.1 Preliminary goal modelling and analysis

### 5.1.1 TROPOS Early Requirements Phase

In early requirements phase, stakeholders and their needs are modelled in a goal model diagram. Each actor may be a depender or a dependee of a goal, task or resource dependency. In this phase, only social dependencies to the system actor and between 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 5.2. 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.

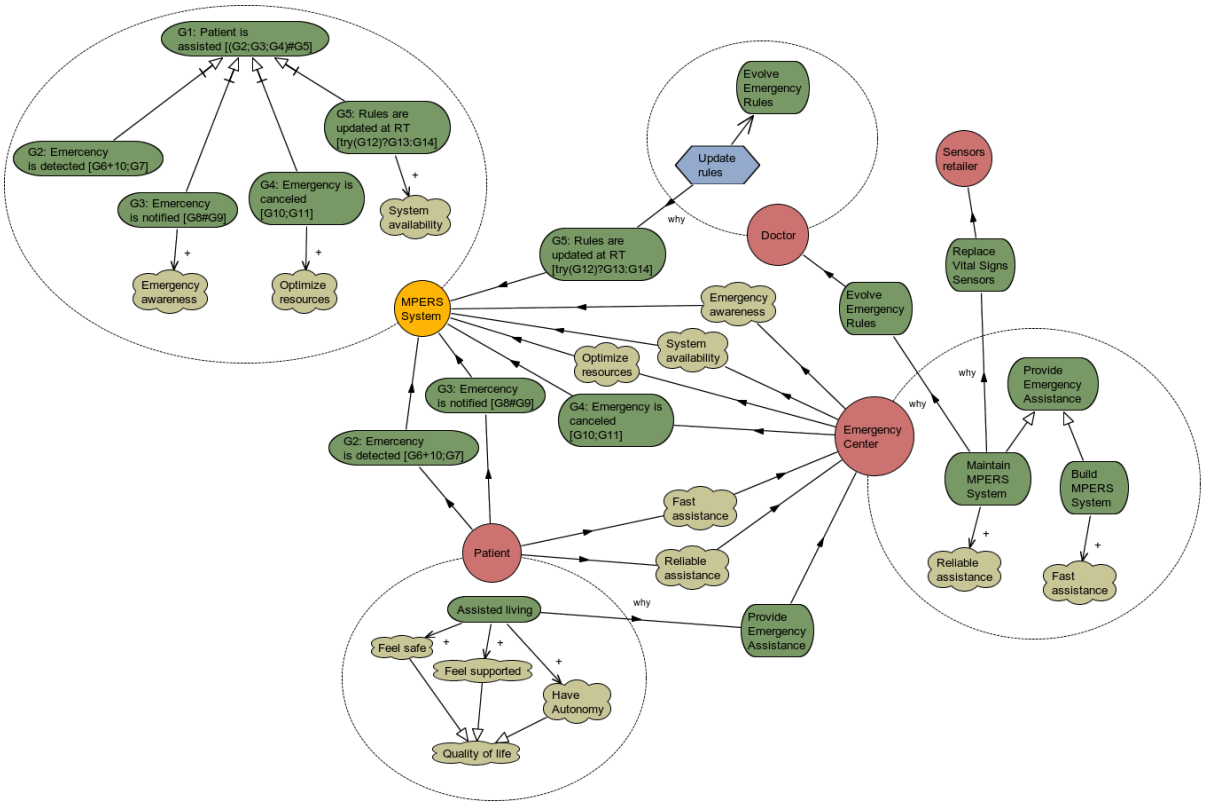



Figure 5.2: MPERS at TROPOS early requirements phase

From the diagram in Figure 5.2 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 and softgoals are directly or indirectly related to stakeholders needs.

The distinguished 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 5.3 illustrates the late requirements diagram for the MPERS.

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 explored the use of the PMC technique for the reliability verification of the system goal model resulting from 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 proposed goal-oriented contextual dependability analysis.

