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

Dependability Verification for Contextual/Runtime Goal Modelling

Danilo F. Mendonça

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. 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 M. de Melo

Banca examinadora composta por:

Prof. Dr. Genaína Nunes Rodrigues (Orientador) — CIC/UnB
Prof. Dr. Célia Ghedini Ralha — CIC/UnB
Prof. Dr. Luciano Baresi — Politecnico di Milano

CIP — Catalogação Internacional na Publicação

Mendonça, Danilo F..

Dependability Verification for Contextual/Runtime Goal Modelling / Danilo F. Mendonça. Brasília : UnB, 2015.
89 p. : il. ; 29,5 cm.

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

1. Dependabilidade, 2. confiabilidade, 3. engenharia de requisitos orientada a objetivos, 4. contextos, 5. verificação de modelo probabilística

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

Dependability Verification for Contextual/Runtime Goal Modelling

Danilo F. Mendonça

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Genaína Nunes Rodrigues (Orientador)
CIC/UnB

Prof. Dr. Célia Ghedini Ralha
CIC/UnB

Prof. Dr. Luciano Baresi
Politecnico di Milano

Prof.ª Dr.ª Alba Cristina M. de Melo
Coordenadora do Mestrado em Informática

Brasília, 15 de fevereiro de 2015

Resumo

Um contexto de operação estático não é a realidade para muitos sistemas de software atualmente. Variações de contextos impõe novos desafios ao desenvolvimento de sistemas seguros, o que inclui a ativação de falhas apenas em contextos específicos de operação. A engenharia de requisitos orientada a objetivos (GORE) explicita o 'por quê' dos requisitos de um sistema, isto é, a intencionalidade por trás de objetivos do sistema e os meios de se atingi-los. Um Runtime goal model (RGM) adiciona especificação de comportamento ao modelo de objetivos convencional, enquanto um Contextual goal model (CGM) especifica efeitos de contextos sobre objetivos, meios e métricas de qualidade. Visando uma verificação formal da dependabilidade de um Contextual-Runtime goal model (CRGM), nesse trabalho é proposta uma nova abordagem para a análise de dependabilidade orientada a objetivos baseada na técnica de verificação probabilística de modelos. Em particular, são definidas regras para a transformação de um CRGM para um modelo cadeia de Makov de tempo discreto (DTMC) com o qual se possa verificar a confiabilidade de se satisfazer um ou mais objetivos do sistema. Adicionalmente, para diminuir o esforço de análise e aumentar a usabilidade de nossa proposta, um gerador automatizado de código CRGM para DTMC foi implementado e integrado com sucesso à ferramenta gráfica que dá suporte às fases de modelagem e análise de objetivos da metodologia TROPOS. A verificação contextual de dependabilidade resultante reflete os requisitos no CRGM, que podem representar: o projeto de um sistema, cuja verificação ocorreria em fase de projetos; ou um sistema em execução, cujo comportamento pode ser verificado em tempo de execução como parte de uma análise de auto-adaptação com foco em dependabilidade.

Palavras-chave: Dependabilidade, confiabilidade, engenharia de requisitos orientada a objetivos, contextos, verificação de modelo probabilística

Abstract

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 requirements engineering (GORE) brings forward the 'why' of system requirements, i.e., the intentionality behind system goals and the means to meet then. A runtime goal model adds a behaviour specification layer to a conventional design goal model, and a contextual goal model specifies the context effects over system goals, means and qualitative metrics. In order to formally verify the dependability of a CRGM, we propose a new goal-oriented dependability analysis based on the probabilistic model checking technique. In particular, we define rules for the transformation of a CRGM into a DTMC model that can be verified for the reliability of the fulfilment of one or more system goals. Also, to mitigate the analysis overhead and increase the usability of our proposal, we have successfully implemented and integrated a CRGM to DTMC code generator to the graphical tool that supports the goal modelling and analysis phases of the TROPOS development methodology. The resulting contextual dependability verification reflects the system requirements in a CRGM, which may represent: a system-to-be, whose verification would take place at design-time; or a running system, whose behaviour can be verified at runtime as part of a self-adaptation analysis targeting dependability.

Keywords: Dependability, reliability, goal-oriented requirements engineering, context-awareness, probabilistic model checking

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

GORE [34] ganhou a atenção de profissionais acadêmicos e industriais devido à sua habilidade de sistematicamente modelar a intencionalidade por trás dos requisitos de sistemas. Mais do que apenas apresentar 'o quê' e 'como', modelos de objetivos também expressam o 'por quê' da existência de diferentes requisitos. Sua notação gráfica simples permite com que partes interessadas não técnicas participem do processo de análise e tenham uma visão clara do sistema. Ademais, a verificação automatizada de modelos deve evitar inconsistências da especificação de requisitos orientada a objetivos.

TROPOS [7] é uma metodologia GORE que também inclui fases de arquitetura e de detalhes de projeto do desenvolvimento de sistema sócio-técnicos e multi-agentes. Sistemas sócio-técnicos proveem e controlam uma vasta gama de serviços usados diariamente. Frequentemente esses serviços são responsáveis por requisitos críticos cujas falhas causariam consequências indesejáveis ou mesmo catastróficas. Isso requer que analistas e desenvolvedores considerem a dependabilidade como um requisito de primeira ordem.

Entre as diferentes causas que levam um sistema a falhar, algumas podem ser rastreadas a decisões de projeto em fases iniciais do processo de desenvolvimento, enquanto outras são causadas por variações no contexto de operação. Contextos dinâmicos aumentam a complexidade do processo de desenvolvimento, visto que essas variações podem alterar quais objetivos do sistema devem ser alcançados e quais meios estão disponíveis para alcançá-los. Além disso, algumas falhas só são ativadas em contextos específicos, fazendo emergir uma nova ameaça ao desenvolvimento de sistemas confiáveis.

Em TROPOS, assim como em outros frameworks GORE, não há uma abordagem específica para a verificação de atributos de dependabilidade e outros requisitos não-funcionais. A *Análise de contribuição* é usada para a comparação e seleção de alternativas de projetos de soluções com base na contribuição de cada alternativa para um ou mais objetivos do sistema, usualmente objetivos de qualidade do tipo *softgoals* [37]. Essa abordagem, contudo, não é otimizada para métricas dependentes do comportamento do

sistema ou de técnicas de análise mais complexas.

Num trabalho anterior, o CGM [1] foi estendido para lidar com os efeitos das variações de contextos sobre atributos de dependabilidade através de regras declarativas definidas em lógica difusa [26]. A partir de nossa experiência, concluímos que uma abordagem mais escalável e precisa para a verificação de métricas de dependabilidade em contextos dinâmicos era necessária. A verificação probabilística de modelos (PMC), técnica já explorada na verificação de dependabilidade no contexto de linhas de produtos de software [28], foi considerada como método formal para uma análise de dependabilidade orientada a objetivos.

Para o melhor de nosso conhecimento, a PMC ainda não foi usada como parte de uma análise de dependabilidade orientada a objetivos, onde a natureza distribuída de sistemas de software é levada em consideração, assim como o impacto causado por variações em seu ambiente de operação. Em específico, o atributo de confiabilidade é o foco dessa proposta e o modelo de verificação resultante deve verificar métricas relacionadas à confiabilidade como parte de análises em tempo de projeto e automatizada em tempo de execução. Finalmente, o efeito de contextos sobre objetivos, meios e métricas de qualidade, tal qual descrito pelo CGM, devem também ser considerados pela análise.

1.1 Definição do Problema

Um processo de engenharia de requisitos sistemático visa, entre outros, aumentar a qualidade do sistema entregue. Entretanto, poucas metodologias e frameworks investigam a conformidade do projeto do sistema a objetivos e métricas de qualidade relacionados a falhas do sistema e a sua segurança, nem provem meios adequados para o monitoramento e análise dessas métricas como parte de um ciclo de auto-adaptação. A despeito da robustez de arquitetura auto-adaptativas propostas na literatura [16], uma análise simplista da dependabilidade de sistemas pode resultar em falhas severas ou catastróficas.

Modelos de objetivos não são restritos aos objetivos estratégicos de mais alto nível. Por meio de decomposições do tipo E/OU, objetivos são refinados, delegados ou finalmente operacionalizados por tarefas. Portanto, tarefas podem ser mapeadas em atividades que compõe o comportamento do sistema. Dalpiaz et al. [9] propuseram uma mudança de paradigma para a engenharia de requisitos orientada a objetivos. Em vez de um modelo de objetivos estático visando apenas o projeto, uma linguagem regular é usada para a especificação de comportamento de objetivos e tarefas compondo um RGM. O RGM possibilita a verificação da conformidade da execução do sistema a seus objetivos por meio do monitoramento da instância de objetivos e tarefas e sua comparação ao RGM correspondente.

A despeito da contribuição para a verificação da conformidade em tempo de execução, a estimativa detalhada das taxas de sucesso e falha sobre janelas temporais não são ainda suportadas pelo framework RGM. Consequentemente, ele não provê os meios para se verificar se um sistema compre o nível esperado de dependabilidade. Adicionalmente, RGM se baseia em medidas sobre a execução passada tais quais 'o percentual de sucesso para um dado objetivo durante o último mês' e 'a tendência de falhas de um objetivo na última semana'. Devido a essa limitação, o RGM original não é apropriada para a auto-adaptação proativa em que sistemas devem evitar violações estimando a probabilidade de ocorrência de falhas futuras. Dada a importância da dependabilidade para sistemas, a proposta original do RGM precisa ser estendida para que a satisfação de objetivos em tempo de execução seja propriamente analisada.

A verificação de modelos é uma técnica formal de verificação que permite com que propriedades de comportamento de um determinado sistema sejam automaticamente verificadas com base num modelo do sistema e através da inspeção sistemática de todos os estados desse modelo [3]. A maior vantagem da verificação de modelos é prover, dado um modelo do sistema e uma ou mais propriedades, automação na verificação, em oposição à prova de teoremas, além de habilitar consultas complexas a respeito da corretude do modelo do sistema e sua conformidade em satisfazer ambos os requisitos funcionais e não funcionais.

Em específico, o PMC tem sido amplamente explorado e suportado por ferramentas tais quais o verificador de modelos PRISM. Contanto que o modelo de verificação construído a partir de uma especificação de comportamento seja precisa, esse método provê uma estimativa precisa para métricas como aquelas relacionadas à dependabilide. Por exemplo, PMC pode estimar a probabilidade de um sistema em alcançar seu estado final de sucesso num modelo DTMC baseado na confiabilidade dos componentes envolvidos na execução, o que define a confiabilidade global do sistema ou da atividade parcial analisada [3, 31].

Assumindo os benefícios do GORE e a especificação de comportamento provida por um RGM, esse trabalho investiga a viabilidade de uma abordagem PMC orientada a objetivos em que as tarefas folhas, que representam o comportamento de alto nível do sistema, são mapeadas num modelo de verificação probabilístico. O objetivo principal é prover ambas análises quantitativas e qualitativas para a satisfação de diferentes objetivos do sistema em tempo de projeto e de execução. Assim, nossa primeira questão de pesquisa é definida:

Questão de Pesquisa 1 (QP1): Dado um correto e consistente RGM para o qual seus objetivos são realizados ultimamente por um conjunto de tarefas folhas, é viável a análise da probabilidade de se atingir um ou mais objetivos do sistema através de uma técnica PMC?

Como descrita pelo CGM, a contextualização da informação obtida em tempo de projeto se torna imperativa uma vez que sua validade pode ser ameaçada por variações no ambiente de operação. Variações de contexto podem relativizar a necessidade de um objetivo, restringir a adoção de meios alternativos e também afetar a qualidade desses meios. Portanto, é desejável que a abordagem de verificação utilizada considere tais efeitos no resultado da análise. Contudo, dado o grande número de estados de contexto que podem afetar os requisitos e comportamento do sistema, a escalabilidade se torna a principal ameaça de uma análise contextual de dependabilidade. A partir disso é definida nossa segunda questão de pesquisa:

Questão de Pesquisa 2 (QP2): Dada análise de dependabilidade orientada a objetivos em QP1, é viável se considerar os efeitos da variação de contextos sobre quais objetivos são requisitados, quais alternativas são adotáveis e sobre a qualidade de cada alternativa?

Como discutido anteriormente, a auto-adaptação deve se basear em análise adequada com a complexidade e criticalidade das métricas sendo avaliadas. Confiabilidade é um importante atributo de dependabilidade e um requisito de primeira classe para a auto-adaptação, visto que define a continuidade do provimento de serviços corretos. Em modelos de objetivos, a resolução de variabilidade é baseada em entradas em tempo de execução e critérios não funcionais medidos ou estimados para cada alternativa [38]. Considerando o ciclo de alto-adaptação para a resolução de variabilidade, nossa terceira questão de pesquisa é definida como:

Questão de Pesquisa 3 (QP3): É viável e escalável empregar uma análise de dependabilidade orientada a objetivos baseada em PMC paramétrico como parte de um ciclo de auto-adaptação que resolva a variabilidade do modelo de objetivos em tempo de execução?

Um importante fator de sucesso para qualquer metodologia de software é o justificável aumento de esforço de desenvolvimento, que inclui conhecimento de domínio específico de linguagens e ferramentas utilizadas. PMC pode significantemente reduzir a ocorrência de falhas de sistemas, porém tal técnica requer conhecimentos adicionais que podem pesar

na decisão de se adotá-la. Portanto, uma geração automática do modelo de verificação para a técnica PMC a partir de um CRGM é desejável para se reduzir o esforço adicional sobre a análise de dependabilidade proposta. Em acordo, nossa quarta e última questão de pesquisa é definida como:

> **Questão de Pesquisa 4 (QP4):** É possível se gerar automaticamente o modelo probabilístico de verificação para a análise de dependabilidade orientada a objetivos definida em QP1 e QP2?

1.1.1 Requisitos para a Análise de Dependabilidade Orientada a Objetivos

Com base na lacuna identificada de uma verificação de dependabilidade orientada a objetivos, contextual e parametrizável, foram definidos os seguintes requisitos que devem ser endereçados pela proposta:

R.1 **Compatibilidade retroativa:** As notações de comportamento e de contexto estendendo o modelo de objetivos da metodologia TROPOS não deverá conflitar com a sintaxe e semântica originais.

R.2 **Escopo de verificação:** O escopo de verificação pode ser restrito a uma parte do sistema (objetivos locais) ou se estender a todo o sistema (objetivo raiz).

R.3 **Geração do modelo:** A geração de um modelo probabilístico representado as atividades de um RGM com restrições de contexto de um CGM devem ser automaticamente geradas a partir do CRGM criado em ambiente de modelagem e análise que dê suporte à metodologia TROPOS com extensão para notações de comportamento e de contextos.

R.4 **Integração de ferramentas:** Em específico, a ferramenta e plugin TAOM4E deverá ser estendida com sintaxes para anotações de comportamento e contexto e para a geração automática de modelo DTMC PRISM a partir de um modelo CRGM.

R.5 **Suporte à sintaxe de projeto:** O modelo de verificação deve ser coerente às decomposições E/OU de objetivos e tarefas e às relações objetivo-tarefa da sintaxe convencional de modelos de objetivos.

R.6 **Suporte à sintaxe de comportamento:** O modelo de verificação deve ser coerente à ordem de satisfação/execução de objetivos/tarefas, à cardinalidade e à satisfação/execução de objetivos/tarefas alternativas, opcionais e condicionais definidos pela sintaxe herdada de um RGM.

R.7 **Suporte à sintaxe de contextos:** O modelo de verificação deve ser coerente aos efeitos de variação de contextos sobre a ativação de objetivos, a adoção de subobjetivos e tarefas e sobre a métrica individual de qualidade de tarefas folhas.

1.2 Solução Proposta

Em contraste às abordagens anteriores para a análise de dependabilidade por meio de PMC, esse trabalho propõe a verificação que é diretamente mapeada a um RGM. Em vez de construir o modelo de verificação probabilístico a partir de modelos de comportamento tradicionais Unified Markup Language (UML), o presente trabalho se beneficia da sintaxe de um RGM que especifica também o comportamento de objetivos e tarefas para construir um modelo de alto nível DTMC para a verificação de confiabilidade de diferentes objetivos do sistema (QP1). Tal modelo, comparado a um diagrama de atividades UML de alto nível, deve também incluir os efeitos de contexto sobre objetivos, meios e métricas de qualidade como definido pelo CGM (QP2).

A ferramenta PRISM provê um rico ambiente de análise para PMC. No entanto, uma análise de auto-adaptação em tempo de execução deve ser automática, isto é, baseada em processos computáveis sem intervenção humana. O PMC paramétrico satisfaz esse requisito ao gerar uma fórmula paramétrica para um determinado modelo probabilístico e propriedade analisada com parâmetros no modelo no lugar de constantes [15]. Portanto, diferentes análises em tempo de execução podem ser feitas a partir da simples inicialização dos parâmetros da fórmula com valores, por exemplo, correspondentes a diferentes alternativas ou configurações. Em nossa proposta, investigamos a viabilidade de um PMC paramétrico como o método formal para a análise de dependabilidade orientada a objetivos em sistemas auto-adaptativos (QP3).

Finalmente, para reduzir o esforço e custo de análise e melhorar a usabilidade, a automaticidade e também reduzir a tendência ao erro da análise proposta, uma implementação em linguagem JAVA foi integrada à ferramenta TAOM4E [27]. Essa extensão permite a geração automática de um modelo DTMC em linguagem PRISM a partir de um CRGM (QP4).

1.3 Avaliação

A proposta foi avaliada com a aplicação de uma análise de confiabilidade orientada a objetivos ao desenvolvimento de um sistema Sistema Pessoal Móvel de Resposta a Emergências (MPERS). Tal sistema se enquadra no contexto de área de sensores do corpo humano [25]. Em particular, o MPERS é um sistema de resposta a emergências executado num dispositivo móvel que recebe informações de sinais vitais coletados pelos sensores. Ao invés de um ambiente estático, o MPERS é concebido para permitir com que pacientes com diferentes riscos de saúde possam preservar a sua mobilidade enquanto são monitorados e assistidos. Por ser um sistema móvel, o MPERS é afetado por variações de contexto e a auto-adaptação se torna um requisito mandatório para se otimizar recursos e se evitar falhas, sobretudo falhas catastróficas que colocariam a vida de usuários em risco.

1.4 Resumo das Contribuições

Essa seção resume as contribuições almejadas por essa proposta.

1. Verificação de dependabilidade para CRGMs.

   - Definição de regras de conversão entre diferentes tipos de decomposição e de regras de comportamento num RGM para um modelo DTMC.
   - Consideração explícita de contextos na modelagem e análise de dependabilidade, isto é, inclusão dos efeitos de contexto de um CGM no modelo DTMC.
   - Habilidades de consulta probabilística sobre métricas relacionadas à confiabilidade em se satisfazer um ou mais objetivos num CRGM.

2. A geração automatizada de um modelo DTMC em linguagem PRISM a partir de um CRGM.

   - Implementação de um parser para as anotações de comportamento especificadas pela linguagem regular num CRGM herdada da proposta original RGM.
   - Implementação de um parser para os efeitos de contexto especificados por expressões lógicas que podem ativar/restringir um ou mais objetivos do sistema e disponibilizar/restringir um ou mais tarefas herdados da proposta original CGM.

- Implementação JAVA de um gerador CRGM para DTMC em PRISM integrado à ferramenta TAOM4E que suporte a metodologia TROPOS conforme arquitetura para plugins da plataforma Eclipse.

1.5 Organização do Documento

Essa dissertação foi organizada como se segue. O capítulo 2 apresenta a base dos conceitos utilizados. O capítulor 3 descreve os mais relevantes trabalhos relacionados. O capítulo 4 detalha o sistema que motiva essa abordagem. O capítulo 5 descreve a proposta em si. O capítulo 6 descreve a implementação do gerador automático de um modelo DTMC a partir de um modelo CRGM. O capítulo 7 avalia a presente proposta com base no *Goal Question Metric framework*. Finalmente, o capítulo 8 conclui esse trabalho com as considerações finais sobre a proposta, assim como nossos trabalhos futuros.

Capítulo 2

Referencial Teórico

2.1 Goal-oriented Requirements Engineering

GORE captures the intentionality behind system requirements [34]. Through a directed graph tree that begins with a root goal, goals are connected through decomposition links. Higher level goals are usually 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 complete if all system goals are either decomposed, delegated to other actors or fulfilled by operational system tasks.

Goal-oriented frameworks and methodologies like KAOS [10], the i* [36] and TROPOS [7] represent the foundations for the goal model analysis used by a variety of other proposals. Despite some syntax differences, most goal-oriented approaches share a set of common and more important core concepts:

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

- **Goal:** actors' strategic interest. A goal with a clear-cut criteria for its satisfaction is called a hard goal. In opposition, softgoals have no clear-cut criteria for deciding whether they are satisfied or not and usually represent non-functional requirements. For example: 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. For example: monitor temperature sensor, persist vital signs data, request emergency assistance.

- **Resource:** a non intentional entity data or physical resource that is generated or required by an actor. For example: a form input from the user, an exported file, the power from the battery component, etc.

- **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).

- **Social dependency:** 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 to the i* framework 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 [11]. Contexts, as already stated, may not be static, but dynamic, and a system has no control over the context in which it operates. Accordingly, a system must be able to support different contexts of operation without violating its functional and non-functional goals. To achieve this, it must be able to monitor the state of their surrounding environment and take adaptive actions regarding the alternatives used for fulfilling their goals.

In a CGM, dynamic contexts may affect what goals a system has 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 [11]. 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' is only required in the context 'patient is not 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.

Context effects may have different causes, among them:

- **Stakeholders preferences:** At a certain context, a stakeholder need (dependency) that justifies a system goal may cease to exist (goal restriction); or it may prefer a given alternative to another (means restriction).

- **Technical impediments:** At a certain context, a required information or physical resource may not be available, imposing a restriction on the selection of one or more alternatives (means restriction).

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 means-end task should be executed to satisfy its upper goal or which subtask should be executed to satisfy its upper task. 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 has different meanings according to the development phase it takes place.

2.4.1 Design-time analysis

At design time, multiple alternatives are elicited through 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 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 using a satisfaction analysis technique [18].

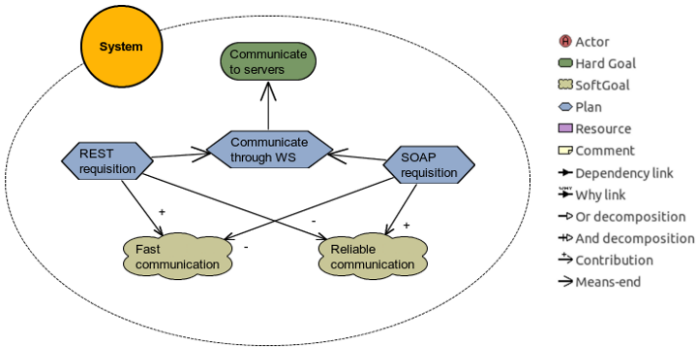


Figura 2.1: Contribution analysis in TROPOS GORE.

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 [38]. Runtime analysis should 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 health condition is 'critical', an emergency chopper is selected for assistance in the place of an ambulance following the context rules in a CGM. We refer to this context effect as 'direct context implication'.

In other cases, alternatives should be monitored and analysed in terms of non-functional metrics to decide which one is suitable for selection. Here, the context of operation does not directly defines the adoptable alternative, but it affects the both the quality of each alternative and the quality constraint that must be achieved. Thus, context indirectly define the more suitable or the valid alternatives. For instance, a more sophisticated analysis must estimate the probability of the ambulance to reach the patient in less than X time units given a traffic condition and the patient health condition. We call this 'indirect context implication'.

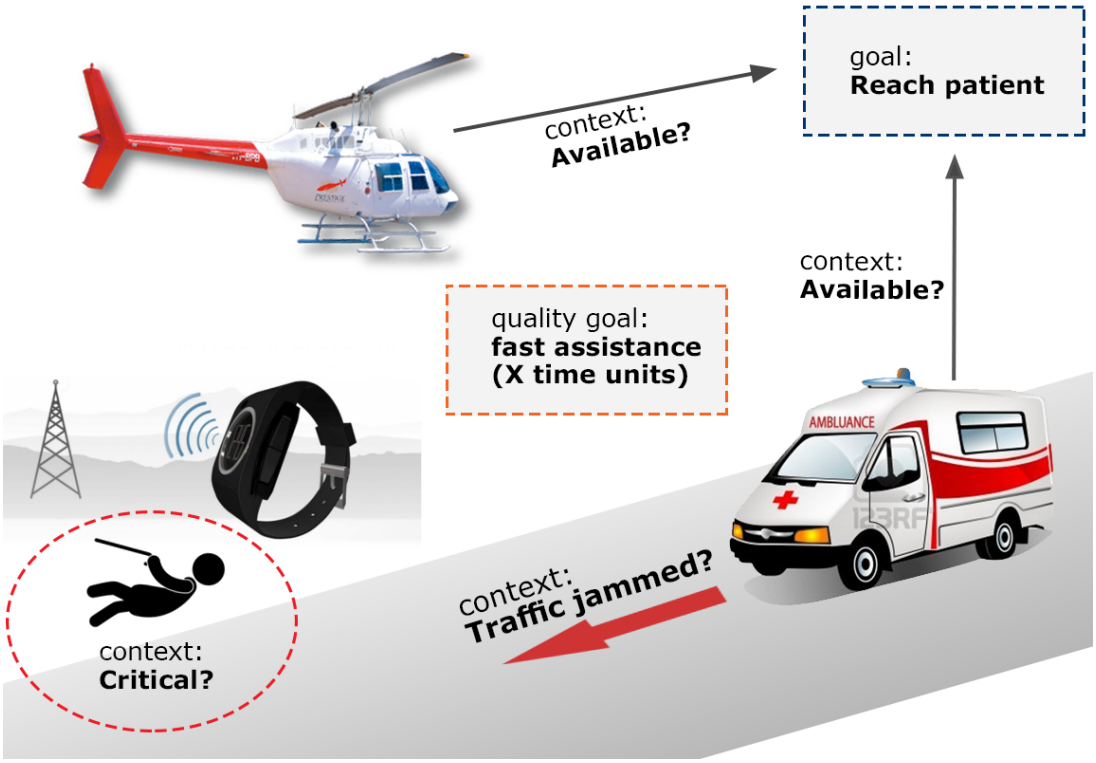


Figura 2.2: Contexts affecting the requirements and the means for an emergency response system.

In Figure 2.2, the availability of the emergency resources are context facts that directly restrict the adoption of corresponding transport means for an emergency team to reach a patient. In contrast, the traffic condition affects the time to reach the patient by ambulance, 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 constraint in the current context of operation?

2.5 Dependability

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 a certain threshold [2]. According to Avizienis et al., dependability encompasses the following attributes:

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

A 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 fail to describe the system: either the goals or the means to fulfil them 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 [2].

Avezienis et al. distinguishes faults from errors and failures in a fault causality chain. In their definition, errors are part of the system's total state that may lead to failures. Failures are characterized by deviations in the external service state, i.e., by external errors. In turn, errors are caused by faults, resulting in the following chain:

$$fault \rightarrow error \rightarrow failure$$

In most cases, a fault first causes an error in an internal state of a system. Not all internal deviations results in external deviations, i.e., not all internal errors result in failures. External faults may also cause internal errors and possibly subsequent service failure(s), but only if a prior vulnerability exists, i.e., a previous internal fault enabling the external fault to harm the system.

Failures can be characterized by different viewpoints. In this work, one important aspect is the failure consequence level, as it describes the consequence of failures to user(s) and to the environment. Two limiting levels are defined by Avezienis et al.:

- **Minor consequence:** where the harm caused by a failure is not higher than the benefit of the correct system behaviour. For example, a momentary interruption in a video streaming service.

- **Major consequence:** where the failure harm is incommensurably higher than the benefit of the correct service. For example, the death of a user.

Other intermediary levels may exist according to each case. In a goal model, the consequence level viewpoint may characterize the failure in achieving one or more system goals. A failure in fulfilling a goal with higher relevance in the goal tree of a critical system, for instance, would have a catastrophic consequence. There are many means to attain systems dependability. Avezienis et al. groups them in four major categories:

- **Fault prevention:** means to prevent the occurrence or introduction of faults.

- **Fault tolerance:** means to avoid service failures as a consequence of faults.

- **Fault removal:** means to reduce the number and severity of faults.

- **Fault forecasting:** means to estimate the present number, the future incidence, and the likely consequence of faults.

The scope of this work is restricted to specification violations, i.e., we assume that a system specification is complete and consistent and failures are caused by 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 goal-oriented dependability analysis is classified as a fault forecasting, as it aims to estimate the probability of different system goals in being fulfilled.

2.6 Probabilistic Model Checking

Many systems are susceptible to various phenomena of stochastic nature and to non determinism in their behaviour. For example, failures may be caused by unpredictable events and by unreliable components. In contrast to model checking techniques for which the absolute correctness of a system is verified, probabilistic model checking aims to verify properties over transitions systems enriched with probabilities [3]. Accordingly, PMC allows quantitative statements to be made about the system behaviour, expressed as probabilities or expectations, in addition to the qualitative statements made by conventional model checking [20].

Among the most popular types of transition systems employed in PMC are those based on Markov chains, e.g., the DTMC and the Markov decision process (MDP) [20]. Also, probabilistic operators extend the conventional time-bounded or unbounded temporal logics for property specification. Regarding dependability, the PMC technique enables the forecasting of systems performance and dependability based on probabilistic events and behaviour described in probabilistic models. As a model checking technique, PMC requires:

1. a description of the system to be analysed, typically given in some high-level modelling language, e.g., in DTMC.

2. a formal specification of quantitative properties of the system that are to be analysed, usually expressed in variants of temporal logic, e.g., in Probabilistic Computation Tree Logic (PCTL) [13].

In PMC, the system description is converted to a probabilistic model. In addition to the quantitative information regarding the probability and/or timing of the transition's occurrence, Markov chains can also be augmented with *rewards* used to specify additional quantitative measures of interest [19].

2.6.1 PRISM tool

The PMC technique used in this approach is supported by the PRISM probabilistic model checker tool [23]. The decision of using PRISM as the probabilistic state-based model checker was due to the richness of its environment and to the number of successful case studies that have used this tool, indicating its maturity [21].

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 property specification are also available in the tool.

2.6.2 PRISM language

PRISM language [22] 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. They are composed of variables and commands. The first describes the finite states a module can be in. The later describes 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:

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

2.6.3 Probabilistic Computation Tree Logic

PCTL is a temporal logic based on the Computation Tree Logic (CTL). Its main difference from CTL is the probabilistic operator $P_J(\varphi)$, where φ is a path formula and J an interval in [0,1] indicating a lower and/or upper bound on the probability. $P_J(\varphi)$ may be read as the probability of a set of paths satisfying φ and starting at state s to meet the bounds given by J [17].

The specification of domain-specific dependability properties with PCTL has been explored in previous works [19, 20, 31]. For example, the reachability property expressed by the Probabilistic existence PCTL formula $P =?\ [\ F\ (\varphi)\ ]$ computes the probability that a system will eventually reach a state that satisfies φ [13]. Accordingly, the satisfaction of this formula guarantees that a final and successful system state will be reached regardless of the time elapsed to reach it. A time-bounded variant would express a similar event in a restricted number of transitions or time units.

In our proposal, PCTL formulas may be specified to verify different properties of a runtime goal model mapped to a DTMC verification model. In specific, Análise de Dependabilidade Orientada a Objetivos (GODA) focus in the time-unbounded reachability of the states representing the fulfilment of one or more system goals.

2.6.4 PARAM Tool

The powerful analysis environment offered by PRISM tool is limited by the verification of a single combination of initialized variables at a time. For instance, if a variable

represents the probability of a transition in the model, PRISM requires the initialization of this value to produce a fixed output for a given specified property. At most, PRISM allows the creation of experiments with undefined variables in the model ranging from two limits at a fixed interval value.

A parametric model checking provides a more flexible analysis, as constant variables in the model can be replaced by parameters [15]. Regarding the PMC, the PARAM tool extends the PRISM language with the additional reserved word *param* to be used with variables describing state transition probabilities. Given a probabilistic model with additional param variables and a PCTL property, a corresponding parametric formula is generated.

The main benefit of the parametric formula generated by PARAM is to enable the verification of multiple combinations of values for each parameter in an efficient manner, as the probabilistic model checking problem has been previously solved by the tool. Also, different sorts of postprocessing operations can be performed by computer algebra packages, e.g., to find the optimal parameter settings and to evaluate the parametric formula for a given setting [15].

In our proposal, a parametric formula generated offline could be evaluated at runtime and integrate a self-adaptation loop. The formula scalability is a relevant concern, as previous works have demonstrated its exponential relation with the number of parameters in the model [28]. Nonetheless, depending on the modelling approach and the scope of the verification, parametric model checking can be proven an efficient approach for dependability analysis. The scalability of our goal-oriented dependability analysis based on parametric PMC is addressed by our third research question.

More recent PRISM versions also supports a parametric model checking. Nonetheless, PARAM has been successfully evaluated in more case studies in which it has been proved a reliable and stable tool, justifying its adoption by this work in detriment of the built-in PRISM parametric analysis.

2.7 ANTLR Language Recognition Tool

ANTLR (Another Tool for Language Recognition) is an open source parser generator for reading, processing, executing or translating structured text or binary files [30]. The main purpose is to automatically generate a parser for a custom language defined in a specific grammar supported by the tool. The parser can then be imported by any JAVA compatible project to build and walk parsed trees from an input stream.

As a result, domain-specific languages may be 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 in a runtime goal model and for the context effect formulas in a contextual goal model.

Capítulo 3

Trabalhos Relacionados

3.1 Goal-oriented Modelling and Analysis

Goal models have been used in requirements engineering (RE) to elicit, model and analyse stakeholders requirements. The first proposal regarding goal orientation, namely the KAOS framework, dates back more than 20 years [10]. The majority of the goal-oriented frameworks and methodologies in the literature share a common set of conceptual elements with variations on the syntax/notation and on the development phase they applies to.

KAOS is a goal-directed requirements acquisition, including early and late requirements of the software development process. It is a knowledge-based system for acquiring the conceptual structure, since requirements acquisition is driven by such higher-level concepts. KAOS framework focuses on the identification of functions that a system-to-be needs to implement to fulfil stakeholders goals. It was the first goal-oriented framework to employ AND/OR *operationalization* links to relate goals to the operations which ensure them [10, 35]. Moreover, KAOS employs responsibility links to relate goals to agent submodels.

The NFR (non-functional requirements) framework [8] aims to represent user intentions in technical systems vis softgoals, i.e., goals whose satisfaction has no clear-cut criteria. AND/OR goal decomposition was also employed and the NFR framework introduced the contribution links representing potentially partial negative and positive contributions to and from softgoals.

The i* is a modelling strategic relationships for process reengineering proposed by proposed by Yu et al. [36, 37]. It inherits the softgoals concept from the NFR framework and includes hard goals with clear-cut satisfaction criteria. In contrast to KAOS, the i* framework adds an early requirements phase in which the organization environment of the system-to-be is modelled and analysed. Agents dependency links define situations

where an agent depends on another for a goal to be achieved, a task to be executed or a resource to become available.

The TROPOS methodology [7], as described in Section 2.1, extends i* with later phases of software life-cycle. In specific, TROPOS focuses on the development of socio-technical systems based on the agent-oriented software architecture. The decision of using TROPOS as the GORE in this work was not because of any particularity in its goal model, but mainly because of its broader scope in the software development life-cycle, including the detailed design phase in which agent behaviour are further specified. This specification may be the input for a design-time dependability analysis based on PMC, which will be covered in Chapter 6. Besides the broader scope, the TAOM4E open source tool supporting TROPOS is in a stable version and has good usability. This tool has become the basis for the implementation of an automatic transformation of a CRGM into a DTMC verification model. More details on different goal-oriented frameworks/methodologies can be found elsewhere [34].

3.2 Contextual Goal Model

The CGM [1] proposes the contextualization of the intentional elements and relations in a goal model. From the activation of a root goal to the adoptability of tasks, CGM defines different context implications that may affect the problem tackled by a system and the validity and quality of possible solutions. CGM uses a graphical notation to associate context to their affected elements. In specific, context can be associated to root goals (activation), OR-decompositions (adoptability), means-end (adoptability), dependencies (activation), AND-decomposition (activation), contribution to softgoals (quality).

The main benefits of the CGM are twofold. First, CGM enriches the conventional goal model with the notation for the contextualization of intentional elements and relations. Second, it provides the modelling constructs to analyse and discover relevant information the system needs to capture in order to verify if a context applies, i.e., the rationale for context monitoring. This last contribution is useful for runtime monitoring and analysis of a self-adaptive system reflecting stakeholder's rationale and the environment in which the system operates [1].

CGM provides a more realistic and precise contribution analysis contextualized by environment conditions, but does not change the nature of the GORE contribution analysis. In contrast to CGM, our work emphasizes the formal verification of non-functional requirements in place of the more subjective contribution analysis based on the direct evaluation of the forward impact between alternatives and softgoals. However, our work has benefited from the CGM conceptual model where our goal-oriented dependability analysis takes

into account the context effects from a CGM over the requirements to be met and the adoptable means in the corresponding probabilistic verification model.

3.3 Runtime Goal Model

Despite the use of goal models in the support of runtime monitoring and adaptation in many works, Dalpiaz et al. argued that these proposals 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, namely a Design goal model (DGM), and the Runtime Goal Model (RGM) that extends DGM with additional state, behavioural and historical information about the fulfilment of goals [9].

The main purpose of the RGM approach is to provide a behaviour specification for the fulfilment of goals and the execution of tasks. RGM defines a class model, while the Instance goal model (IGM) captures instance states of monitored goals and tasks that must conform to their class model, the RGM. If an IGM violates its RGM, then a corrective action is expected to take place.

The IGM representation is built from algorithms that parses the execution traces of the instrumented implementation of a running system. The contribution, however, is restricted to the runtime regex and to the IGM algorithms. Other research questions concerning the percentage of success/failures for a given goal with or without temporal frames is not addressed by the RGM framework [9].

Our proposed goal-oriented dependability analysis has benefited from the RGM as the PMC technique requires a system behaviour specification and the RGM provides a high-level description of a system behaviour. In our work, RGM leaf-tasks are mapped to a probabilistic verification model in PRISM language preserving its behaviour semantics. Our proposal aims to answer qualitative and quantitative questions about the time-unbounded success/failure probability in fulfilling different system goals.