### 5.1.3 Non-functional requirements analysis

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

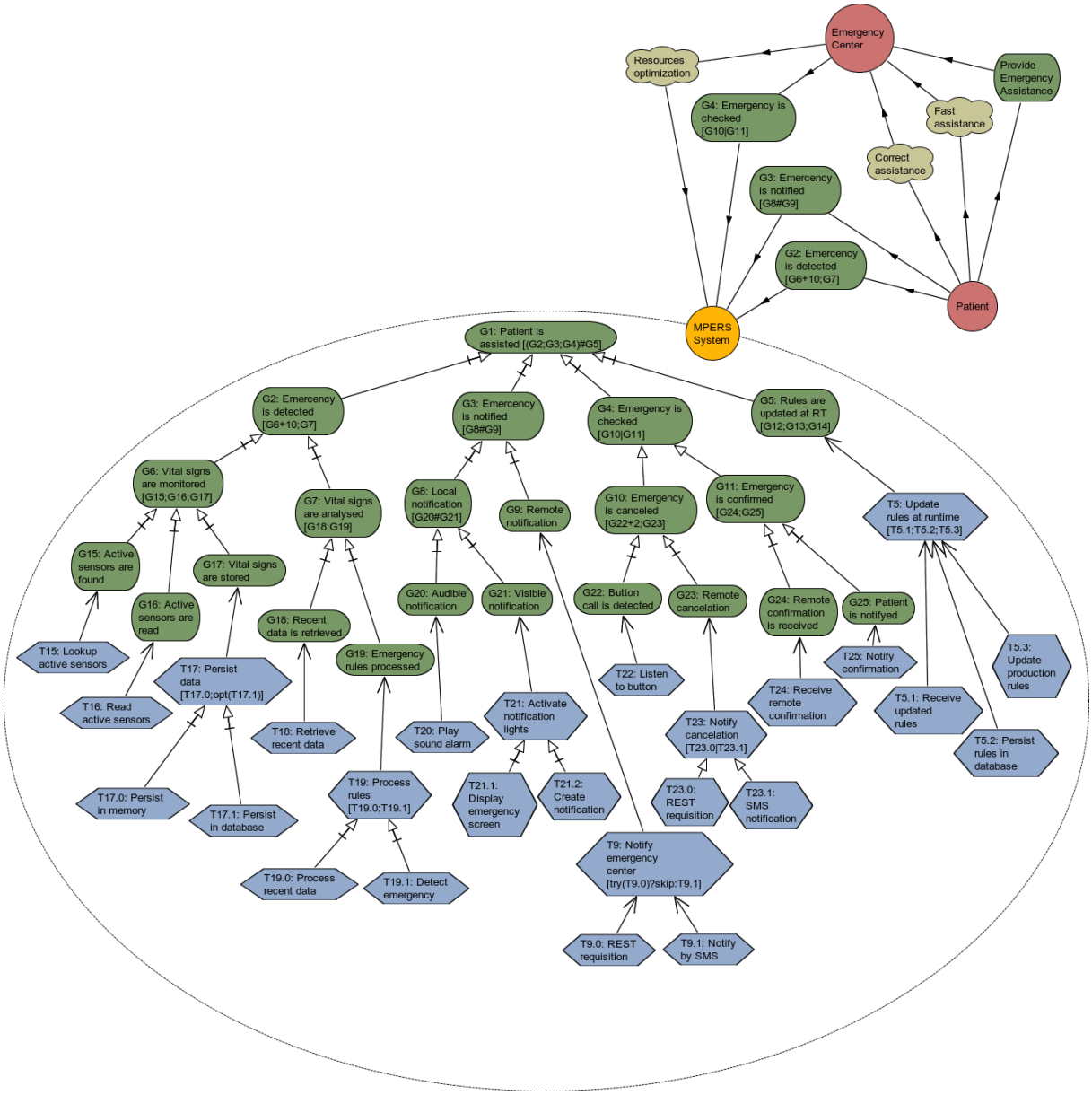



Figure 5.3: MPERS at TROPOS early requirements phase

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 the NFRs for the MPERS system actor.

Similarly to the Awareness Requirements by Souza et al. [16], 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.

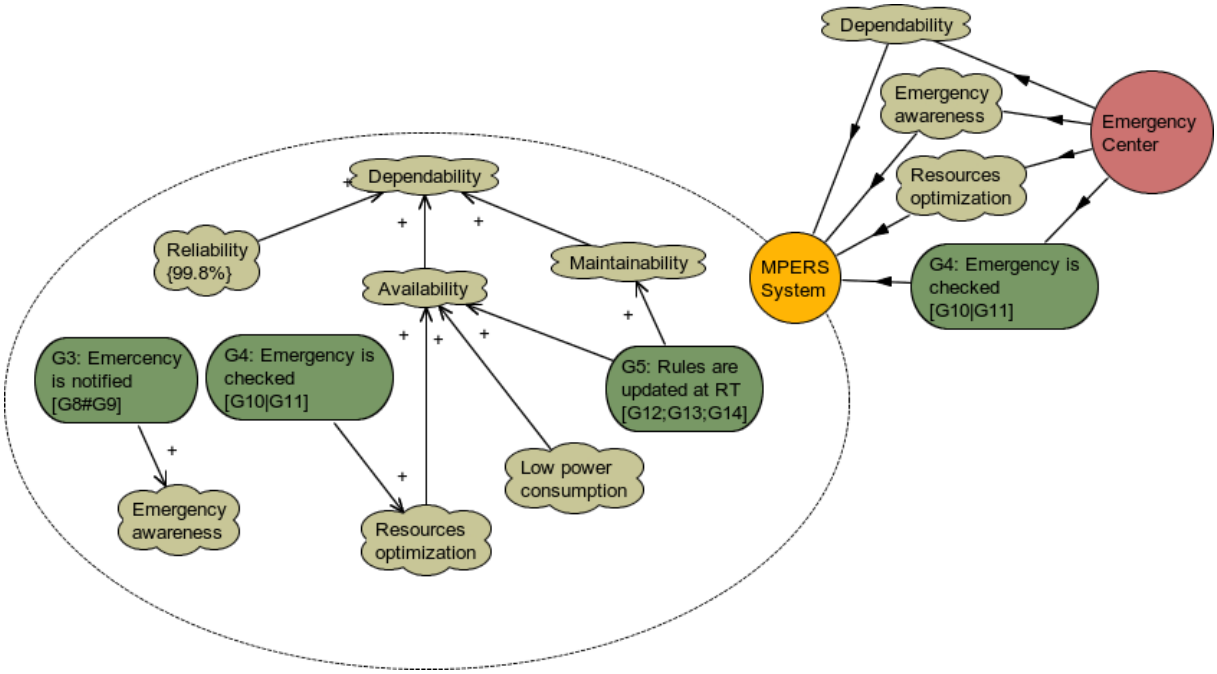



Figure 5.4: MPERS non-functional requirements.

Hence, the reliability metric depends on the probability of system functionalities to successfully meeting their goals.

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.

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 proposal focus on reliability analysis and will not discuss the verification of other relevant NFRs.

## 5.2 RGM Behaviour Analysis

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 5.1 provides a textual description of each RGM rule and the corresponding meaning in terms of what behaviour it specifies and also an example from the MPERS runtime goal model of Figure 5.3. A formal and detailed description can be found in the RGM reference publication [8].

| **Expression** | **Meaning** | **Example (MPERS)** |
|---|---|---|
| E1;E2 | A goal/task E1 must be fulfilled/executed before E2. | $G1$;$G2$;$G3$;$G4$ |
| E1#E2 | Interleaved fulfillment/execution of goal/task E1 and E2. | $(G1; G2; G3; G4)\#G5$ |
| E+n | Goal/task E must be fulfilled/executed $n$ times, with $n > 0$. | $G22 + 2$ |
| E#n | Interleaved fulfillment/execution of $n$ instances of E, with $n > 0$. | - |
| E1\|E2 | Fulfillment/execution of goal/task E1 is alternative with respect to E2. | $T23.0 \| T23.1$ |
| opt(E) | Fulfillment/execution of goal/task E is not mandatory. | $opt(T17.2)$ |
| try(E)?E1:E2 | If goal/task E succeeds, E1 must be fulfilled/executed; otherwise, E2. | $try(T9.0)?skip : T9.1$ |
| skip | No action. Useful for conditional ternary expressions involving two elements. | $try(T9.0)?skip : T9.1$ |

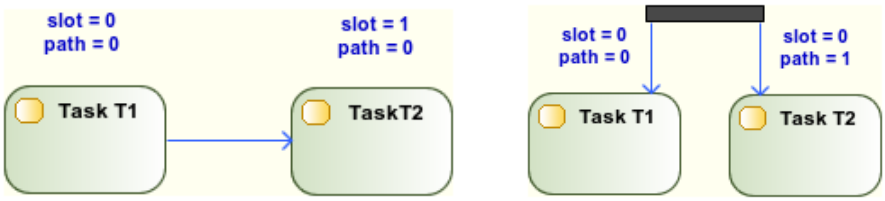
Table 5.1: Description of RGM behaviour rules used by the proposal.

The following sections further describe each of the RGM rules with additional UML activity diagrams to illustrate the resulting behaviour on the execution of system leaf-tasks that are later mapped to a probabilistic verification model. The *slot* and *path* notations refers to discrete time abstractions in a DTMC model as it will be explained in Section 5.2.6.

### 5.2.1 Temporal order

The most trivial behaviour rule consist of the temporal fulfilment and execution order of goals and tasks. In a RGM, the ';' and '#' symbols represent sequential and parallel

temporal orders, respectively. Figures 5.5a and 5.5b illustrate this behaviours in UML activity diagrams.




(a) Sequential tasks T1;T2. (b) Interleaved tasks T1#T2.

### 5.2.2 XOR-decomposition

In the case of OR-decomposition with additional '|' behaviour rule, a resulting XOR-decomposition specifies that only one goal/task among two or more may be selected, i.e., they are mutually exclusive. Figures 5.6 illustrate the alternative OR-decomposition in an UML activity diagram.

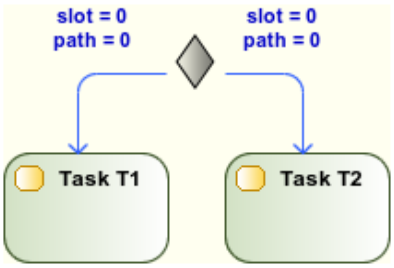



Figure 5.6: Alternative tasks T1|T2.

### 5.2.3 Optional decomposition

Given a decomposed goal or task, its fulfilment or execution may be optional, meaning that at least one other goal/task must necessarily be fulfilled/executed. It is trivial to realize that optional behaviour rules $opt(E)$ are only possible in OR-decompositions. Figure 5.7 illustrate an optional decomposition.

### 5.2.4 Conditional decomposition

Some goals/tasks are conditioned to the fulfilment/execution of another goal/task. In such cases, a AND/OR-decomposition have an additional ternary rule $try(E1)?E2 : E3$ where E2 is conditioned to the success of E1, while E3 is conditioned to its failure. The

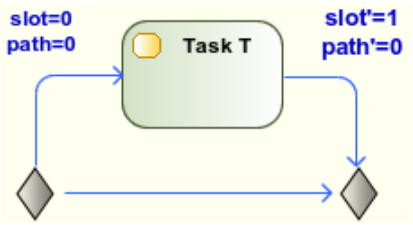



Figure 5.7: Optional task T.

skip term is used if no further behaviour is conditioned to either the success or the failure of E1, for instance, in $try(T1)?skip : T2$ where task T2 is only required for execution if T1 fails and no further behaviour is expected if T1 succeeds. Figure 5.8 illustrate a conditional decomposition.

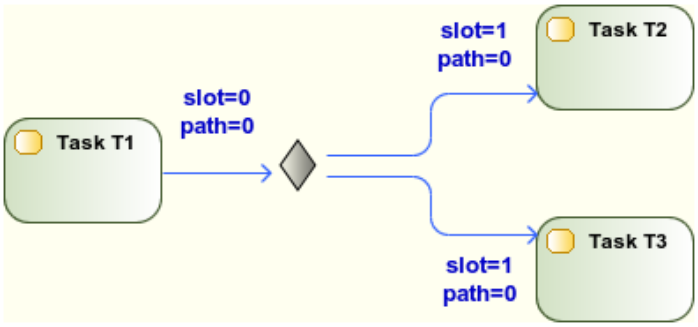
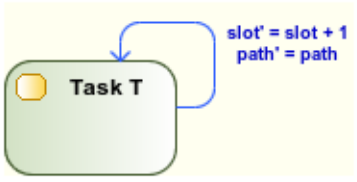


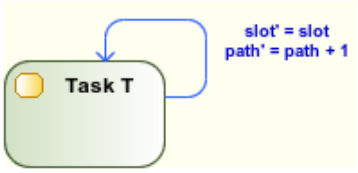

Figure 5.8: Optional task T.