Table 3.1 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 4.2. A formal and detailed description can be found in [9].

3.4 Dependability Contextual Goal Model

The work presented in [26] by another proposal concerning GORE, dependability analysis and dynamic contexts, namely the Dependability Contextual Goal Model (DCGM). The contribution was focused on the context effect over both dependability requirements and dependability attributes based on declarative fuzzy logic rules.

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.	$try(T9.0)?\ skip : T9.1$

Tabela 3.1: Description of RGM behaviour rules used by the proposal.

In DCGM, a failure classification scheme was used to classify the consequence level and domain of failures in achieving system goals. This process leads 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 (CDR). These requirements inherited the same form of the AwReq [32], but instead of being static, CDRs are associated to context conditions. Another DCGM characteristic is the contextual failure implication (CFI), which consisted of a dependability domain-specific contribution analysis supported by fuzzy logic to define IF-THEN rules between context conditions and the corresponding level of a dependability attribute of a given alternative, e.g., the reliability of a task in a context condition.

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 manual analysis task. 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, leads to the idea of coupling a more precise and reliable verification approach such as the PMC technique to a runtime goal

model. Still, the idea of a failure classification and the specification of dependability requirements as non-functional constraints have been also addressed in this proposal.

3.5 Awareness Requirements

Souza et al. [32] proposed the AwReq as a meta-requirements class in a goal model, i.e., AwReqs specify, among others, the success/failure rate for other 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 performance and to provide criteria for 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 does not provide an approach for analysis and validation before system implementation or through system 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, similarly to the DCGM [26]. In contrast, our approach tackles the verification of context-specific dependability meta-requirements through probabilistic model checking technique that can be performed at design-time to support alternative design decision or at runtime as part of a self-adaptation loop analysis.

3.6 Formal TROPOS

The idea behind the formalization of a goal model, as proposed by Formal TROPOS (FTROPOS) [12], is to provide a formal specification of sufficient and necessary conditions to create and achieve intentional elements like goals, tasks and dependencies in a goal model and also 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, FTROPOS uses a first-order linear-time temporal logic as a specification language.

The nature of the verification proposed by FTROPOS is different from the PMC used by our work. FTROPOS aims to provide the information required for a consistency verification of the goal model. The verification is performed not only in respect to the conventional goal model syntax of intentional elements and relations, but also to domain specific information about 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 automatically verified by a model checker tool.

In contrast to FTROPOS, our proposal uses a probabilistic model checking technique for the verification of dependability properties. Despite similarities in the behaviour specification, FTROPOS focuses on consistency verification and not on the quantitative analysis of the system performance and dependability. Therefore, both aproaches are complementary.

Capítulo 4

Sistema Pessoal Móvel de Resposta a Emergências

This chapter covers some relevant aspects of MPERS that motivates the use of a goal-oriented dependability analysis. The MPERS system-to-be as well as its organizational and operational environments are modelled and analysed with the conventional requirements phases in TROPOS methodology and by the additional context/runtime modelling and analysis phases as proposed by the CGM [1] and the RGM [9], respectively.

4.1 Introduction

An emergency response system is a mission-critical system. Thus, failures in achieving its goals by the time they are required may lead to catastrophic consequences on monitored individuals 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 and to attain dependability.

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 it must also be addressed as user's privacy may not be violated by disclosing his personal health or geolocation information 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.

Context is an important factor for MPERS. First, as a mobile system, indispensable resources may variate throughout operation, preventing a correct system behaviour or affecting its performance. Moreover, an attentive user with good health condition who

wants to be monitored differs from a person with severe health problems history or who has reached a specific age group of a higher health risk. Accordingly, the following context variables potentially affecting MPERS have been elicited:

- The battery of the mobile device;
- The battery of the 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;
- Health incidents and emergencies;

Later in this chapter, context formulas representing context effects on both goals and tasks are presented. The next sections describe each of the phases involved in the modelling and analysis of a CRGM for the MPERS.

4.2 TROPOS Requirements Engineering Phases

4.2.1 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. 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 4.1. System actors and social actors are displayed in different colors. Among the stakeholders, the emergency center represents a private or public organization interested in providing an emergency response service to patients. Patient and doctor represent, respectively, the assisted person and the medical responsible for defining and evolving the emergency detection rules as part of an evolutionary approach for personal emergency response. Finally, sensors retailer should provide the vital signs sensors required for monitoring.

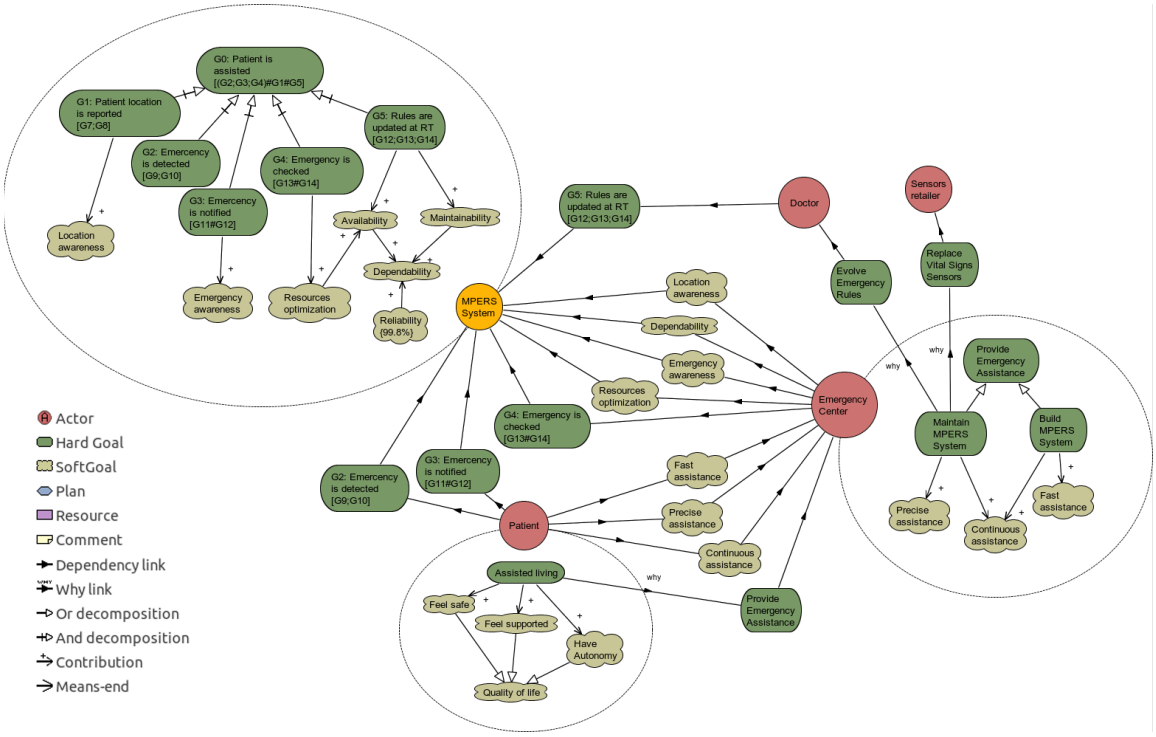


Figura 4.1: TROPOS mixed model for MPERS at early requirements phase

From the diagram in Figure 4.1 it is possible to have a first look 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 fifth 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 orange circle indicates that MPERS is a system actor. MPERS goals can be seen with an additional behaviour annotation indicating its dynamic behaviour as part of the runtime goal model as presented before in Section 3.3, Table 3.1. This annotation reflects the RGM phase presented later in this chapter, as the TAOM4E tool used for goal modelling shares unique entities and relations among different development phases.

4.2.2 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 decomposition specifies how leaf-goals can be fulfilled by tasks performed by the system actor. Figure 4.2 illustrates the late requirements diagram for the MPERS.

To evaluate our proposal, we have explored the PMC technique for the reliability analysis of a system goal model resulting from the TROPOS late requirements phase with additional behaviour and contextual specifications described in the next subsections. The idea is to initially evaluate the approach in a monolithic representation without the additional complexity of a multi-agent architecture. Hence, the evaluation involving later TROPOS phases should be explored in future work.

4.3 Runtime Goal Modelling

At this point, the system is modelled as a monolithic actor and its goal model may be extended with the behaviour specification to compose a RGM. This extended model merges multiple views in the same diagram: goals, tasks and relations represent the requirements view for the system-to-be as well as the intentionality behind then, while the runtime specification provides the behaviour specification required for our goal-oriented dependability analysis based on PMC.

The MPERS goal model in Figure 4.2 already presents the behaviour annotations composing a RGM. Goals and tasks have received an unique identifier (ID). Elements IDs are prefixed with an 'G', in the case of goals, and 'T', in the case of tasks. For each decomposed element, a corresponding behaviour annotation specifies the behaviour of all immediately underlying goals and tasks. Parenthesis are used for grouping related goals/tasks and for clarification purposes only, as temporal order is parsed from the type of the rule and its position in the behaviour annotation.

RGM behaviour rules have been described in Chapter 2. In this work, runtime annotations have been added to the name area of goals and tasks in the goal model diagram built with TAOM4E tool. They are enclosed by square brackets to enable their parsing. Future works should extend TAOM4E business model with a proper field for that purpose.

The following subsection compares the MPERS RGM presented in Figure 4.2 to an UML activity diagram for better understanding of the behaviour specified by a runtime goal model.

4.3.1 RGM - UML activity diagram comparison

A similar behaviour specification achieved with the RGM in Figure 4.2 could be provided by an UML activity diagram with activities representing the leaf-tasks in the 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 behaviour specification. This simple notation

increases the utility of a goal model diagram. Figure 4.3 presents an activity diagram corresponding to the MPERS leaf-tasks.

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, interleaved, 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 for the system-to-be.

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, 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 [9]. Depending on the complexity of the behaviour specification, a more robust runtime syntax would have to be employed or complemented by traditional UML behaviour models. In this work, RGM is used as input for our goal-oriented dependability analysis based on PMC.

4.4 Contextual Goal Modelling

At this phase, the analysis of the environment in which the system will operate is performed. A careful investigation of what is static and what may change during operation should list the contexts facts and variables that may potentially affect goals, operational means (tasks) and the quality of different means (alternatives). In CGM, contexts are described by statements and facts. Statements must be supported by facts, while facts can be directly monitored by the system. In our work, we simplify the analysis by disregarding context statements. Facts are specified as boolean expressions composed of variables and operators. Variables can be either boolean or floating numbers. The following operators can be used in the modelling environment:

- <, <=, >=, > (relational operators)
- =, != (equality operators)
- ! (negation)
- & (conjunction)
- | (disjunction)

Table 4.1 presents the contexts affecting the MPERS system. Each context has an unique identifier (CID), a textual description and the corresponding boolean expression that later will be parsed into a PRISM language formula.

CID	Description	Boolean expression	A.E.
C1	User is at home	$HOME\_WIFI \ \& \ LOCATION\_AGE <= 10$	G6
!C1	User is out	$!HOME\_WIFI \ \mid \ LOCATION\_AGE > 10$	G7
C2	Mobile signal	$SERVICE \ != \ false$	T8.1
C3	WI-FI is on	$WIFI \ != \ false$	T6.0. T7.01
C4	GPS signal	$GPS \ != \ false$	T7.10
C5	Internet	$INTERNET \ != \ false$	T8.0, T12.0 T23.0, G5
C6	Disk space available	$DISK\_SPACE >= 5$	T17.1
C7	Low health risk	$HR = 1$	G13
C8	Moderate health risk	$HR = 2$	-
C9	Severe health risk	$HR = 3$	-
C10	Emergency	$EMERGENCY \ != \ false$	G5

Tabela 4.1: Contexts affecting the MPERS system.

Column A.E. (Affected Element) indicates one or more goals/tasks that are activated/-restricted by the context. Isolated boolean facts have been explicitly specified using the *false* term just for clarification. In order to specify the context effects in the TAOM4E modelling environment, we have applied logic predicates for each affected element, as exemplified by Figure 4.4.

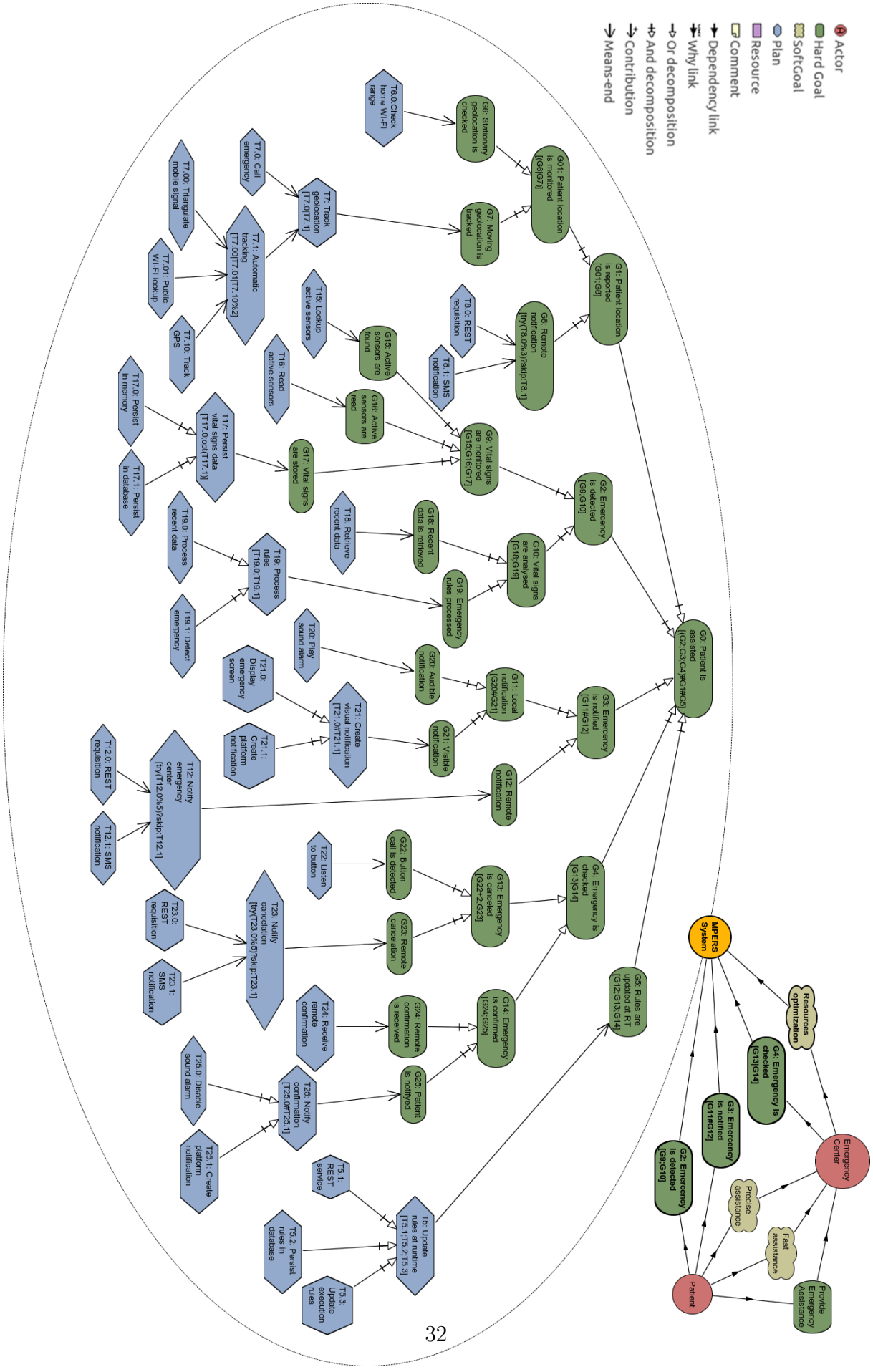


Figura 4.2: TROPOS mixed diagram for MPERS at late requirements phase

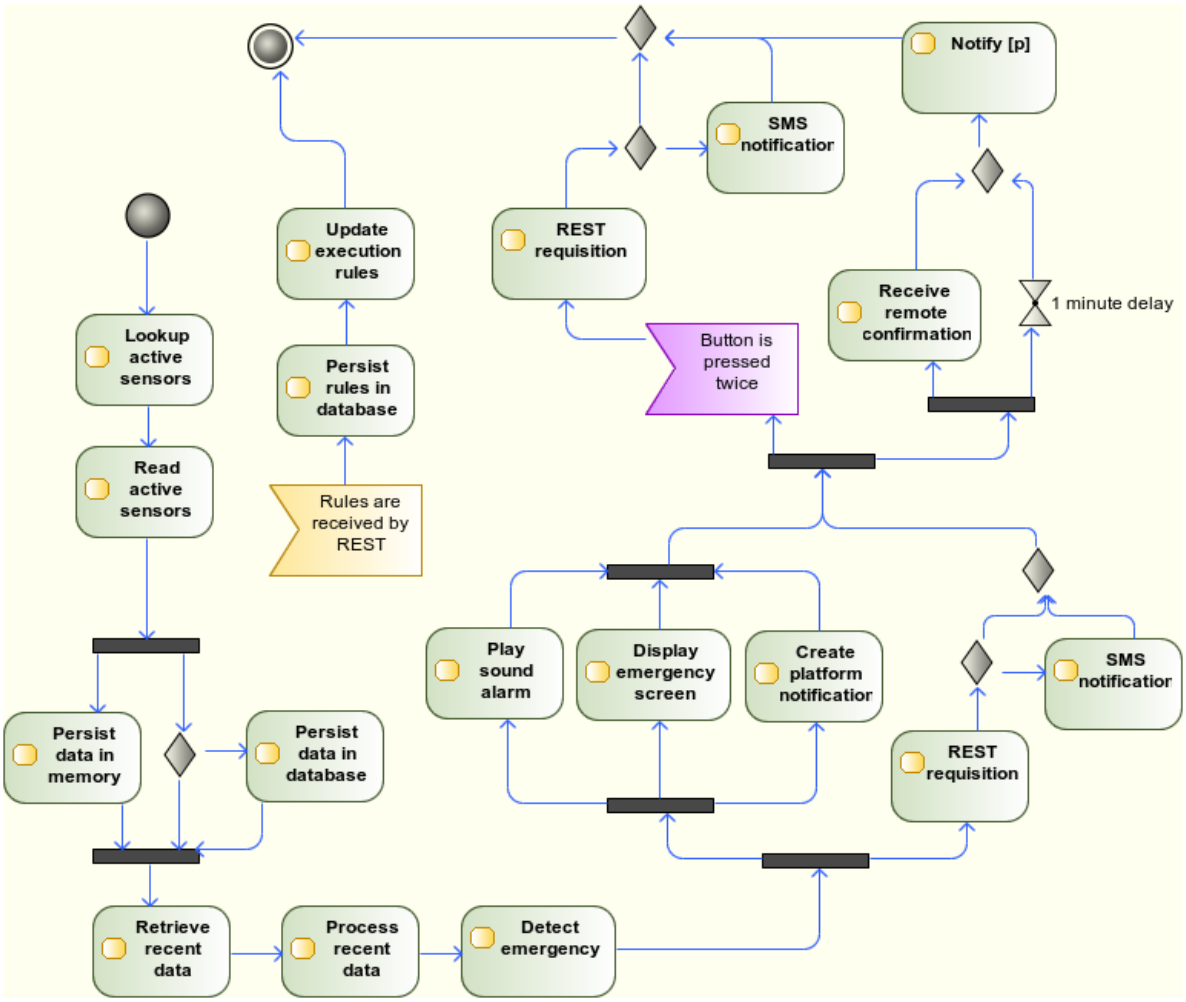


Figura 4.3: MPERS tasks represented by an UML activity diagram

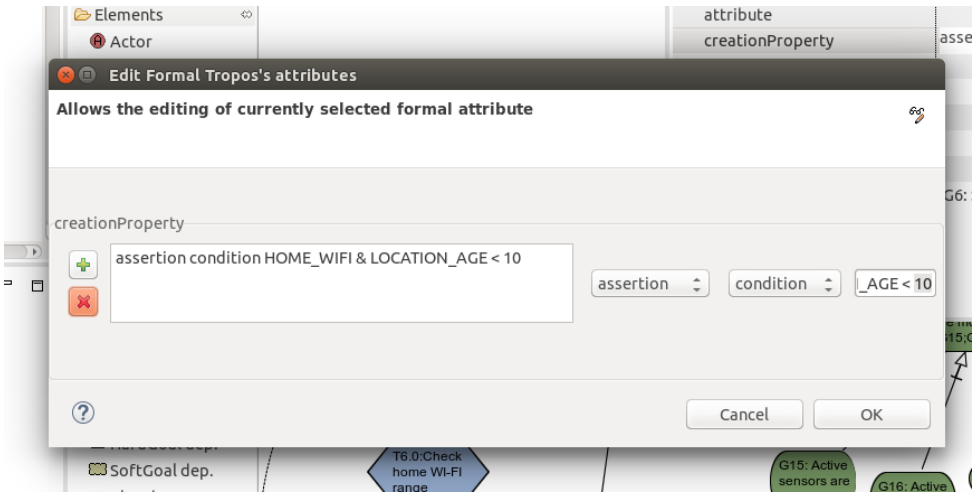


Figura 4.4: Context effect associated in the TAOM4E formal specification area.