### 5.2.5 Cardinality

A small variation of the original RGM syntax 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 DTMC verification model from a runtime goal model. Figures 5.9a and 5.9b respectively illustrate sequential and interleaved task behaviours.




(a) Sequential executions of task T.




(b) Interleaved executions of task T.

### 5.2.6 RGM - UML activity diagram comparison

A similar behaviour specification achieved with the RGM in Figure 5.3 could be provided by an UML activity diagram with activities as the leaf-tasks in 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 5.10 presents an activity diagram corresponding to the MPERS RGM.

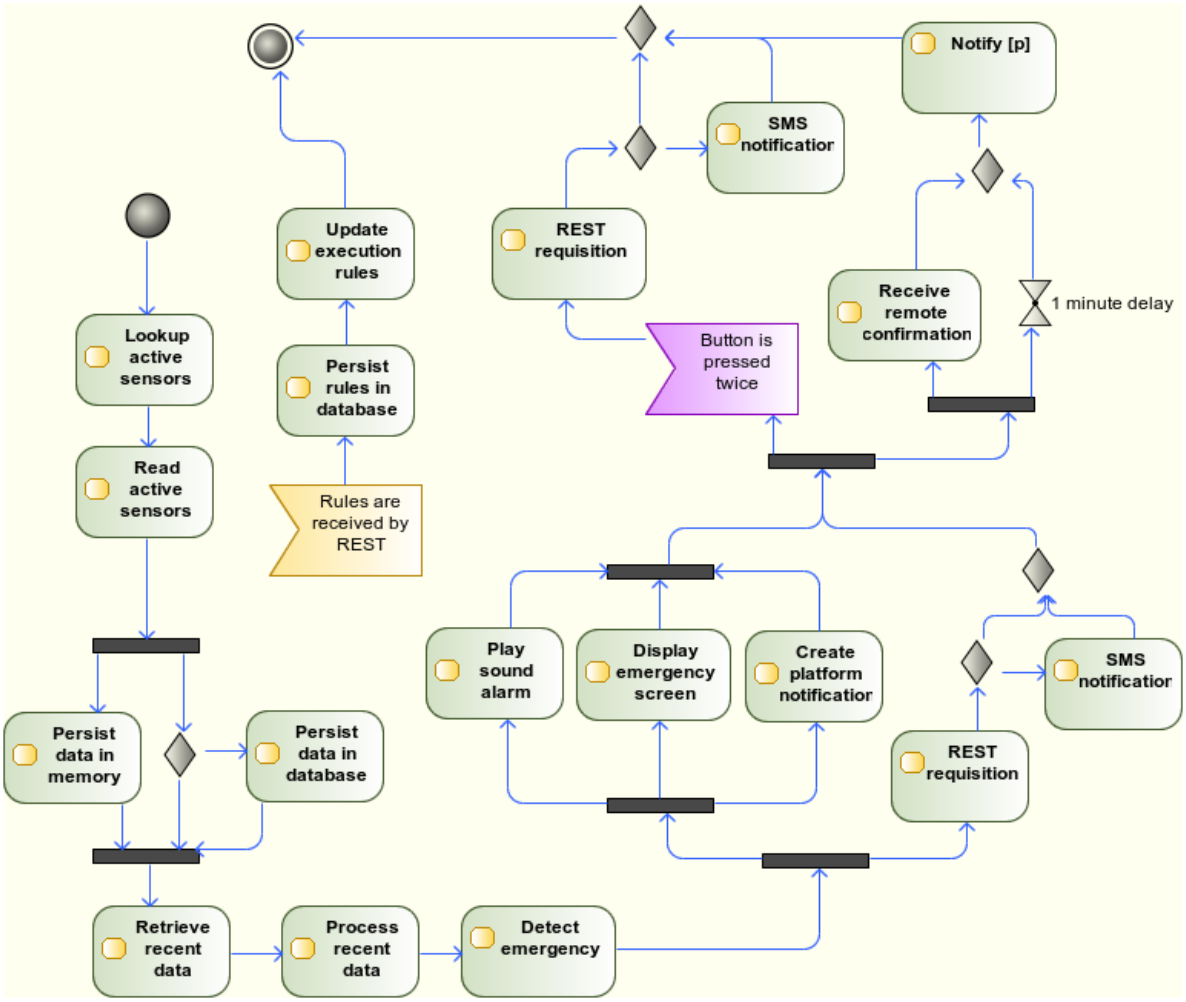



Figure 5.10: MPERS tasks represented by a UML activity diagram

Among its limitations, RGM does not express that an emergency has to be confirmed after a time (clock) 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 specification language. Still, sequential, parallel,

alternative, optional and conditional execution flows as well as multiple executions of the same task can be expressed by the RGM, providing a rich behaviour specification.

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 documentation, team communication and for conformance verification at both design time - e.g., through model-based verification - 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. In this work, RGM is used as input for our goal-oriented probabilistic verification.

## 5.3 CGM Context Analysis

## 5.4 Goal-oriented probabilistic verification

This section describes the application of a goal-oriented PMC technique for the reliability verification of a runtime goal model with additional context effect notations. We call it a goal-oriented probabilistic verification or goal-oriented dependability analysis because the probabilistic model, in this case a DTMC model, is built directly from a runtime goal model with the purpose of evaluating PCTL properties related to the reliability of the system, i.e., to the success probability of different goals in being fulfilled.

### 5.4.1 Leaf-tasks as DTMC modules

The MPERS RGM in Figure 5.3 expands its main goals in further subgoals 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. Figure 5.11 presents a system task as a PRISM module.