Capítulo 5

Análise de Dependabilidade Orientada a Objetivos

This chapter describes our proposed goal-oriented dependability analysis, hereafter referred to as GODA. Figure 5.1 illustrates GODA process that starts with requirements phases of the TROPOS methodology involving goal-oriented analysis analysis briefly described in Section 4.2. Conventional goal modelling is followed by the runtime analysis (Sections 4.3) and context analysis (Section 4.4) that result in the CRGM. Next activity is the transformation of the CRGM into a DTMC, which is the core of our contribution and is described along this chapter, and by the specification of dependability properties in PCTL language as described in Section 5.2. The GODA process finishes with the design-time and runtime analysis presented in Sections 5.3 and 5.4, respectively.

Dashed arrows in the process indicates an iterative approach in which goal modelling activities may be performed multiple times until a final version of the system-to-be is reached. For instance, context restrictions may end up with no alternatives to fulfil a certain goal, which should be the case for a new iteration over the goal modelling and analysis. Also, the dependability verification can reveal violations of the dependability constraints for one or more elicited alternatives that could be tackled before implementation.

The next sections are structured as follows. First, the high-level DTMC model for MPERS is built and details about the mapping of a CRGM into a DTMC in PRISM language are provided. Next, dependability properties represented by the probability in fulfilling different system goals are defined. Finally, we briefly describe how our verification approach may be employed in both design-time analysis of the system-to-be and runtime analysis through self-adaptation.

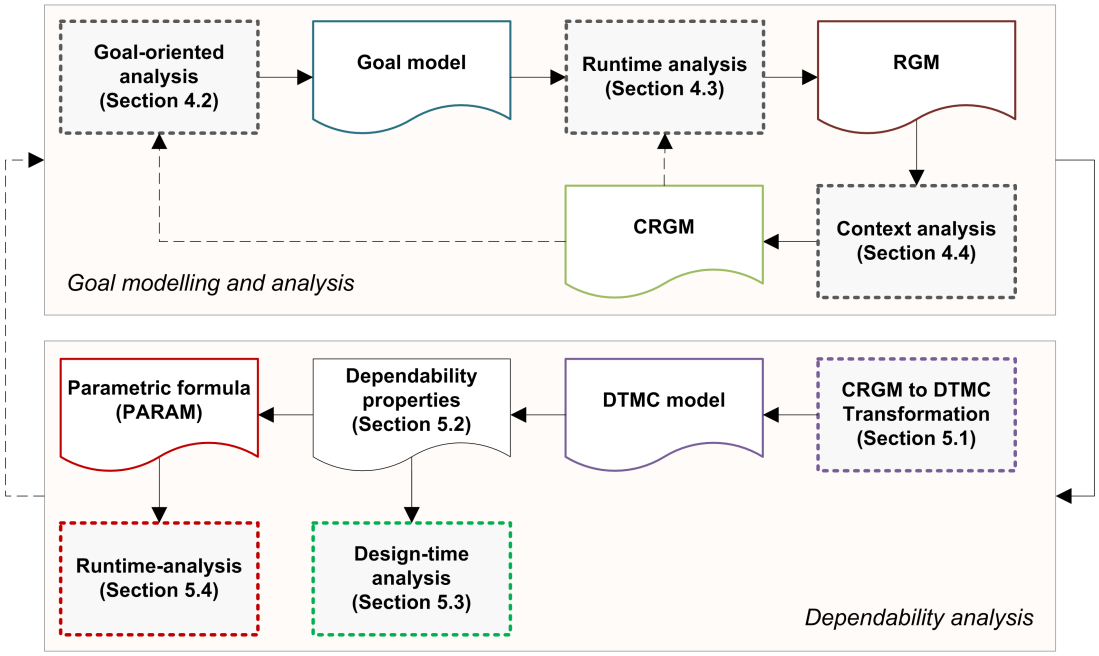


Figura 5.1: Goal-oriented dependability analysis process.

5.1 Goal-oriented Probabilistic Verification Model

This section describes our proposal regarding the transformation of a CRGM into a DTMC verification model. We call it a goal-oriented dependability analysis because the probabilistic verification model, in this case a DTMC model, is built directly from a Contextual-Runtime goal model with the purpose of evaluating PCTL properties related to the probability of different goals in being fulfilled, i.e., there is a clear mapping between the goal model representation of the system and its dependability verification.

Throughout this work, the conventional goal model will be called DGM. A DGM is formally defined as a directed graph $M = (I, R)$, where I is a set of intentional elements (nodes), while R is a set of relations (edges) [9]. For simplification purposes, only goals and tasks have been considered as intentional elements and the relations are reduced to AND/OR-decompositions. Thus, intentional elements can be in one of the two disjoint refinement groups: AND-nodes and OR-nodes. Means-end refinements are considered as OR-decomposition. A goal refined by AND/OR tasks is called a leaf-goal, while a leaf-task has no further refinements.

Given a DGM $M = (I, R)$, a RGM $R_M = (I, R, rt\_annot())$ is built by associating each non-leaf element $\epsilon \in I$ with an expression $rt\_annot(\epsilon)$, called a behaviour annotation, that is intended to specify the behaviour of ϵ. The only trivial case are single means-end tasks whose behaviour is inherited from their leaf-goals [9].

All goals and tasks elements in the model receive an unique identifier $ID(\epsilon) = prefix(\epsilon) + counter(\epsilon)$, where $prefix$ returns a G or a T for goals and tasks, respectively, and $counter$

returns an unique positive integer in the goals IDs set, if $type(\epsilon) = Goal$, or a repetition of the leaf-goal numeric ID with further decimals to distinct multiple tasks refinements for the same leaf-goal. Figure 5.2 illustrates the identifiers for a leaf-goal, for its means-end task and for this task's refinements.

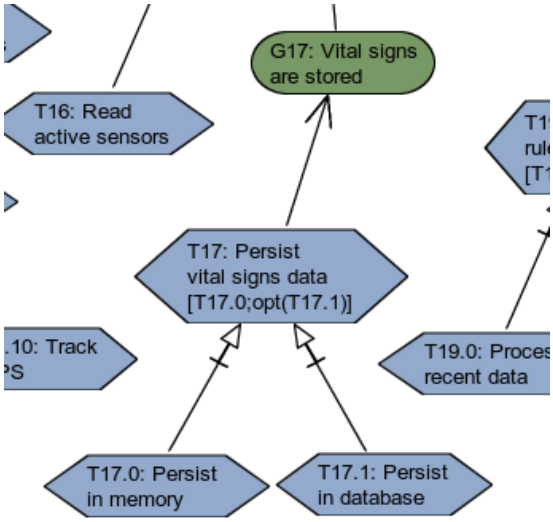


Figura 5.2: Goal G_{17} decomposed by task T_{17} that is refined by subtasks $T_{17.0}$ and $T_{17.1}$.

5.1.1 Leaf-tasks as transition systems

Since goals are ultimately realized by leaf-tasks, the overall system behaviour is the realization of those tasks. As a result, the probabilistic behaviour of the leaf-tasks can be modelled into a corresponding DTMC model. To present the well-formedness rules of our transformation from a CRGM into a DTMC model, we first represent leaf-tasks as transition systems and then as a DTMC model.

Despite the final representation of a leaf-tasks being a DTMC, we have decided for a transition system as an intermediary representation due to its simplicity and also because a DTMC is a transition system specialization whose transitions follow probabilistic distributions. In our work, only the transition from the running state to the success or failure states are non-deterministic, as can be seen in the DTMC examples presented in each case.

A transition system TS is defined by the tuple $(S, Act, \rightarrow, I, AP, L)$, where:

- S is the set of states,
- Act is a set of actions,

- $\rightarrow \subset S \times Act \times S$ is a transition relation,
- $IS \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L: S \rightarrow 2^{AP}$ is a state labelling function.

A TS is called finite if S, Act, and AP are finite. An action α evolves the system from state s to s' if $(s, \alpha, s') \in \rightarrow$, $\{s, s'\} \subseteq S$ and s is the current state. In case a state has more than one outgoing transition, the next transition is chosen nondeterministically. Labels relates a set $L(s) \in 2^{AP}$ of atomic propositions to any state s. Given a propositional logic formula Φ, then $s \models \Phi \iff L(s) \models \Phi$ [3]. The TS representing leaf-tasks is defined as follows:

- $S = \{initial, skipped, running, success, failure\}$
- $IS = \{initial\}$
- $L(initial) = \{\emptyset\}$, $L(skipped) = \{not\_selected\}$, $L(running) = \{selected\}$, $L(success) = \{succeeded\}$, $L(failure) = \{failed\}$
- $AP = \{not\_selected, selected, succeeded, failed\}$
- $Act = \{skip, run, succeed, fail\}$

Figure 5.3 presents a state diagram for a RGM. In contrast to the state diagram in [9], the *waiting* state has been suppressed, as we are interested in the outcome of a task execution and not in the execution itself, and the *skipped* state has been added to represent leaf-tasks that have been skipped for different reasons.

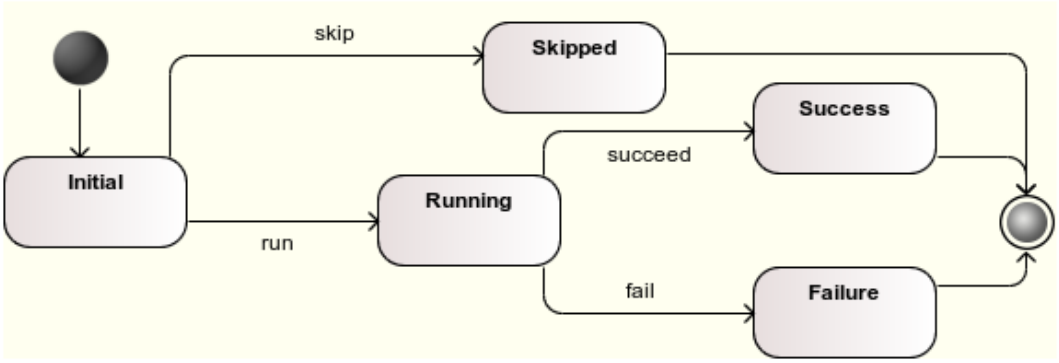


Figura 5.3: State diagram for leaf-tasks in a RGM.

Next, the leaf-tasks states are further described. For each state name, the corresponding number used in the PRISM DTMC model is associated to the variable $s \in S$.

- Initial(s=0): corresponds to the initial/ready state of a given leaf-task. From this state, a transition may occur to the running state, if the task is part of a selected system alternative, or to the final skipped state, otherwise.

- Running(s=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.

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

- Skipped(s=3): indicates that a task does not participate in the fulfilment of the analysed goal and should not impact the analysis results.

- Failure(s=4): the opposite from the final success state, meaning that a singular task execution has deviated from its expected behaviour as a consequence of a natural phenomena, a human-made fault, a malicious fault or an interaction fault, as stated in the failure definition in Section 2.5.

5.1.2 Building the high-level DTMC model from a RGM

Once the transition system of leaf-tasks has been defined, the corresponding DTMC representation follows. DTMC is an specialization of a TS defined by the tuple (S, s_i, P, AP, L), where:

- S is the set of states,

- $s_i \in S$ is the initial state,

- $P : S \times S \to [0, 1]$ is a transition probability matrix such that $\forall s \in S$, $\sum_{s' \in S} P(s, s') = 1$,

- AP is a set of atomic propositions, and

- $L : S \to 2^{AP}$ a state labelling function assigning to each state $s \in S$ a set of atomic propositions from AP.

In particular for a leaf-task, P represents the reliability of the task execution in case of a transition from running to the success state. Complementarily, $1 - P$ in case of a transition from running to the failure state.

For any goal G_i and a set Λ of leaf-tasks that fulfils all branches of subgoals up to G_i, including all alternative branches, with $G_i \in I$ and $\Lambda \subset I$, a DTMC model for G_i

is composed of k leaf-tasks DTMCs, where $k = |\Lambda|$. This model must represent, in its finite state abstraction, the same behaviour of the corresponding RGM workflow, i.e., it must preserve goals/tasks temporal order and other behaviour semantics specified by the runtime annotations. We call the resulting verification model of a high-level DTMC because leaf-tasks may represent high-level activities, as in an UML Activity Diagram, which could be further refined, e.g., by sequence diagrams [31].

In order to obtain the high-level DTMC from the RGM construct, we defined an abstraction of the execution time of different leaf-tasks in the model for better understandability of the resulting DTMC and also for building an execution chain composed of both sequential and interleaved leaf-tasks. We have formalized this abstraction according to the following definition:

Definição 1 Execution time of a RGM is a tuple $\tau = (f, p)$, where f represents a discrete time frame and p a discrete time path, with $f, p \in \mathbb{N}$.

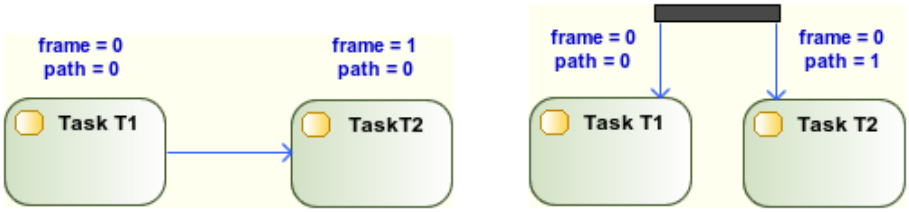
The CRGM starts in $\tau = (0, 0)$ and is parsed following a depth first algorithm. Sibling elements in the goal tree are sorted by their position in the runtime annotation ($rt\_annot()$) from left to right. Leaf-tasks can be executed either sequentially or in parallel. Sequential leaf-tasks are in subsequent time frames, i.e., $\tau_i = (f, p) \Rightarrow \tau_{i+1} = (f+1, p)$, while parallel leaf-tasks share the same time frame, but occupy different time paths, i.e., $\tau_i = (f, p) \Rightarrow \tau_{i+1} = (f, p+1)$. Each leaf-task in a CRGM is mapped to a specific τ.

Next, each behaviour rule is detailed within three levels: an UML activity diagram representation that graphically explains the behaviour specified by the rule, the resulting transition system and the following DTMC representation in PRISM. The effect of each behaviour rule on the τ of the involved tasks is also described and can be noted in the corresponding DTMC example as part of the [*action*] suffix in PRISM transition commands, where the first term represents the time frame f and the second, separated by an underscore, represent the time path p. Between different rules, only the actions in AP of the transition systems variate, while S, $\rightarrow$, IS, and L are kept the same.

Sequential and Interleaved Rule

Two fundamental behaviour rules consist of the sequential and interleaved execution orders for the fulfilment of goals and execution of tasks. In a runtime annotation, the ';' and '#' symbols represent sequential and interleaved orders, respectively. Figures 5.4a and 5.4b illustrate these behaviours in UML activity diagrams.

Sequential tasks $(T_1; T_2)$ have subsequent time frames, meaning that T_2's execution is subsequent to T_1's execution. To model this, we have synchronized the initial transition



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

of T_2 to the final transition of T_1. Considering the transition systems TS_1 and TS_2 corresponding to T_1 and T_2: $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, $1 \leq i \leq 2$, the transition system resulting from $TS_1; TS_2$ is defined by:

$$Act_1 \cap Act_2 = \{\alpha\}, \quad \frac{s_1 \xrightarrow{\alpha}_1 s'_1 \wedge s_2 \xrightarrow{\alpha}_2 s'_2}{< s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2' >}$$

where $s_1 = running$, $s_2 = initial$, $s'_1 = success$, $s'_2 = running$ and $\alpha = succeed_1 = run_2$.

In contrast, interleaved tasks ($T1\#T2$) are in the same time frame, but occupy different time paths. The interleaving occurs on the running state, while the synchronization occurs on the initial and final success states. The transition system resulting from $TS_1\#TS_2$ is defined by:

$$Act_1 \cap Act_2 = \{\alpha\}, \quad \frac{s_1 \xrightarrow{\alpha}_1 s'_1 \wedge s_2 \xrightarrow{\alpha}_2 s'_2}{< s_1, s_2 > \xrightarrow{\alpha} < s_1', s_2' >}$$

where $s_1 = initial$, $s_2 = initial$, $s'_1 = running$, $s'_2 = running$ and $\alpha = init$, and:

$$\beta_1 \in Act_1, \ \beta_2 \in Act_2, \quad \frac{s_1 \xrightarrow{\beta_1}_1 s'_1}{< s_1, s_2 > \xrightarrow{\beta_1} < s'_1, s_2 >}, \quad \frac{s_2 \xrightarrow{\beta_2}_2 s'_2}{< s_1, s_2 > \xrightarrow{\beta_2} < s_1, s'_2 >}$$

where $s_i = running$, $s'_i = success$, and $\beta_i = succeed_i$, $1 \leq i \leq 2$.

Generically, for any finite number of interleaved tasks succeeding a transition system TS_0, a resulting transition system $TS_0; (TS_1\#..\#TS_i\#..\#TS_N)$, where $0 < i \leq N$ is defined as:

$$\bigcap_{0 \leq i \leq N} Act_i = \{\alpha\}, \quad \frac{s_0 \xrightarrow{\alpha}_0 s'_0 \wedge \ldots \wedge s_i \xrightarrow{\alpha}_i s'_i \wedge \ldots \wedge s_N \xrightarrow{\alpha}_n s'_N}{< s_0, \ldots, s_i, \ldots, s_N > \xrightarrow{\alpha} < s_0', \ldots, s_i', \ldots, s_N' >}$$

where $s_0 = running$, $s_0{}' = success$, $s_i = initial$, $s'_i = running$, for $0 < i \leq N$ and $\alpha = succeed_0 = run_i$, and:

$$\bigcup_{0<i\leq N} \frac{s_i \xrightarrow{\beta_i}_i s'_i}{< s_1, ..., s_i, ...s_N > \xrightarrow{\beta_i} < s_1, ..., s'_i, ..., s_N >}$$

where $s_i = running$, $s'_i = success$, $\beta_i \in Act_i$ and $\beta_i = succeed_i$. Listings 5.1 and 5.2 present the PRISM DTMC modules for $T_{8.0}; T_{8.1}$ and $T22\#T23.0$, respectively.

```
const double rTaskT8_0;
module T8_0_RESTRequisition
  sT8_0 :[0..4] init 0;

  [success1_1] sT8_0 = 0 -> (sT8_0'=1);//init to running
  [] sT8_0 =  1 -> rTaskT8_0 : (sT8_0'=2) + (1 - rTaskT8_0) : (sT8_0'=4);//running to
      final state
  [success1_2] sT8_0 = 2 -> (sT8_0'=2);//final state success
  [success1_2] sT8_0 = 3 -> (sT8_0'=3);//final state skipped
  [failT8_0] sT8_0 = 4 -> (sT8_0'=4);//final state failure
endmodule

const double rTaskT8_1;
module T8_1_SMSNotification
  sT8_1 :[0..4] init 0;

  [success1_2] sT8_1 = 0 -> (sT8_1'=1);//init to running
  [] sT8_1 =  1 -> rTaskT8_1 : (sT8_1'=2) + (1 - rTaskT8_1) : (sT8_1'=4);//running to
      final state
  [success1_3] sT8_1 = 2 -> (sT8_1'=2);//final state success
  [success1_3] sT8_1 = 3 -> (sT8_1'=3);//final state skipped
  [failT8_1] sT8_1 = 4 -> (sT8_1'=4);//final state failure
endmodule
```

Listing 5.1: Sequential tasks T15 and T16 as DTMC modules with final transitions of the first module synchronized to the initial transition of the second.

```
const double rTaskT22;

module T22_PlaySoundAlarm
  sT22 :[0..4] init 0;

  [success0_7] sT22 = 0 -> (sT22'=1);//init to running
  [] sT22 =  1 -> rTaskT22 : (sT22'=2) + (1 - rTaskT22) : (sT22'=4);//running to final
      state
  [success0_8] sT22 = 2 -> (sT22'=2);//final state success
  [success0_8] sT22 = 3 -> (sT22'=3);//final state skipped
  [failT22] sT22 = 4 -> (sT22'=4);//final state failure
endmodule

const double rTaskT23_0;

module T23_0_DisplayEmergencyScreen
```

```
16  sT23_0 :[0..4] init 0;
17
18  [success0_7] sT23_0 = 0 -> (sT23_0'=1);//init to running
19  [] sT23_0 =  1 -> rTaskT23_0 : (sT23_0'=2) + (1 - rTaskT23_0) : (sT23_0'=4);//running
        to final state
20  [success0_8] sT23_0 = 2 -> (sT23_0'=2);//final state success
21  [success0_8] sT23_0 = 3 -> (sT23_0'=3);//final state skipped
22  [failT23_0] sT23_0 = 4 -> (sT23_0'=4);//final state failure
23 endmodule
```

Listing 5.2: Parallel tasks T22 and T23.0 as DTMC modules with initial and final transitions synchronized and interleaved running transitions.

Alternative Rule

Alternative behaviour rule specifies that only one goal/task among two or more may be selected, i.e., they are mutually exclusive (XOR-decomposition). Figure 5.5 illustrate alternative behaviour in UML activity diagram.

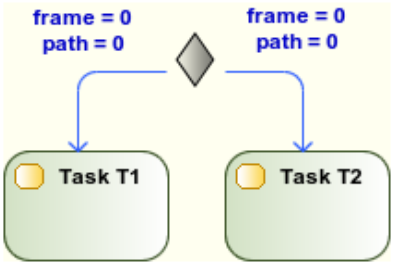


Figura 5.5: Alternative tasks T1|T2.

Alternative leaf-tasks share both time frame and path, as only one task may be selected. The increment in τ is defined by the preceding sequential or interleaved order rule. The alternative selection is conditioned to an input data provided at design-time or runtime. Considering the transition systems TS_1 and TS_2 corresponding to alternative tasks T_1 and T_2, where $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, for $1 \leq i \leq 2$, the transition system resulting from $TS_1|TS_2$ is defined by the rules for the transitions from the initial state to the running or skipped states:

$$\{\gamma_1, \gamma_2\} \subset (Act_1 \cap Act_2)$$

$$\frac{s_1 \xrightarrow{\gamma_1}_1 s'_1}{< s_1, s_2 > \xrightarrow{\gamma_1} < s'_1, s'_2 >}, \quad \frac{s_2 \xrightarrow{\gamma_2}_2 s''_2}{< s_1, s_2 > \xrightarrow{\gamma_2} < s''_1, s''_2 >}$$

where $s_1 = initial$, $s_2 = initial$, $s'_1 = running$, $s'_2 = skipped$, $s''_1 = skipped$, $s''_2 = running$, $\gamma_1 = run_1$ and $\gamma_2 = run_2$.

Generically, for any finite number of alternative tasks, a resulting transition system $TS_1|..|TS_i|..|TS_N$, for $0 < i \leq N$, is defined as:

$$\{\gamma_1, .., \gamma_i, .., \gamma_N\} \subset \bigcap_{0<i\leq N} Act_i$$

$$\bigcup_{0<i\leq N} \frac{s_i \xrightarrow{\gamma_i}_i s'_i}{< s_1, ..., s_i, ...s_N > \xrightarrow{\gamma_i} < s'_1, ..., s'_i, ..., s'_N >}$$

where $s_i = initial$, $s_i' = running$, $s'_j = skipped$, for $0 < j \leq N$, such that $j \neq i$, and $\gamma_i = run_i$.

To represent this rule in a DTMC model, additional selecting enumerators define which alternative from a set of two or more is selected for execution instead of different actions for selecting different alternatives ($\{run_1, run_2, ..\}$). As well as in optional execution, $XOR\_E_i = 0$ results in a deterministic transition from the initial state to the skipped state for all leaf-tasks underlying the element E_i, in the case of a goal or a non-leaf task, or for E_i itself, in the case E_i is a leaf-task. In opposition, a deterministic transition from the initial state to the running state occurs if $XOR\_E_i = 1$. We guarantee the structural validity for alternative execution by using a binary construction that avoids more than one alternative to be selected at the same time. In specific, for any number of alternative tasks N, where $T_1|..|T_i|..|T_N$ and $0 < i \leq N$, their initial transition in a DTMC is built with the following pattern:

$$[success]sT_i = 0 \rightarrow \prod_{j=1;j\neq i}^{N} (1 - XOR\_T_j) * (XOR\_T_i) : (sT_i' = 1) + \prod_{j=1;j\neq i}^{N} (XOR\_T_j) * (1 - XOR\_T_i) : (sT_i' = 3) \quad (5.1)$$

For any invalid combination of the selecting enumerators, the DTMC becomes invalid, as the sum of probabilities would not be 1. Listing 5.3 presents the DTMC modules for alternative tasks T16.20, T16.21 and T16.22.

```
param int XOR_T16_20;
param int XOR_T16_21;
```

```
3  param int XOR_T16_22;
4
5  const double rTaskT16_20;
6
7  module T16_20_TriangulateMobileSignal
8    sT16_20 :[0..4] init 0;
9
10   [success0_0]   sT16_20 = 0 -> XOR_T16_20*(1 - XOR_T16_21)*(1 - XOR_T16_22) : (sT16_20
         '=1) + (1 - XOR_T16_20)*XOR_T16_21*XOR_T16_22 : (sT16_20'=3);//init to running or
         skip
11   [] sT16_20 =  1 -> rTaskT16_20 : (sT16_20'=2) + (1 - rTaskT16_20) : (sT16_20'=4);//
         running to final state
12   [success1_1] sT16_20 = 2 -> (sT16_20'=2);//final state success
13   [success1_1] sT16_20 = 3 -> (sT16_20'=3);//final state skipped
14   [failT16_20] sT16_20 = 4 -> (sT16_20'=4);//final state failure
15 endmodule
16
17 const double rTaskT16_21;
18
19 module T16_21_PublicWI_FILookup
20   sT16_21 :[0..4] init 0;
21
22   [success0_0]   sT16_21 = 0 -> (1 - XOR_T16_20)*XOR_T16_21*(1 - XOR_T16_22) : (sT16_20
         '=1) : (sT16_21'=1) + XOR_T16_20*(1 - XOR_T16_21)*XOR_T16_22 : (sT16_21'=3);//init
         to running or skip
23   [] sT16_21 =  1 -> rTaskT16_21 : (sT16_21'=2) + (1 - rTaskT16_21) : (sT16_21'=4);//
         running to final state
24   [success1_1] sT16_21 = 2 -> (sT16_21'=2);//final state success
25   [success1_1] sT16_21 = 3 -> (sT16_21'=3);//final state skipped
26   [failT16_21] sT16_21 = 4 -> (sT16_21'=4);//final state failure
27 endmodule
28
29 const double rTaskT16_22;
30
31 module T16_22_TrackGPS
32   sT16_22 :[0..4] init 0;
33
34   [success0_0]   sT16_22 = 0 -> (1 - XOR_T16_20)*(1 - XOR_T16_21)*XOR_T16_22 : (sT16_22
         '=1) + XOR_T16_20*XOR_T16_21*(1 - XOR_G16) : (sT16_22'=3);//init to running or skip
35   [] sT16_22 =  1 -> rTaskT16_22 : (sT16_22'=2) + (1 - rTaskT16_22) : (sT16_22'=4);//
         running to final state
36   [success1_1] sT16_22 = 2 -> (sT16_22'=2);//final state success
37   [success1_1] sT16_22 = 3 -> (sT16_22'=3);//final state skipped
38   [failT16_22] sT16_22 = 4 -> (sT16_22'=4);//final state failure
39 endmodule
```

Listing 5.3: Alternative tasks T7.00, T7.01 and T7.10 as DTMC modules with additional integer variable used for selection.

Optional Rule

Given a decomposed goal or task, its fulfilment or execution may be tagged as optional, meaning that the non-satisfaction of this element does not prevent its refined goal or task

to be met. In this case, the other non-optional elements in an AND-decomposition must be satisfied. Figure 5.6 illustrates an optional task T.

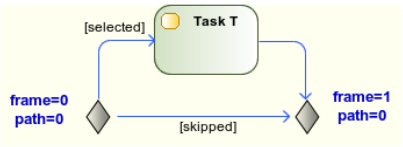


Figura 5.6: Optional task T.

Optional behaviour does not causes any additional increment to τ, which is defined by previous sequential or interleaved order rule. In order to synchronize the optional task T_0 with a subsequent task T_1, the initial transition of T_1 is synchronized to the *skip* action of T_0, in addition to the synchronization to T_0's *succeed* action. Thus, considering the transition system for these tasks defined by $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, for $0 \leq i \leq 1$, the resulting transition system from $opt(TS_0); TS_1$ is defined by the rules:

$$\{\alpha, \gamma\} \subset (Act_0 \cap Act_1)$$

$$\frac{s_o \xrightarrow{\alpha}_0 s'_0 \wedge s_1 \xrightarrow{\alpha}_1 s'_1}{< s_0, s_1 > \xrightarrow{\alpha} < s'_0, s'_1 >}$$

where $s_o = running$, $s'_o = success$, $s_1 = initial$, $s'_1 = running$, $\alpha = succeed_0 = run_1$.

$$\frac{s_0 \xrightarrow{\gamma}_0 s'_0 \wedge s_1 \xrightarrow{\gamma}_1 s'_1}{< s_0, s_1 > \xrightarrow{\gamma} < s'_0, s'_1 >}$$

where $s_o = initial$, $s'_o = skiped$, $s_1 = initial$, $s'_1 = running$ and $\gamma = skip_0 = run_1$.

Similarly to alternative rule, optional leaf-tasks are selected by an additional variable in their DTMC model. Instead of different actions (*run*, *skip*), a select enumerator determines in a single action which state will follow the initial state: *running*, if $OPT\_T\_ID = 1$, or *skipped*, if $OPT\_T\_ID = 0$. In PRISM, we achieved this by using this enumerator to model a deterministic transition to either the running state or to the skipped state. Listing 5.4 illustrates the DTMC module of optional task T17.1 with the select enumerator $OPT\_T\_17\_1$.

```
1 const double rTaskT17_1;
2 const int OPT_T17_1;
3
4 module T17_1_PersistInDatabase
```

```
5   sT17_1 :[0..4] init 0;
6
7   [success1_3] sT17_1 = 0 -> (OPT_T17_1) : (sT17_1'=1) + (1 - OPT_T17_1) : (sT17_1'=3);//
        init to running or to skipped
8   [success1_3] sT17_1 = 0 -> (sT17_1'=4);//init to fail
9   [] sT17_1 =  1 -> rTaskT17_1 : (sT17_1'=2) + (1 - rTaskT17_1) : (sT17_1'=4);//running
        to final state
10  [success1_4] sT17_1 = 2 -> (sT17_1'=2);//final state success
11  [success1_4] sT17_1 = 3 -> (sT17_1'=3);//final state skipped
12  [failT17_1] sT17_1 = 4 -> (sT17_1'=4);//final state failure
13 endmodule
```

Listing 5.4: An optional task T17.1 as a DTMC with additional select enumerator OPT_T17_1.

Conditional Rule

Some goals/tasks are conditioned to the fulfilment/execution of another goal/task. In such cases, an OR-decomposition is annotated with a ternary rule $try(E_1)\ ?\ E_2 : E_3$ where E_2 is conditioned to the success of E_1, while E_3 is conditioned to its failure. The *skip* term is used if no further behaviour is conditioned to either the success or the failure of E_1. For example, in $try(T_1)\ ?\ skip : T_2$, task T_2 is only required for execution if T_1 fails and no further behaviour is expected if T_1 succeeds. In particular, $try(T_1)\ ?\ skip : T_2$ defines a design diversity as a fault tolerance for T_1. Figure 5.7 illustrates a conditional decomposition.

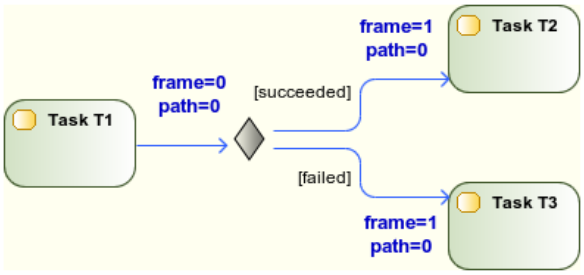


Figura 5.7: Conditional tasks $T_{9.0}$ and $T_{9.1}$.

Conditional behaviour does increment the time frame as the conditioned leaf-tasks are subsequent to the success or failure of $E1$. Considering the transition systems TS_1, TS_2 and TS_3 corresponding to alternative tasks T_1, T_2 and T_3: $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, for $1 \leq i \leq 3$, the transition system resulting from $try(TS_1)\ ?\ TS_2 : TS_3$ is defined by the rules:

$$\{\delta_1, \delta_2\} \subset (Act_1 \cap Act_2 \cap Act_3)$$

$$\frac{s_1 \xrightarrow{\delta_1} s'_1 \ \wedge \ s_2 \xrightarrow{\delta_1} s'_2 \ \wedge \ s_3 \xrightarrow{\delta_1} s'_3}{< s_1, s_2, s_3 > \xrightarrow{\delta_1} < s'_1, s'_2, s'_3 >}, \quad \frac{s_1 \xrightarrow{\delta_2} s''_1 \ \wedge \ s_2 \xrightarrow{\delta_2} s''_2 \ \wedge \ s_3 \xrightarrow{\delta_2} s''_3}{< s_1, s_2, s_3 > \xrightarrow{\delta_2} < s''_1, s''_2, s''_3 >}$$

where $s_1 = initial$, $s_2 = initial$, $s_3 = initial$, $s'_1 = success$, $s'_2 = running$, $s'_3 = skipped$, $s''_1 = failure$, $s''_2 = skipped$, $s''_3 = running$, $\delta_1 = success_1$ and $\delta_2 = failure_1$.

Any task subsequent to a conditional execution must have its initial transition synchronized to all possible successful outcomes, i.e., to the *succeed* actions of E_2 and E_3. Given the transition systems TS_1, TS_2, TS_3 and TS_4 corresponding to the leaf-tasks T_1, T_2, T_3 and T_4: $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$, for $1 \leq i \leq 4$, the resulting transition system for $(try(TS_1) \ ? \ TS_2 : TS_3); TS_4$ is defined by the additional rules:

$$\alpha_2 \in (Act_2 \cap Act_4), \quad \alpha_3 \in (Act_3 \cap Act_4)$$

$$\frac{s_2 \xrightarrow{\alpha_2}_2 s'_2 \ \wedge \ s_4 \xrightarrow{\alpha_2}_4 s'_4}{< s_2, s_4 > \xrightarrow{\alpha_2} < s'_2, s'_4 >}, \quad \frac{s_3 \xrightarrow{\alpha_3}_3 s'_3 \ \wedge \ s_4 \xrightarrow{\alpha_3}_4 s'_4}{< s_3, s_4 > \xrightarrow{\alpha_3} < s'_3, s'_4 >}$$

where $s_2 = running$, $s_3 = running$, $s_4 = initial$, $s'_2 = success$, $s'_3 = success$, $s'_4 = running$, $\alpha_2 = success_2$ and $\alpha_3 = success_3$.

To represent conditional behaviour in DTMC models, shared actions synchronize the initial transition of leaf-tasks conditioned to the success and failure of a third leaf-task or a set of leaf-tasks, as the conditional rule may be applied to one or to multiple elements at a time or to non-leaf elements. Figure 5.5 present the DTMCs for conditional tasks in the rule $try(T_{9.0}) \ ? \ skip : T_{9.1}$.

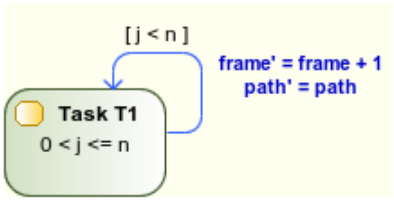
```
1  const double rTaskT9_0;
2
3  module T9_0_RESTRequisition
4    sT9_0 :[0..4] init 0;
5
6    [success0_2] sT9_0 = 0 -> (sT9_0'=1);//init to running
7    [] sT9_0 =  1 -> rTaskT9_0 : (sT9_0'=2) + (1 - rTaskT9_0) : (sT9_0'=4);//running to
         final state
8    [success2_3] sT9_0 = 2 -> (sT9_0'=2);//final state success
9    [success2_3] sT9_0 = 3 -> (sT9_0'=3);//final state skipped
10   [failT9_0] sT9_0 = 4 -> (sT9_0'=4);//final state failure
11 endmodule
12
13 const double rTaskT9_1;
14
15 module T9_1_SMSNotification
16   sT9_1 :[0..4] init 0;
```

```
17
18   [failT9_0]   sT9_1 = 0 -> (sT9_1'=1);//init to running
19   [success2_3] sT9_1 = 0 -> (sT9_1'=3);//not used, skip running
20   [] sT9_1 =  1 -> rTaskT9_1 : (sT9_1'=2) + (1 - rTaskT9_1) : (sT9_1'=4);//running to
         final state
21   [success2_4] sT9_1 = 2 -> (sT9_1'=2);//final state success
22   [success2_4] sT9_1 = 3 -> (sT9_1'=3);//final state skipped
23   [failT9_1] sT9_1 = 4 -> (sT9_1'=4);//final state failure
24 endmodule
```

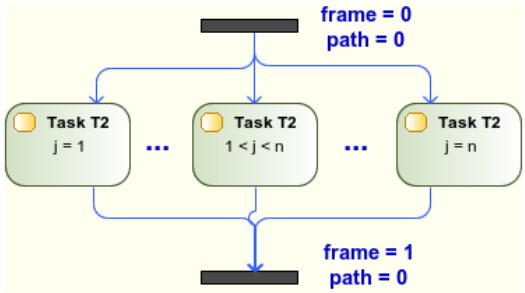
Listing 5.5: Conditional tasks T9.0 and T9.1 as DTMCs.