In the DTMC model, leaf-tasks have their execution state mapped to a variable in the module (sT15 variable in Figure 5.11). From the RGM original set of goal/tasks instance states, we considered the following values:

- Init(sTask=0): corresponds to the initial/ready 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 to be analysed, or directly to the final success state, if the opposite.

```
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
```

Figure 5.11: A PRISM DTMC module representing T15 (mandatory) task.

- Running(sTask=1): corresponds to the execution 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 5.11).

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

- Failure(sTask=3): the opposite from the final success state, meaning that a singular task execution has failed.

Figure 5.12 illustrate leaf-tasks' states in a UML state diagram.

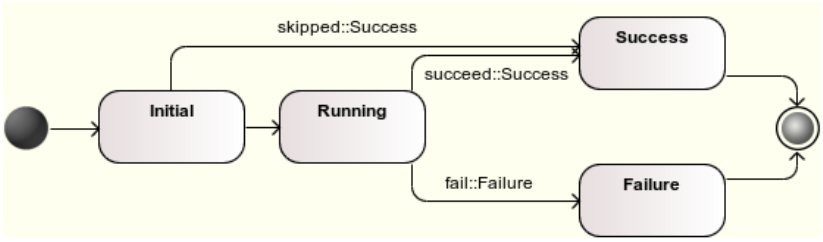



Figure 5.12: State diagram for leaf-tasks in the DTMC module.

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

The first step in building a DTMC from a RGM is parsing the behaviour rules. All goals and tasks elements in the model receive an unique identification (ID). Numeric IDs are prefixed with an 'G', in the case of goals, and 'T', in the case of tasks. For each decomposed element, a corresponding runtime regex consisting of IDs and rules specifies the behaviour of all immediately underlying goals and tasks. Parenthesis are used for

grouping related goals/tasks and for clarification purpose only, as temporal order is parsed from the type of the rule and its position in the regex.

Given a set of leaf-tasks that fulfils a chain of subgoals until a certain goal G, a high-level DTMC model composed of modules for each leaf-task states and transitions is build. This model must abstract the same behavioural of the corresponding RGM, i.e., it must preserve activities temporal order and other semantics specified by the runtime regex. We call the resulting verification model of a high-level DTMC because leaf-tasks are similar to activities in a UML diagram whose behaviours could be further detailed, e.g., by sequence diagrams.

Each leaf-task in the DTMC model starts at a discrete time slot. Time slots maps the sequence order of tasks executions. To differentiate the time of parallel tasks, time path is incremented by interleaved task executions, while time slot is incremented by sequential executions. Next, RGM rules with corresponding DTMC modules are presented, as well as the increment in the time slot and time path caused by each rule in accordance to the UML activity diagrams in Section 5.2.

### Sequential order

Sequential tasks ($T1;T2$) have subsequent time slots, meaning that T2's initial transition is synchronized to T1's final transition through PRISM labels. In contrast, 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 of these tasks are interleaved. Figure 5.5a and 5.5b present the DTMC modules for each case.

### Alternative execution

In alternative execution, or XOR-decomposition, an additional parameter in the model defines which alternative from a set of two or more is selected for analysis. Figure 5.15 presents the DTMC modules for alternative tasks T9.00 and T9.01. For this behaviour rule, both time slot and time path are equal for the alternatives, as only one task may be selected. The increment is defined by surrounding ';' or '#' rule.

### Optional execution

Similarly to alternative execution, optional tasks are enabled for analysis by an additional parameter in the model. Figure 5.16 illustrates the DTMC module for optional task T17.2. Time slot or path increment is also defined by surrounding ';' or '#' rule.

```
const double rTaskT15;

module T15_LookupActiveSensors
        sT15 :[0..3] init 0;

        [success0_0] 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

const double rTaskT16;

module T16_ReadActiveSensors
        sT16 :[0..3] init 0;

        [success0_1] sT16 = 0 -> (sT16'=1);//init to running
        [] sT16 = 1 -> rTaskT16 : (sT16'=2) + (1 - rTaskT16) : (sT16'=3);//running to final state
        [success0_2] sT16 = 2 -> (sT16'=2);//final state success
        [failT16] sERROR = 0 & sT16 = 3 -> (sT16'=3);//final state fail
endmodule
```

Figure 5.13: Sequential tasks T15 and T16 as DTMC modules with final transitions of the first module synchronized to the initial transition of the former.

```
const double rTaskT21_0;

module T21_0_DisplayEmergencyScreen
        sT21_0 :[0..3] init 0;

        [success0_7] sT21_0 = 0 -> (sT21_0'=1);//init to running
        [] sT21_0 = 1 -> rTaskT21_0 : (sT21_0'=2) + (1 - rTaskT21_0) : (sT21_0'=3);//running to final state
        [success1_8] sT21_0 = 2 -> (sT21_0'=2);//final state success
        [failT21_0] sERROR = 0 & sT21_0 = 3 -> (sT21_0'=3);//final state fail
endmodule

const double rTaskT21_1;

module T21_1_CreatePlatformNotification
        sT21_1 :[0..3] init 0;

        [success0_7] sT21_1 = 0 -> (sT21_1'=1);//init to running
        [] sT21_1 = 1 -> rTaskT21_1 : (sT21_1'=2) + (1 - rTaskT21_1) : (sT21_1'=3);//running to final state
        [success2_8] sT21_1 = 2 -> (sT21_1'=2);//final state success
        [failT21_1] sERROR = 0 & sT21_1 = 3 -> (sT21_1'=3);//final state fail
endmodule
```

Figure 5.14: Interleaved tasks T21.0 and T21.1 as DTMC modules with initial transition synchronized.