Cardinality Rules

Variations of the original RGM proposal were employed for the $E+$ and $E\#$ cardinality rules. Instead of an infinite/undetermined cardinality, analysts should provide the upper bound cardinality limit. Figures 5.8a and 5.8b illustrate sequential and interleaved task behaviours, respectively.



(a) Sequential cardinality T_1, for $0 < j \leq n$.



(b) Interleaved cardinality of task T_2, for $0 < j \leq n$.

In our proposal, cardinality is represented by the number of sequential $(E + n)$ or interleaved $(E\#n)$ executions of the same task. Considering a transition system TS, cardinality follows the same rules of sequential or interleaved executions, except that the same transition system is repeated n times. Thus, the resulting transition systems $TS+n$ and $TS\#n$ are the same as $TS_1; ..; TS_i; ..; TS_n$ and $TS_1\#..\#TS_i\#..\#TS_n$, where:

$$\sum_{i=1}^{N-1} TS_i = TS_{i+1}$$

.

To represent these rules in a DTMC model, the corresponding leaf-tasks modules are repeated $n - 1$ times though PRISM language module renaming, as $n = 1$ is trivially represented by the original module. In case of sequential cardinality, the state variable, the initial and final transition actions are renamed for all repetitions, forming a sequential execution chain of the same task, where $\tau_i = (f, p) \Rightarrow \tau_{i+1} = (f + 1, p)$. In case of

interleaved cardinality, only the state variable is renamed, resulting in the synchronization of the initial and the final transitions of the renamed modules, leaving only the run actions unsynchronized. This module renaming forms an interleaved execution of the same task, with $\tau_i = (f, p) \Rightarrow \tau_i = (f, p+1)$. It must be noted that the time path resulting from the interleaved execution is not incremented, as the parallelism ends with the rule and the main execution path is restored.

Listings 5.6 and 5.7 present the DTMC models for both cases. The second example does not come from the MPERS RGM in Figure 4.2 and was introduced to exemplify the interleaved cardinality module renaming approach.

```
const double rTaskT22;
module T22_ListenToButton
  sT22 :[0..4] init 0;

  [success0_8] sT22 = 0 -> (sT22'=1);//init to running
  [] sT22 =  1 -> rTaskT22 : (sT22'=2) + (1 - rTaskT22) : (sT22'=4);//running to final
      state
  [success0_9] sT22 = 2 -> (sT22'=2);//final state success
  [success0_9] sT22 = 3 -> (sT22'=3);//final state skipped
  [failT22] sT22 = 4 -> (sT22'=4);//final state failure
endmodule
module T22_S2 = T22_ListenToButton [ sT22=sT22_S2, success0_9=success0_10, success0_8=
    success0_9, failT22=failT22_S2 ] endmodule
```

Listing 5.6: Sequential cardinality with n=2 for task T22.

```
const double rTaskT22;
module T22_ListenToButton
  sT22 :[0..4] init 0;

  [success0_8]  sT22 = 0 -> (sT22'=1);//init to running
  [] sT22 = 1 -> rTaskT22 : (sT22'=2) + (1 - rTaskT22) : (sT22'=4);//running to final
      state
  [success0_9] sT22 = 2 -> (sT22'=2);//final state success
  [success0_9] sT22 = 3 -> (sT22'=3);//final state skipped
  [failT22] sT22 = 4 -> (sT22'=4);//final state failure
endmodule
module T22_N2 = T22_ListenToButton [ sT22=sT22_N2, failT22=failT22_N2 ] endmodule
module T22_N3 = T22_ListenToButton [ sT22=sT22_N3, failT22=failT22_N3 ] endmodule
```

Listing 5.7: Interleaved cardinality with n=3 for task T22.

In addition to the original RGM, we also support the specification of retries for tasks executions ($E@n$) as a cardinality behaviour rule. In this particular case, the *run* action is performed multiple times until the maximum number of retries or the final success state are reached. Only in the first case the fail action takes the task module to the final failure state. Listing 5.8 illustrates this behaviour in a DTMC model.

```
const double rTaskT26_0;
const double maxRetriesT26_0=3;
```

```
3
4  module T26_0_RESTRequisition
5    sT26_0 :[0..4] init 0;
6    triesT26_0 : [0..3] init 0;
7
8    [success0_10] (G25) &  sT26_0 = 0 -> (sT26_0'=1);//init to running
9    [] sT26_0 = 1 & triesT26_0 < maxRetriesT26_0 -> rTaskT26_0 : (sT26_0'=2) + (1 -
         rTaskT26_0) : (triesT26_0'=triesT26_0+1);//try
10   [] sT26_0 = 1 & triesT26_0 = maxRetriesT26_0 -> (sT26_0'=4);//no more retries
11   [success0_11] sT26_0 = 2 -> (sT26_0'=2);//final state success
12   [success0_11] sT26_0 = 3 -> (sT26_0'=3);//final state skipped
13   [failT26_0] sT26_0 = 4 -> (sT26_0'=4);//final state failure
14 endmodule
```

Listing 5.8: Retry behaviour with n=2 for task T26.0.

5.1.3 Context effects in the high-level DTMC model

Regarding a goal-oriented dependability analysis, context variation may change the scope of the verification and consequently limit the operational leaf-tasks that must be part of the analysis - either because they do not have a goal to satisfy in that context or because they are restricted by that context and cannot be selected for execution. Besides the effects over goals and tasks, we have not considered the direct context effect over qualitative softgoals, as softgoals provide no clear-cut criteria for their analysis. Nonetheless, our methodology evaluates the dependability of systems as the overall goal achievement that may be affected by context restrictions on goals and tasks, which in turn can be considered as an indirect context effect over a qualitative goal.

Given a CRGM, all context effects are parsed following a depth first approach for traversing the directed tree graph. Even if a context variable occurs multiple times in the same context formula or across other contexts in the model, only one variable is declared in the DTMC model (PRISM variable declaration). A reference to this variable is then used to compose the boolean formulas in the DTMC models corresponding to the affected leaf-tasks, according to the type of effect it imposes. Next, the modelling of the context effects on goals and tasks in a DTMC model is described.

Contextual goals activation/restriction

If a goal is activated/restricted by a context, the corresponding verification scope must be adjusted. The MPERS RGM in Figure 4.2 has a few goals whose activation depends on specific context conditions, as specified in Table 4.1. Consequently, leaf-tasks related to these goals must not affect the analysis results if these conditions are not satisfied. In our proposal, the leaf-tasks related to a restricted goal are skipped from execution.

For instance, the goals '*stationary geolocation is checked*' and '*moving geolocation is tracked*' are mandatory subgoals for the '*patient location is monitored*' goal. The first is activated by the context 'user is at home' and the later by the negation of this same context. Analysts have put this restriction to avoid an aggressive geolocation tracking that consumes too much battery when the user is known to be at home. The following formulas specify this restricting context and its negation:

$$\textbf{C1:} \quad HOME\_WIFI \;\; \& \;\; LOCATION\_AGE \; < \; 10 \tag{5.2}$$

$$\textbf{!C1:} \quad !HOME\_WIFI \;\; | \;\; LOCATION\_AGE \; >= \; 10 \tag{5.3}$$

Home WI-FI BSSID is configured once and stored by the application. If its signal is not in reach for the last 10 minutes, patient is considered to be outdoors. To map these restrictions into the DTMC model, $HOME\_WIFI$ and $LOCATION\_AGE$ are declared as boolean and double variables, respectively. The $C1$ and $!C1$ formulas are then employed as guard conditions in the corresponding leaf-task modules. If the context formula is not satisfied, a deterministic transition to the skipped final state (sTask=3) automatically excludes the related leaf-tasks from the analysis results. If the context formula holds, a deterministic transition evolves the leaf-task to the running state (sTask=1). Listing 5.9 presents the DTMC model of a MPERS leaf-task whose goal is restricted by the context formula $C1$.

```
1  const bool HOME_WIFI;
2  const double LOCATION_AGE;
3  const double rTaskT6_0=0.999;
4
5  module T6_0_CheckHomeWI_FIRange
6    sT6_0 :[0..4] init 0;
7
8    [success0_0] (HOME_WIFI & LOCATION_AGE < 10) & sT6_0 = 0 -> (sT6_0'=1);//init to
         running
9    [success0_0] !(HOME_WIFI & LOCATION_AGE < 10) & sT6_0 = 0 -> (sT6_0'=3);//init to
         skipped
10
11   [] sT6_0 =  1 -> rTaskT6_0 : (sT6_0'=2) + (1 - rTaskT6_0) : (sT6_0'=4);//running to
         final state
12   [success0_1] sT6_0 = 2 -> (sT6_0'=2);//final state success
13   [success0_1] sT6_0 = 3 -> (sT6_0'=3);//final state skipped
14   [failT6_0] sT6_0 = 4 -> (sT6_0'=4);//final state failure
15 endmodule
```

Listing 5.9: Contextual restrictions on a leaf-task execution.

Tasks restriction

Regarding the context restriction on system tasks, all operations depending on a specific system resource considered to be dynamic could be tagged with a context restriction. Also, stakeholder preferences may specify the availability of a given task only in specific contexts.

The MPERS CRGM has also a few context restrictions on tasks. For example, the restrictions over the communication tasks '*REST requisition*' and '*REST service*' (context 'internet is available'), over the geolocation tracking tasks '*public WIFI*' and '*track GPS*' (contexts 'WI-FI is on' and 'GPS signal is available') and over the storage task '*persist in database*' (context 'disk memory available'). The following boolean formulas represent these contexts:

$$\textbf{C2:} \quad CONNECTION \tag{5.4}$$

$$\textbf{C3:} \quad WI\_FI \tag{5.5}$$

$$\textbf{C4:} \quad GPS\_SIGNAL \tag{5.6}$$

$$\textbf{C5:} \quad USED\_DISK <= 95 \tag{5.7}$$

These formulas follow the same principle from the previous goal restriction formulas. For C5, a threshold of 95% of disk usage and not 100% has been defined to avoid instabilities in the operational system. The availabilities of the Wi-Fi and GPS signals are provided by the platform and have been modelled as boolean facts.

In contrast to the goals restriction, tasks restrictions are modelled by an additional deterministic transition to the failure state guarded by the negation of the corresponding context formula instead of a transition to the skipped state. Consequently, if this formula is evaluated as false, the leaf-task will certainly fail (sTask=4). This is in accordance to the perception that, if a task is required and restricted at the same time, a failure will occur. Similarly to the contextual goal activation/restriction, if the context formula holds, a deterministic transition evolves the leaf-task to the running state (sTask=1). Listing 5.10 presents a DTMC model with one restricted MPERS leaf-task.

```
1 const bool CONNECTION;//context variable declaration
2 const double rTaskT9_0;
3
4 module T9_0_RESTRequisition
5   sT9_0 :[0..4] init 0;
6
```

```
7   [success0_2] CONNECTION & sT9_0 = 0 -> (sT9_0'=1);//init to running
8   [success0_2] !CONNECTION & sT9_0 = 0 -> (sT9_0'=4);//init to failure
9
10
11  [] sT9_0 =  1 -> rTaskT9_0 : (sT9_0'=2) + (1 - rTaskT9_0) : (sT9_0'=4);//running to
        final state
12  [success2_3] sT9_0 = 2 -> (sT9_0'=2);//final state success
13  [success2_3] sT9_0 = 3 -> (sT9_0'=3);//final state skipped
14  [failT9_0] sT9_0 = 4 -> (sT9_0'=4);//final state failure
15 endmodule
```

Listing 5.10: Contextual restrictions on a leaf-task execution.

5.2 Dependability Property Specification

Once a probabilistic verification model is built from a system CRGM, different PCTL properties may be specified for verification. As part of our goal-oriented dependability analysis, we focus on the reachability property for the success of system goals. The reachability of a final and successful system state defines the reliability of that system, i.e., the probability in fulfilling its root or ultimate goal. Accordingly, the reachability of a partial success state indicates the partial reliability of the system, i.e., the probability in fulfilling one or more of its subgoals. Reachability can be represented by the Probabilistic existence property [14]:

$$P =?\ [\ F\ (\varphi)\ ] \tag{5.8}$$

Given a CRGM $CR_M = (I, R, rt\_annot(), ctx\_annot())$ and a set of leaf-tasks $\Lambda \in I$, the scope of the probabilistic verification is defined as:

- *Global*, if Λ is a minimum set composed of the leaf-tasks that satisfies the chain of subgoals up to the root goal G_{root}, where $G_{root} \in I$.
- *Local*, if Λ is a set composed of leaf-tasks that satisfies the chain of subgoals up to a goal G_x, where $G_i \in I$ and $G_i \neq G_{root}$.

For a global or local verification of G_i, the proposition φ in (5.8) represents the success of G_i and is composed of atomic propositions for the transition systems of the leaf-tasks in Λ according to the behaviour rules and the AND/OR-decompositions in the CRGM following a depth-first algorithm: More coarse propositions are recursively formed from the connection of propositions of underlying elements until a final proposition for the fulfilment of G_i is composed, where G_i is satisfied $\iff$ φ is satisfied. Figure 5.9 presents a small example of the atomic propositions underlying goal G_9.

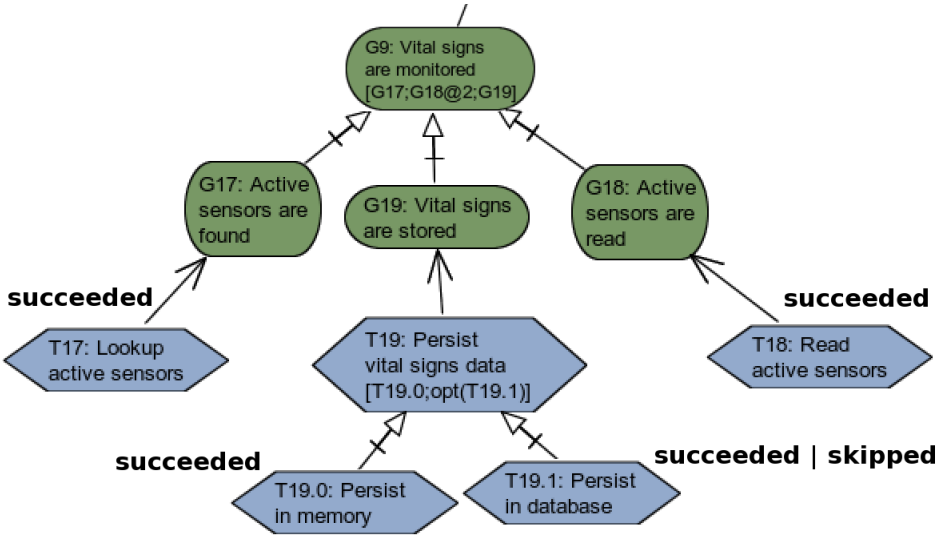


Figura 5.9: Atomic propositions required for the success of local goal G_9

Next, the propositions ϕ for the leaf-tasks whose behaviour have been specified by different rules in a CRGM are individually described in terms of the atomic propositions of leaf-tasks defined in Section 5.1.1:

- Sequential or interleaved execution of tasks T_1 and T_2:

$$\phi = (succeeded_1 \ \wedge \ succeeded_2)$$

- Optional execution of leaf-task T:

$$\phi = (succeeded \ \vee \ failed)$$

- Alternative execution of leaf-tasks T_i, where $0 < i \leq N$:

$$\phi = \bigvee_{i=1}^{N} (succeeded_i)$$

- Conditional execution of tasks T_1, T_2 and T_3:

$$\phi = ((succeeded_1 \ \wedge \ succeeded_2) \ \vee \ (failed_1 \ \wedge \ succeeded_3))$$

- Sequential or interleaved cardinality of n leaf-tasks T_i, where $0 < i \leq N$:

$$\phi = \bigwedge_{i=1}^{n} (succeeded_i)$$

- n retries of a leaf-task instances T_j:

$$\phi = \bigvee_{j=1}^{n} (succeeded_j)$$

5.3 Design-time analysis

At design-time, analysts may benefit from the rich graphical environment provided by PRISM probabilistic model checker tool. Given a propositional formula specifying the success in fulfilling a specific MPERS goal G_9 whose leaf-tasks have been mapped into the DTMC model, a corresponding proposition φ is defined as:

$$\varphi_9 = (succeeded_{T15} \,\&\, succeeded_{T16}) \,\&\, (succeeded_{T17\_0} \,\&\, (succeeded_{T17\_1} \mid skipped_{T17\_1})) \tag{5.9}$$

The probability of fulfilling G_9, i.e., the reachability of G_{9S} is specified by the following Probabilistic existence property:

$$P = ? \; [\; F \; (\varphi_9)] \tag{5.10}$$

Once the reliabilities of the involved leaf-tasks have been collected, the verification of this property through PRISM *verification feature* reveals the probability in eventually fulfilling the analysed goal. To illustrate this, considering a fixed reliability of 0.98 for all leaf-tasks $\epsilon \in \Lambda$, the probability of fulfilling G_9 is evaluated by PRISM as:

$$P_{reach}(\varphi_9) = 0.929199$$

PRISM also provides the *experiment feature* that automates multiple instances of a model checking. Instead of fixing the value of all constants in the model, one or more constants may range from an initial value to a final value in fixed steps. The experiment creates a series graph and reveals the trend for the analysed property in respect to the input range. For example, by ranging the reliability of tasks $T_{8.0}$ and $T_{7.10}$ from 0 to 1 in steps of 0.1 and leaving all other leaf-tasks reliabilities values fixed in 0.98 as part of

the G_{1S} reachability property (5.12) verification, the experiments produces the composed graphic in Figure 5.10.

$$\varphi 1 \ = \ (succeeded_{T15} \,\&\, succeeded_{T16}) \,\&\, (succeeded_{T17\_0} \,\&\, (succeeded_{T17\_1} \mid skippedT17\_1)) \tag{5.11}$$

$$P \ = ? \ [\ F \ (G_{1S})] \tag{5.12}$$

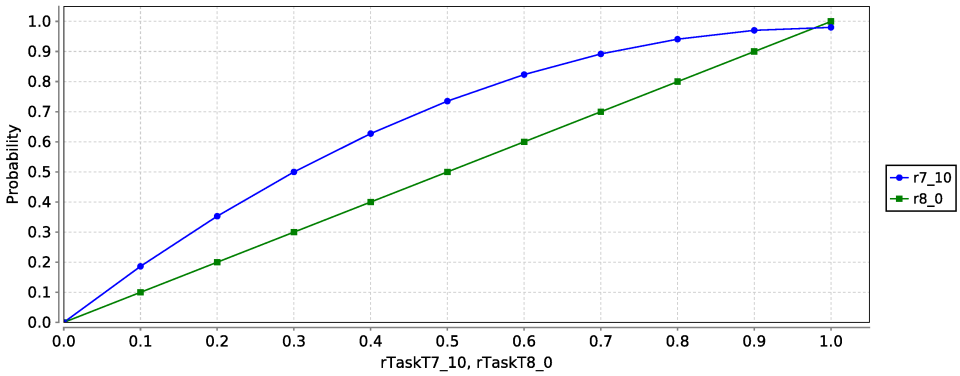


Figura 5.10: Experiments results with $T_{7.10}$ and $T_{8.0}$ ranging from 0 to 1.

5.4 Runtime analysis

Given a verification model, the feasibility of a runtime analysis depends on metrics such as its scalability and performance and also on the technical feasibility of an automated verification. As part of our third research question, we investigate the use of the PARAM tool to create a parametric formula for the probability of fulfilling one or more system goals in a CRGM.

PARAM extends PRISM with the *param* keyword that indicates which variables are parametrized. PARAM has a few restrictions regarding where parameters can be placed in the model. In fact, parameters can only be part of the right part of a PRISM command as a transition probability ranging from 0 to 1. In our proposal, parameters were used to represent, in a parametric DTMC model, the following constructs:

- The select enumerators for optional and alternative behaviour rules.
- The reliability of the leaf-tasks.

Once the parametric DTMC model for a goal G_i of a CRGM is built and a Probabilistic existence property with φ_i is specified, PARAM tool generates an arithmetic formula, also

called a parametric formula. The evaluation of this formula requires the initialization of all corresponding parameters with the values for the selection of optional and alternative tasks and also for the reliability of all involved leaf-tasks. The result of this evaluation represents the probability of G_i in being fulfilled, i.e., it represents the reachability of φ_i.

For example, given the parametric DTMC model and the success proposition φ_2 for the G_2 goal of the MPERS CRGM, the following parametric formula is generated:

$$\begin{aligned} &OPT\_T_{19.1} * rTaskT_{17} * rTaskT_{18} * rTaskT_{19.0} * rTaskT_{19.1} * rTaskT_{20} * rTaskT_{21.0} * rTaskT_{21.1} \\ &- OPT\_T\_19.1 * rTaskT_{17} * rTaskT_{18} * rTaskT_{19.0} * rTaskT_{20} * rTaskT_{21.0} * rTaskT_{21.1} \\ &+ rTaskT_{17} * rTaskT_{18} * rTaskT_{19.0} * rTaskT_{20} * rTaskT_{21.0} * rTaskT_{21.1} \end{aligned} \quad (5.13)$$

The evaluation of the parametric formula in (5.13) for a fixed reliability of 0.98 for all leaf-tasks involved and the optional task $T_{19.1}$ selected ($OPT\_T_{19.1} = 1$) results in the following probability for the fulfilment of G_2:

$$P_{reach}(\varphi_2) = 0.86812553324672$$

5.5 Evaluating leaf-tasks reliabilities

As have been demonstrated in the previous sections regarding design-time and runtime analysis, the reliability values for the leaf-tasks involved in the quantitative verification of a CRGM must be set. In this section, we present two possible approaches for evaluating leaf-tasks reliabilities. The purpose is to illustrate, based on previous approaches, how a precise quantitative verification could be achieved by the high-level DTMC generated by our proposed goal-oriented dependability analysis.

5.5.1 Model-based verification

Leaf-tasks are not necessarily atomic system operations and are generally described with a high abstraction level. 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 activity and the corresponding system operation(s) becomes less evident. For instance, the MPERS task 'Find active sensors' could be further decomposed in more granular and concrete tasks according to the platform, architecture and language used for implementation.

As has been described in Section 2.2, TROPOS methodology defines additional phases of architecture design and detailed design. This last phase involves the specification of

the internal behaviour of leaf-tasks through UML activity and sequence diagrams. In previous works, UML behaviour models serve as input for the generation of a DTMC model representing system activities in terms of components interactions [13, 31]. In a similar approach, leaf-tasks can have their behaviour specified at TROPOS detailed design phase, enabling their individual reliability verification through PMC.

5.5.2 Mean-time to failure verification at runtime

The original RGM proposal relies on the assumption that the instance states of goals and tasks can be monitored. In fact, if concrete system operations can be traced back to the leaf-tasks in a RGM, i.e., if an execution trace can be generated indicating the state in which the leaf-tasks instances have been, different statistical measures could be calculated for all the monitored system tasks. For example, the mean-time to failure (MTTF) can be calculated as the average time between failures of a task, i.e., between transitions to their failure state. Disregarding the time to repair (non-repairable system), MTTF defines the reliability of the monitored leaf-tasks.

Capítulo 6

Geração Automática de Código DTMC

In the probabilistic verification adapted by this proposal, a behavioural specification is manually converted to a probabilistic model in PRISM language. However, this tends to be a costly and error-prone process which might be proportionally complex to the number of components, actions and interactions causing state transitions. By traversing a goal model from strategical root goal to operational leaf-tasks, a behaviour specification as proposed by the RGM serves as input for the generation of a DTMC model in PRISM language. Additional context effects in a CRGM may also be considered, as we have demonstrated in Chapter 5. However, the manual conversion from a CRGM to a DTMC model is still a costly and error-prone task.

To tackle this problem, an automated generation of the DTMC model in PRISM and PARAM languages have been implemented based on an existing JAVA open source tool supporting TROPOS development methodology named TAOM4E [27]. TAOM4E provides a graphical environment for goal modelling with TROPOS methodology based on the Eclipse Modelling Framework (EMF) and Graphical Editing Framework (GEF). Moreover, it also supports a model driven agent code generation. The CRGM to DTMC code generator have also been implemented as an Eclipse plugin integrated to the goal modelling environment provided by TAOM4E tool.

The purpose of our CRGM to DTMC automated code generation is to optimize the formal verification step by abstracting the probabilistic modelling know-how from the analysts and reduce the overhead and errors caused by the manual generation of the verification model. 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 behaviour and its different contexts of operation. The integration of our CRGM to DTMC generation to the TAOM4E environment forms a tool chain for a goal-oriented dependability analysis.

6.1 Architecture

The CRGM to DTMC code generator have been designed as a modular plugin for the Eclipse platform. Figures 6.1 presents the reference architecture used by the CRGM-toDTMC plugin.

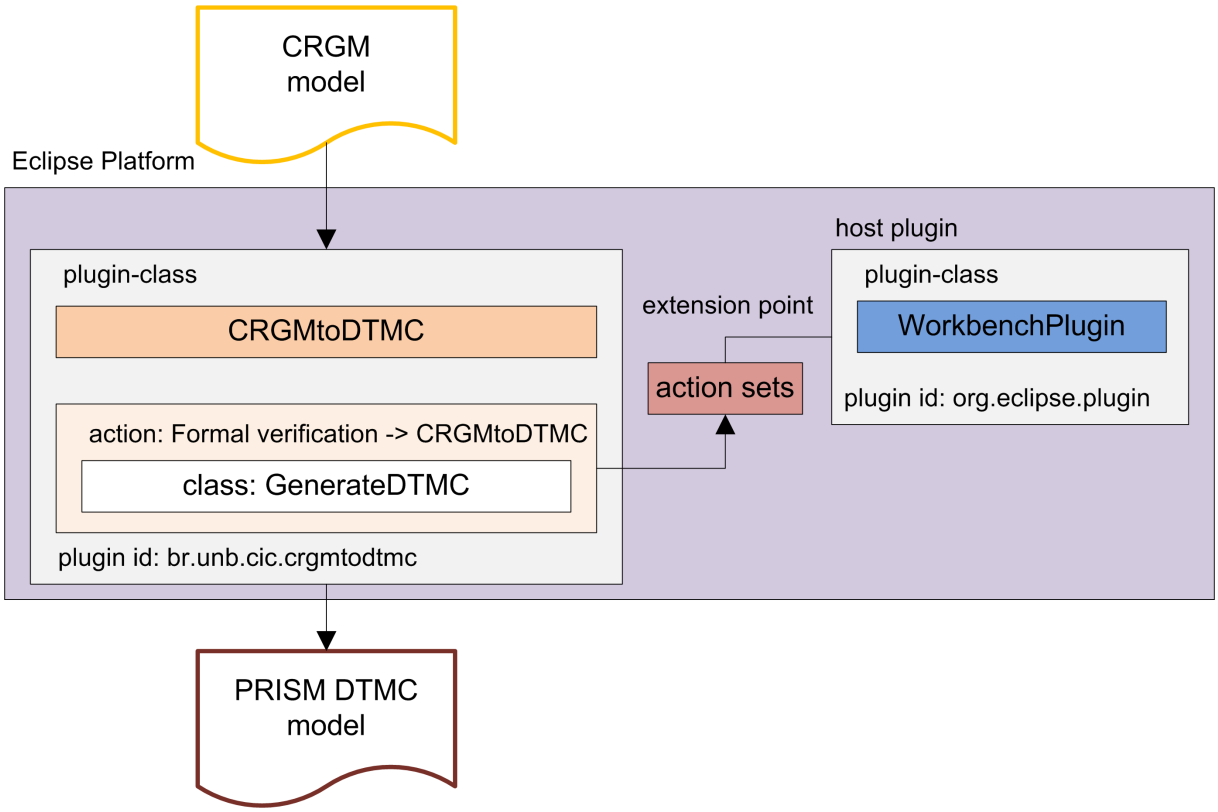


Figura 6.1: High-level architecture of the CRGMtoDTMC generator.

6.2 Implementation

The CRGM to DTMC implementation consists four major packages: the CRGM to DTMC (1), the TAO4ME model (2), the ANTLR regex (3) and the Eclipse Plugin SDK (4). Figure 6.2 depicts the implementation architecture for the code generator.

The CRGM to DTMC code generation process was divided in two phases:

1. **Parsing phase:** the TROPOS input file containing a goal model within a system actor is parsed using a depth first algorithm. Container objects for goals and tasks are kept in memory with any relevant metadata. The behaviour annotations in non-leaf elements and the context effects in any goal or task are also parsed by ANTLR generated APIs for the grammars in Listings 6.1 and 6.2. Both behaviour rules and

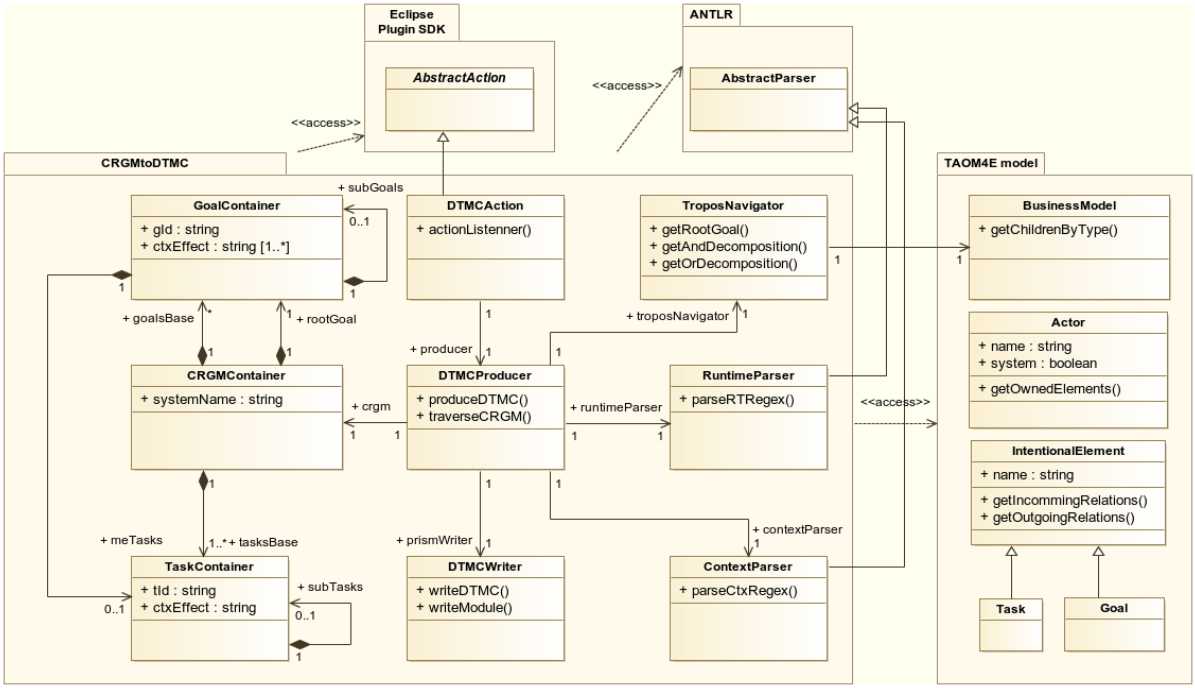


Figura 6.2: Implementation architecture of the CRGMtoDTMC generator.

context effects in a decomposition at any tree level must reach the leaf-tasks in the end of the related branches.

2. **Writing phase:** the container objects with all information required for the generation of a DTMC model are traversed from the root goal container. Leaf-task DTMC modules are created and concatenated to a global variable. For each goal G_i in the model, a success formula declared as G_1 is also concatenated to the model. These formulas can be part of the Probability existence PCTL properties specifying the probability in fulfilling one or more system goals, which eliminates the effort of manually building such formulas. After processing all context effects, only one declaration of the context variables parsed from context conditions associated to goals and tasks in the CRGM are appended to the model. Finally, the DTMC model file output is generated in a folder defined in the preferences section of the modelling environment with the agent's name.

The decision to divide the generation process in two phases was due to the complexity of parsing the goal model with additional behaviour rules and context effects. The two phases process has successfully divided different aspects of the code generation: parsing the input CRGM and writing the output DTMC. Consequently, the cohesion of each class has been preserved and future implementations could generate different types of output model.

The classes in Figure 6.2 interact as follows:

1. A user clicks on a specific menu item in the TAOM4E goal modelling tool, namely the Generate DTMC model in the Formal Verification menu.
2. The action listener in the $DTMCAction$ singleton receives the TROPOS goal model environment context and creates a new thread for the $DTMCProducer$, passing the TROPOS input file from the modelling context.
3. The $DTMCProducer$ instantiates the $TroposNavigator$ with the TROPOS input file.
4. The $TroposNaviagator$ accesses the business model from the TAOM4E model plugin and retrieves the actors in the input file.
5. The system actor is identified and its root goal extracted.
6. The $DTMCProducer$ starts the depth-first algorithm by the root goal calling the $addGoal()$ method.
7. Each goal or non-leaf task have their children elements extracted and saved as containers in the $CRGMDefinition$ class instance before a recursive call to $addGoal()/addTask()$:
   - Behaviour annotation are parsed by the $RTParser$ and the corresponding attributes setted in the child elements containers.
   - All context effects in goals and tasks are parsed by the $CtxParser$ and the corresponding attributes are setted in the child elements containers.
   - A recursive call to $addGoal()/addTask()$ is performed.
8. The recursive call return of a given child element is added to the parent element attributes to propagate nested time increments to subsequent goals/tasks.
9. The $DTMCDefinition$ file with all goals and tasks and a reference to the root goal is passed to the $writeModel()$ method in $DTMCWriter$ class.
10. PRISM language patterns for the creation of the DTMC model are loaded from files in the Eclipse Plugin package.
11. A depth-first algorithm traverses the transient container objects starting from the root goal.
    - Leaf-tasks have their modules created by replacing leaf-tasks attributes for time, context effects and other behaviour particularities in the corresponding PRISM language patterns.

- Variables for optional and alternative behaviours are appended to the model before corresponding leaf-tasks modules.
- Each leaf-task module creation call returns the boolean formula for its own success.
- Goals and non-leaf tasks success formulas are created by concatenating the formulas returned by recursive calls with logical PRISM language operators according to the decomposition type of the current root element.
- Context effects in goals and tasks have their variables type/value pairs stored in a set collection with no duplications.

12. The output DTMC file is opened.
13. The DTMC model header is written.
14. Context variables pairs are written as unsigned variable declarations.
15. The leaf-tasks modules, goals success formulas and additional alternative/optional unsigned variables are written.
16. The output DTMC file is closed.

6.2.1 ANTLR Grammars

Listing 6.1 and 6.2 present the grammar for the RGM behaviour rules and for the context effects, respectively. In both cases, a recursive definition of the *expr* parser rule encompasses all syntax alternatives and lexer rules define the tokens and fragments used by each grammar.

```
1  grammar RTRegex;
2  rt:       expr NEWLINE                          # printExpr
3     |   NEWLINE                                   # blank
4     ;
5
6  expr: expr op=('+'|'%'|'@') expr                 # gCard
7     |   expr op='|' expr                          # gAlt
8     |   'opt(' expr ')'                           # gOpt
9     |   'try(' expr ')' '?' expr ':' expr         # gTry
10    |   expr op=(';'|'#') expr                    # gTime
11    |   SKIP                                      # gSkip
12    |   GID                                       # gId
```

```
13     |  FLOAT                                  # n
14     |  '(' expr ')'                           # parens
15     ;
16
17 GID       : [GT] FLOAT      ;
18 FLOAT     : DIGIT+'.'?DIGIT*;
19 SEQ       : ';'             ;
20 INT       : '#'             ;
21 C_SEQ     : '+'             ;
22 C_INT     : '%'             ;
23 C_RTRY    : '@'             ;
24 ALT       : '|'             ;
25 SKIP      : 'skip'          ;
26 NEWLINE   : [\r\n]+         ;
27 WS        : [\t]+ -> skip   ;
28
29 fragment
30 DIGIT : [0-9]               ;
```

Listing 6.1: ANTLR grammar for the RGM behaviour rules.

```
1  grammar CtxRegex;
2  ctx:     expr NEWLINE          # printExpr
3    |      NEWLINE               # blank
4    ;
5
6  expr:    expr op='<' expr      # cLT
7     |   expr op='<=' expr       # cLE
8     |   expr op='>' expr        # cGT
9     |   expr op='>=' expr       # cGE
10    |   expr op='=' expr        # cEQ
11    |   expr op='!=' expr       # cDIFF
12    |   expr op='&' expr        # cAnd
13    |   expr op='|' expr        # cOr
14    |   BOOL                   # cBool
15    |    VAR                   # cVar
16    |    FLOAT                 # cFloat
17    |    '(' expr ')'          # cParens
18    ;
19
20 BOOL      : [false|true]            ;
```

```
21 VAR       : ('a'..'z'|'A'..'Z'|'_')+ ;
22 FLOAT     : DIGIT+'.'?DIGIT*         ;
23 NEWLINE   : [\r\n]+                  ;
24 WS        : (' '|'\t')+ -> skip      ;
25
26 fragment
27 DIGIT     : [0-9]                    ;
```

Listing 6.2: ANTLR grammar for the context effects in CGM.

Capítulo 7

Avaliação

In this chapter, qualitative and quantitative aspects of our proposal are evaluated. First, we focus on the CRGM model and on the automatic generation process. Second, we focus on the PRISM DTMC model itself. Third, we focus on the parametric formula generated by PARAM. Our evaluation is performed with the MPERS CRGM presented in Chapter 5.