### Conditional execution

In conditional execution, labels are used to condition the execution of tasks to the success and failure of a third task. Figure 5.17 present the DTMC module for conditional tasks T9.0 and T9.1.

### Cardinality

Cardinality is also supported by the runtime regex specification. In our proposal, execution cardinality is represented by the number of retries of a given task transition

```
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
```

Figure 5.15: Alternative tasks T9.00 and T9.01 as DTMC modules with additional integer parameter used for selection.

```
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
```

Figure 5.16: An optional task T17.2 as a DTMC module with additional boolean parameter.

from the initial state to the final state, in the case of a sequential execution ($E+n$), or by interleaved transitions with PRISM language module renaming, in the case of interleaved execution ($E\#n$). Despite the multiplicity of transitions, the discrete time slot is only incremented once, while the time path is not incremented. Figures 5.18 and 5.19 present the DTMC modules for both cases. The former example does not come from the MPERS RGM and illustrate the module renaming.

```
param double rTaskT9_0;
module T9_0_RESTRequisition
        sT9_0 :[0..3] init 0;

        [success0_7] sT9_0 = 0 -> (sT9_0'=1);//init to running
        [] sT9_0 = 1 -> rTaskT9_0 : (sT9_0'=2) + (1 - rTaskT9_0) : (sT9_0'=3);//running to final state
        [success2_8] sT9_0 = 2 -> (sT9_0'=2);//final state success
        [failT9_0] sT9_0 = 3 -> (sT9_0'=3);//final state fail
endmodule

param double rTaskT9_1;
module T9_1_SMSNotification
        sT9_1 :[0..3] init 0;

        [failT9_0] sT9_1 = 0 -> (sT9_1'=1);//init to running
        [success2_8] 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] sT9_1 = 3 -> (sT9_1'=3);//final state fail
endmodule
```

Figure 5.17: Conditional tasks T9.00 and T9.1 as DTMC modules.

```
const double rTaskT22;
const double maxRetriesT22=2;

module T22_ListenToButton
       sT22 :[0..3] init 0;
       triesT22 : [0..2] init 0;

       [success0_8] XOR_T22_T24=0 & sT22 = 0 -> (sT22'=1);//init to running
       [success0_8] XOR_T22_T24!=0 -> (sT22'=2);//not used, skip running


       [] sT22 = 1 & triesT22 < maxRetriesT22 -> rTaskT22 : (sT22'=2) + (1 - rTaskT22) : (triesT22'=triesT22+1);
       [] sT22 = 1 & triesT22 = maxRetriesT22 -> (sT22'=3);//no more retries
       [success0_9] sT22 = 2 -> (sT22'=2);//final state success
       [failT22] sT22 = 3 -> (sT22'=3);//final state fail
endmodule
```

Figure 5.18: Sequential cardinality with n=2 for task T22.

```
const double rTaskT22;

module T22_ListenToButton
        sT22 :[0..3] init 0;

        [success0_8] XOR_T22_T24=0 & sT22 = 0 -> (sT22'=1);//init to running
        [success0_8] XOR_T22_T24!=0 -> (sT22'=2);//not used, skip running

        [] sT22 = 1 -> rTaskT22 : (sT22'=2) + (1 - rTaskT22) : (sT22'=3);//try
        [success0_9] sT22 = 2 -> (sT22'=2);//final state success
        [failT22] sT22 = 3 -> (sT22'=3);//final state fail
endmodule
module T22_ListenToButton_2 = T22_ListenToButton [ sT22=sT22_2] endmodule
module T22_ListenToButton_3 = T22_ListenToButton [ sT22=sT22_3] endmodule
```

Figure 5.19: Interleaved cardinality with n=3 for task T22 (illustrative).

In Table 5.2, 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 presented.

| Task | Time path | Time slot | Optional | Conditional | Alternative | Cardinality |
|---|---|---|---|---|---|---|
| **T15** | 0 | 0 | false | false | false | 1 |
| **T16** | 0 | 1 | false | false | false | 1 |
| **T17.0** | 0 | 2 | false | false | false | 1 |
| **T17.1** | 0 | 3 | **true** | false | false | 1 |
| **T18** | 0 | 4 | false | false | false | 1 |
| **T19.0** | 0 | 5 | false | false | false | 1 |
| **T19.1** | 0 | 6 | false | false | false | 1 |
| **T20** | 0 | 7 | false | false | false | 1 |
| **T21.0** | 1 | 7 | false | false | false | 1 |
| **T21.1** | 2 | 7 | false | false | false | 1 |
| **T9.0** | 3 | 7 | false | false | false | 1 |
| **T9.1** | 3 | 7 | false | **S. T9.0** | false | 1 |
| **T22** | 0 | 8 | false | false | **T24** | **2** |
| **T23.0** | 0 | 9 | false | false | false | 1 |
| **T23.1** | 0 | 9 | false | **S. T23.0** | false | 1 |
| **T24** | 0 | 8 | false | false | **T22** | 1 |
| **T25.0** | 0 | 9 | false | false | **T25.1** | 1 |
| **T25.1** | 0 | 9 | false | false | **T25.0** | 1 |
| **T5.0** | 4 | 0 | false | false | false | 1 |
| **T5.1** | 4 | 0 | false | false | false | 1 |
| **T5.2** | 4 | 0 | false | false | false | 1 |

Table 5.2: Dynamic aspects of the MPERS leaf-tasks in the DTMC model.

### 5.4.3 Gathering leaf-tasks individual reliability

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 5.3, 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 - like their reliability, as the trace between an higher-level, abstract activity and the

corresponding system operation(s) becomes less evident. For a global evaluation of a given metric, PCM technique requires the individual value for parts involved in the evaluated system behaviour, e.g., activities, subactivities or components.

Gathering this information may follow different approaches. For instance, if internal behavioural specification like a sequence diagram is available for leaf-tasks, and if the reliabilities of the involved components are know, then a PMC approach can estimate the individual reliability of leaf-tasks. This value would then be used in the higher-level DTMC model.

The original RGM proposal relies on a strong assumption that goals and tasks instance states can be monitored through some code instrumentation. Accordingly, the success rate of tasks execution, i.e., their reliability, can be extracted by monitoring the system execution trace in controlled executions or in real production environment. Figures 5.20 and 5.21 illustrate both approaches.

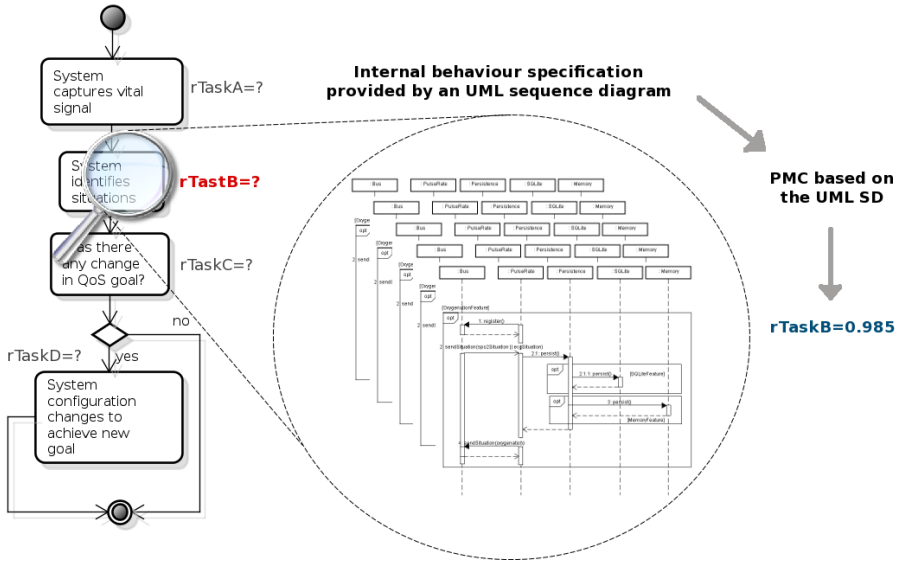



Figure 5.20: Individual reliabilities of leaf-tasks are estimated through PMC of internal behaviour specification.

As our proposal focus on the runtime reliability analysis for self-adaptive systems and monitoring is already part of a self-adaptation feedback loop, our evaluation with the MPERS case study is based on individual leaf-tasks reliabilities measured through instrumented code execution.

### 5.4.4 Specifying non-functional constraints

The specification of non-functional constraints is a sensible task that depends on both expertise and 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

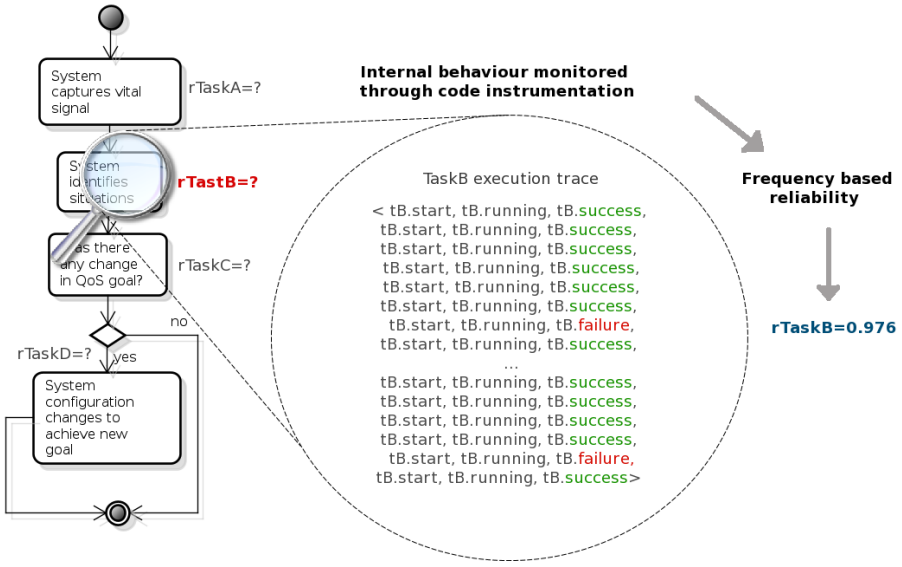



Figure 5.21: Individual reliabilities of leaf-tasks are estimated by monitoring their execution.

be achieved by any alternative solution and must be coherent to the system criticality - a minor failure consequence could be tolerable, but 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 like in service-oriented computing. Table 5.3 summarizes two possible non-functional constraints for the MPERS.

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

Table 5.3: 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 5.3 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 5.3, power consumption is (locally) associated to goal G2.

### 5.4.5 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. In this evaluation, we focus on the dependability attribute of reliability.

Given a DTMC model representing system activities, global reliability may be defined as the probability of reaching a final state where all system goals are achieved. This property is called reachability and can be specified using the PCTL language. For the MPERS root goal G1, reliability can be defined by: $P =?[G(noError)]$, with noError a boolean formula indicating that no task has transitioned to the failure state and P the steady-state probability of a true value for noError formula.

### 5.4.6 Reasoning with Ex-Tropos

The probabilistic verification of NFRs, performed as part of the Validation & Verification (VV) phase in RE, should anticipate violations of non-functional constraints. Treating a detected violation at design time may correspond to actions such as making a different choice for underlying technical components or social actors involved in the execution of tasks, optimizing the behaviour specification or even disposing a violating alternative as a means to satisfy its goal if there is at least one other valid alternative.

#### Solving variability

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 other models that supports 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 as parameters in a PRISM probabilistic model generates a single parametric formula for all products in the SPL for a given PCTL property. Individual product evaluation is achieved by the initialization of corresponding parameters in the formula.

In our proposal, PCM verification follows the same principle of the SPL family-based verification. A parametric DTMC model should be generated from the runtime goal

model. 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, as described in Section 5.4.

### Context selection

In a contextual goal model, or CGM, contexts may restrict which goals must be achieved, limit the adoptable alternatives (means) and also affect the non-functional metrics of individual tasks. These effects 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.

We considered two different approaches for the NFR verification of system with dynamic contexts of operation:

- *Deterministic context activation, or DCA*: a single context is activated by parameters in the model. In this approach, a parametric formula evaluates a given alternative for a specific context of operation. It is useful for the individual verification of context-alternative pairs, e.g., a context where battery is fully charged and all vital signs sensors are activated.

- *Probabilistic context selection, or PCA*: a probability distribution will define the likelihood of a context to be activated and the corresponding context implications in the verification model to be enabled. It is useful for the emulation of an approximate scenario in which the context of operation varies following some probabilist distribution. The resulting model evaluates an activated alternative for multiple contexts.

Both approaches are complementary as the first verifies the selected alternative for one context at a time and the former verifies a realistic scenario with multiple possible contexts. Table 5.4 summarizes each verification approach.

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 approach provides measures for the validation of self-adaptive systems against non-functional constraints like reliability.

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

| | Context selection | Alternative selection |
|---|---|---|
| **DCS** | Contexts are individually activated by parameters. | Alternative selection by the analyst is limited by the activated context. |
| **PCS** | Context selection follows a probabilistic distribution. | Verification model should select adoptable alternatives according to the activated context. |

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

patient location' is defined by the expression 0.9*rGPS + 0.1*0.7*rTRI, considering that GPS has priority over triangulation.

## 5.5 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.

# Chapter 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.

# Chapter 7

# Validation

# Chapter 8

# Conclusion

# Referências


[1] Raian Ali, Fabiano Dalpiaz, and Paolo Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15(4):439–458, July 2010. 3, 4, 9, 10

[2] Anthony Finkelstein Andrea and Andrea Savigni. A framework for requirements engineering for context-aware services. In *In Proc. of 1 st International Workshop From Software Requirements to Architectures (STRAW 01*, pages 200–1, 2001. 1, 9

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004. 1, 11, 12

[4] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. 3

[5] Luciano Baresi and Liliana Pasquale. Adaptive Goals for Self-Adaptive Service Compositions. In *2010 IEEE International Conference on Web Services*, pages 353–360. IEEE, July 2010. 2

[6] Luciano Baresi, Liliana Pasquale, and Paola Spoletini. Fuzzy Goals for Requirements-Driven Adaptation. In *2010 18th IEEE International Requirements Engineering Conference*, pages 125–134. IEEE, September 2010. 2

[7] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004. 2, 4, 7, 8, 23

[8] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos. Runtime goal models: Keynote. In *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, pages 1–11, May 2013. 2, 4, 28

[9] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1):3–50, 1993. 4, 7

[10] Jennifer Horkoff and Eric Yu. Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requirements Engineering*, 18(3):199–222, 2013. 10

[11] Oxford University Computing Laboratory. PRISM case studies. `http://www.prismmodelchecker.org/casestudies/`, 2015. [Online; accessed 25-january-2015]. 13

[12] PHILIPS® LIFETIME. GoSafe MPERS. http://www.lifelinesys.com/content/lifeline-products/get-life-gosafe/, 2015. [Online; accessed 25-january-2015]. 5

[13] K. Lorincz, D.J. Malan, T.R.F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton. Sensor Networks for Emergency Response: Challenges and Opportunities. *IEEE Pervasive Computing*, 3(4):16–23, October 2004. 5

[14] Danilo F. Mendonça, Raian Ali, and Genaína N. Rodrigues. Modelling and analysing contextual failures for dependability requirements. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 55–64, New York, NY, USA, 2014. ACM. 20

[15] V. Nunes, P. Fernandes, V. Alves, and G. Rodrigues. Variability management of reliability models in software product lines: An expressiveness and scalability analysis. In *Software Components Architectures and Reuse (SBCARS), 2012 Sixth Brazilian Symposium on*, pages 51–60, Sept 2012. 3, 5

[16] Vítor E. Silva Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness requirements for adaptive systems. In *Proceeding of the 6th international symposium on Software engineering for adaptive and self-managing systems - SEAMS '11*, page 60, 2011. 2, 26

[17] Shan Tang, Xin Peng, Yijun Yu, and Wenyun Zhao. Goal-directed modeling of self-adaptive software architecture. In Terry Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 313–325. Springer Berlin Heidelberg, 2009. 2

[18] Eric Siu-Kwong Yu. Modelling strategic relationships for process reengineering. January 1996. 4, 7

[19] Yijun Yu, Alexei Lapouchnian, Sotirios Liaskos, John Mylopoulos, and JulioC.S.P. Leite. From goals to high-variability software design. In Aijun An, Stan Matwin, ZbigniewW. Raś, and Dominik Ślęzak, editors, *Foundations of Intelligent Systems*, volume 4994 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2008. 2, 4, 10