7.1 Goal Question Metric

As we aim to integrate our approach to a software development methodology (SDM), questions related to the time of the analysis emerges. Time to market is an important factor in the adoption of a SDM and the analysis overhead must be justified by its benefits. Also, the scalability of the verification may become a threat to its validity. The space consumed by the generated verification model and by the parametric formula as well as the time consumed in the verification and the generation of the parametric formula from this model are technical concerns addressed by this evaluation.

The Goal Question Metric (GQM) method provides a systematic structure to guide the evaluation process [6]. GQM is a top-down method to define measures tied to the organizational context and goals. The idea is to first state formal *measurements goals*, then ask questions that must be answered in order to meet the goals and finally define the metrics that could answer the questions.

As an initial GQM step, Table 7.1 summarizes the quality goal aspects of this evaluation:

Purpose	Evaluate
Issue	the scalability of the
Object	goal-oriented dependability analysis
Viewpoint	requirements engineer/application engineer
Context	MPERS CRGM

Tabela 7.1: Definition of the evaluation objectives for the CRGM.

Regarding each of the model artefacts and generations processes, the following questions and metrics have been defined:

- **Q1**: What is the time-space complexity of the automatic DTMC generation as a function of the number of leaf-tasks?
  - M1.1: The generation time.
  - M1.2: The size of the DTMC model file.

- **Q2**: What is the time-space complexity of the reliability verification as a function of the number of interleaved leaf-tasks?
  - M2.1: The memory consumed by the fixed reliability verification.
  - M2.2: The time consumed by the fixed reliability verification.
  - M2.3: The time consumed by the generation of the parametric formula.

- **Q3**: What is the time-space complexity of the parametric formula evaluation as a function of the number of interleaved leaf-tasks?
  - M3.1: The size of the parametric formula.
  - M3.2: The time consumed by the parametric formula evaluation.

7.2 Evaluation Scenario

Once the GQM has been defined, the evaluation scenario is described. The MPERS used in this work is based on the functional requirements as industrially advertised. The evaluation of the GQM metrics is performed over the MPERS requirements modelled and analysed in the extended TAOM4E tool environment to produce a CRGM.

Regarding the behaviour rules inherited from the RGM proposal, the MPERS CRGM specifies all but one rule regarding the interleaved cardinality. Nonetheless, a similar behaviour is achieved by the interleaving of different tasks that exist in the MPERS

CRGM. Context effects have been specified to the extent of goals, which are enabled in specific context conditions, and tasks, whose *adoptability* is restricted to specific context conditions. The context space regarding a mobile system may include several other effects on goals and tasks. However, we considered the context selection for the MPERS CRGM a viable starting point for the scalability evaluation of the inclusion of context effects in the proposal.

The final CRGM model has its space dimension characterized as follows:

- **Number of goals:** 28
- **Number of tasks:** 49
- **Number of leaf-tasks:** 28
- **File size:** 468.674 kbytes

In our proposal, the scope of the dependability verification is defined by the target goal G_i. Thus, from the complete MPERS CGM in Figure 4.2, we have defined groups of goals according to the number of leaf-tasks they contain and how many of these tasks are interleaved to each other. The verification of systems with interleaving behaviour through model checking leads to the well known state explosion problem [3]. For this reason, metrics in the GQM affected by the interleaving of leaf-tasks are evaluated for each of these groups. Table 7.2 presents the evaluation groups. Nested interleaving rules in the RGM tree are summed, which explains the interleaved tasks number in some groups to be higher than the number of leaf-tasks in these groups.

Goal(s)	**|Leaf-tasks|**	**|Interleaved tasks|**
G5	3	0
G1	7	0
G2-4	18	11
G1-2	14	14
G1-3	19	24
G1-4	25	36
G0	28	39

Tabela 7.2: Different groups of goals involved in the evaluated of the metrics in the GQM.

The measurements in this evaluation have been collected with the PRISM model checker version 4.0.3-linux64 or the PARAM tool version 2-2-64α running in a personal

computer with dual-core/4-threads Intel i5 2.40GHz processor, 8GB DDR3-800 1066 MHz memory and Linux Ubuntu 14.04 operational system.

7.3 Results and Analysis

Following the GQM and the evaluation scenario, the measurements have been performed. Next subsections present each of the metrics in the GQM as well as their analysis.

7.3.1 DTMC generation and model

The automatic generation of a DTMC from the MPERS CRGM has its time complexity defined by the depth-first search algorithm employed by the implementation. Accordingly, the time complexity for the automatic generation of a DTMC model from a CRGM $CR_M = (I, R, rt_annot(), ctx_annot())$ is proportional to the complexity of the depth-first algorithm, which is linear to the number of nodes in the tree, or $\Omega(|I|)$. The automatic generation time for different evaluation groups is presented by Table 7.3.

Leaf-tasks	**Mean (ms)**	**S.D. (ms)**
3	7.43303	1.7224
7	18.8	3.96232
18	24	7.43303
28	15.33333	2.06559

Tabela 7.3: DTMC generation time (M1.1)

Regarding the values for the automatic DTMC generation time, we found that the time consumed by the JAVA implementation for an increasing number of leaf-tasks did not present a clear tendency of growth, as the average time consumed by the last generation involving 28 leaf-tasks was actually lower than the average time consumed for the generation of 18 leaf-tasks. We consider that an evaluation involving a higher magnitude of leaf-tasks should point out the linear growth tendency more precisely.

The DTMC model for a goal $G_i \in I$ contains $|\Lambda|$ modules for each of the leaf-goals $\epsilon \in \Lambda$. Thus, the textual size of a DTMC model for G_i is linearly proportional to $|\Lambda|$, as presented by the curve in Figure 7.1.

The measurements for the DTMC model generation and for the textual model size indicate the scalability of the proposed CRGM to DTMC automatic generation.

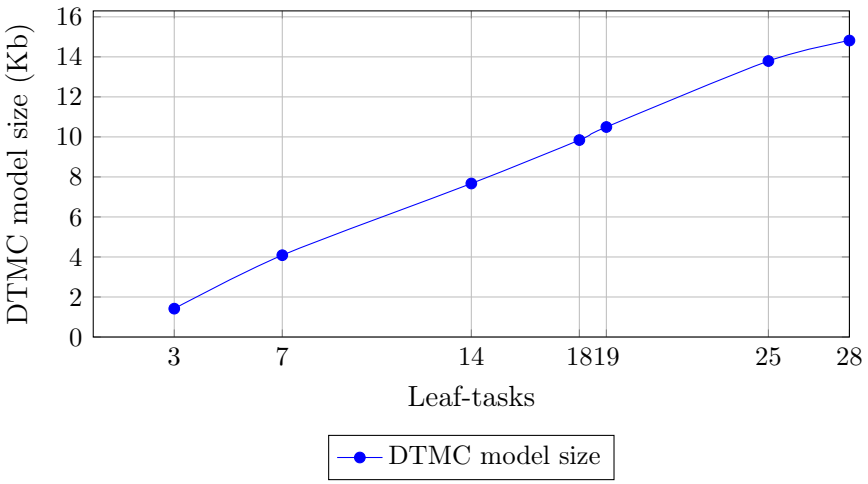


Figura 7.1: DTMC model size (M1.2)

7.3.2 Reliability verification

The time-space complexity of the dependability verification performed by PRISM and PARAM tools have been evaluated for each of the evaluation groups in Table 7.2. The plots in Figures 7.2, 7.3 and 7.4 present the values for the memory and time consumed by the design-time PRISM verification and the time consumed for the parametric formula generation by PARAM as a function of the number of interleaved leaf-tasks. As predicted, the metric represented by these figures grow exponentially with the number of interleaved leaf-goals verified.

A relevant fact observed in the design-time PRISM verification results for both performance and memory consumption is the decrease between points 11 and 14. A possible explanation for this discrepancy may be found in the number of leaf-tasks in both cases: at 11, 18 leaf-tasks are verified, while 14 leaf-tasks are verified at point 14, as described by the evaluation groups in Table 7.2. Despite the count of 18 interleaved leaf-tasks at the second point to be considerably higher than the 11 interleaved leaf-tasks in the first point, the methodology to obtain these numbers included the repetition of interleaved order in nested branches of the CRGM tree and disregarded the total number of sequential leaf-tasks. A more sophisticated approach considering both the number of interleaved and sequential leaf-tasks could result in a better configuration for the for the definition of the horizontal axis of the results.

The exponential complexity of the dependability verification in the case of interleaved behaviour could potentially indicate the infeasibility of the GODA approach for systems with more than a certain threshold of interleaved tasks, as well ground on the literature. Also, this certainly depends on the processor ability to concurrent executions. In our case, this threshold was close to 35 interleaved tasks.

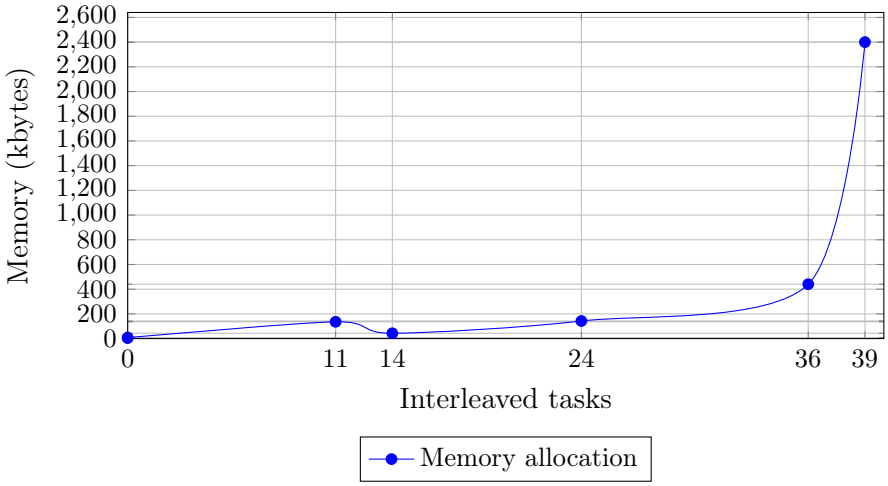


Figura 7.2: Memory allocated by design-time PRISM verification (M2.1)

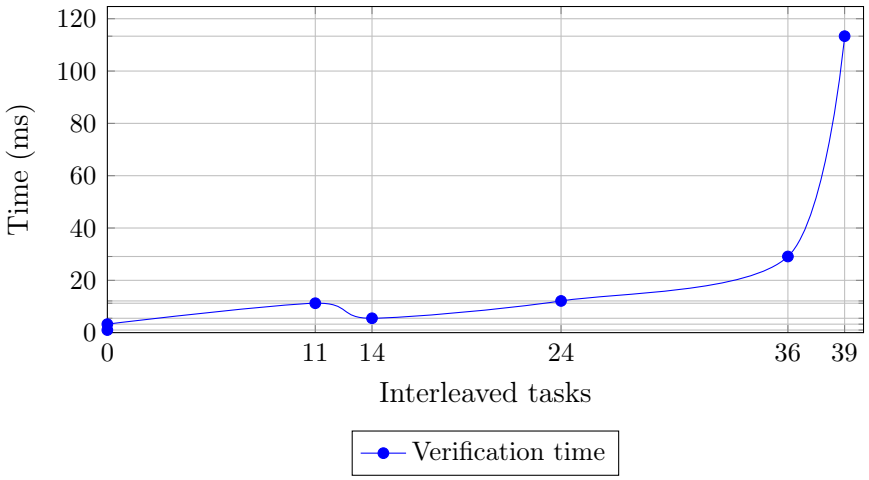


Figura 7.3: Design-time PRISM verification time (M2.2)

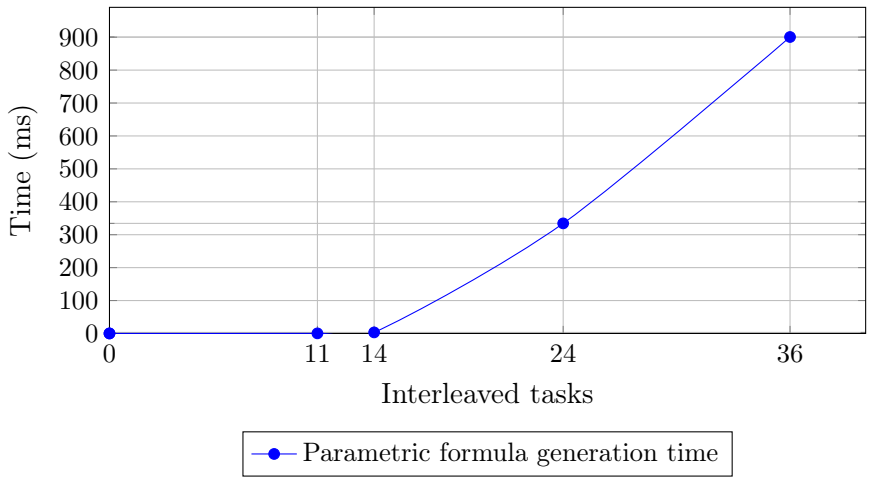


Figura 7.4: Parametric formula generation time (M2.3)

As it has been proposed in previous works [28], heuristics for how the system behaviour should be modelled and verified can also be used by our proposal. In particular, by verifying two interleaved branches of the tree separately and multiplying the results generates the exact same result of the one instance verification of the two branches. For example, a non-factorized formula for the root goal may be composed of the multiplication of formulas for underlying interleaved subgoals, which are smaller and much more efficient to obtain, store and evaluate.

7.3.3 Parametric formula evaluation

The time for the formula evaluation consists on the time for the arithmetic expression to be solved. Thus, its time complexity is proportional to the number of operations and the type of each operation. Once the space of the states and transitions in the DTMC model representing a goal G_i have been checked for the reliability in fulfilling this goal, the evaluation time of the parametric formula should grow at a similar rate of the formula size. Figures 7.5 and 7.6 do confirm this hypothesis. Also, Figure 7.6 indicates that the formula size grows exponentially with the increase in the number of interleaved tasks.

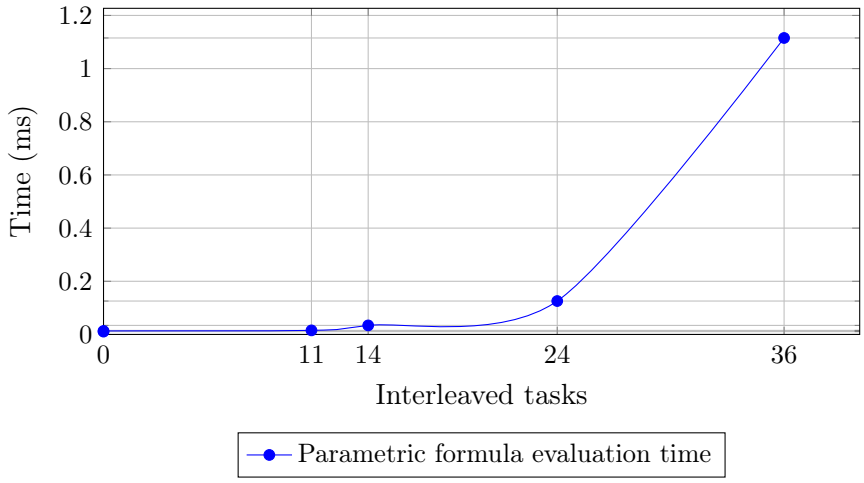


Figura 7.5: Parametric formula evaluation time (M3.1)

7.4 Threats to Validity

Some factors can be considered as threats to validity of the evaluation presented in this paper. They have been structured in four different viewpoints: the construct validity, that concerns the correctness of the experiment design, including the questions, concepts and metrics examined; the internal validity, that defines the correctness of the causal relationship between facts observed and the conclusions they lead to; the external validity,

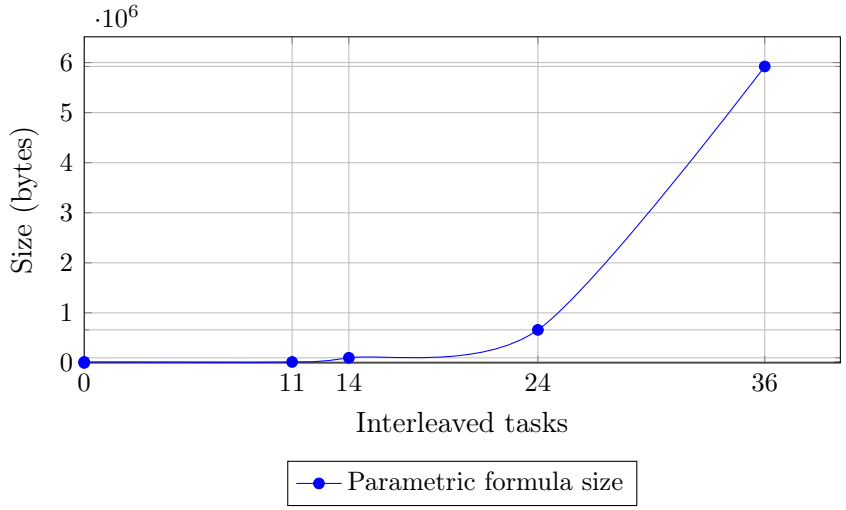


Figura 7.6: Parametric formula size (M3.2)

that concerns the domain from which the results may be generalized; the reliability, that tackles the experiment reproducibility.

Next, the particular threats to validity of this work are detailed.

- : **Construct validity:**
  - The GQM method mitigates a potential mislead definition of the metrics involved in the evaluation of the proposed goal-oriented dependability analysis. However, the selection of the evaluation groups composed of different goals in the MPERS CRGM was focused on the number of interleaved leaf-tasks in each group.

- : **Internal validity:**
  - The MPERS CRGM for which this experiment has been designed encompasses all but one behaviour rule for the interleaved cardinality. Despite this rule being a particular case of the Interleaved Order rule, it may have some unexpected influence on the results. Nonetheless, we have been able to analyse the rules for sequential and interleaved orders, alternative, optional and conditional behaviour, as well as sequential cardinality and the retry behaviour.

- **External validity:**
  - The MPERS CRGM, which this evaluation was based on, have been modelled and analysed referencing the diagrams and the advertise of similar products

provided by the industry. Despite the evaluation with a single system CRGM, our proposal is not limited to any specific domain; In fact, it can be applied to any other domain as long as the system can be modelled as a CRGM.

  - The TROPOS methodology could be replaced by other frameworks for goal modelling and analysis, as our approach is not limited to a specific goal model syntax. However, we enforce that in our proposal we tackle the operationalization of goals as leaf-tasks, in contrast to goal models without any representation of system activities, for instance, the one defined by the i* framework.

- **Reliability:**

  - Despite the systematic preparation of the experiment environment, the measurements have been performed in a personal computer running an operational system with concurrent processes that could have affected the results at some unexpected degree.

Capítulo 8

Conclusão

Neste trabalho propusemos uma nova abordagem para a verificação de dependabilidade que reflite os requisitos de alto nível especificados por um modelo de objetivos. Em particular, focamos na verificação de confiabilidade de um CRGM utilizando a técnica de verificação probabilística de modelos.

Considerando nossas quatro questões de pesquisa, as principais conquistas alcançadas podem ser sumarizadas como se segue: primeiro, a viabilidade técnica de uma verificação probabilística sobre a satisfação de um ou mais objetivos de um RGM foi alcançada pelas regras de transformação propostas (QP1). Em segundo, a viabilidade da inclusão dos efeitos de contexto sobre objetivos e tarefas como especificados num CGM foi contemplada pela proposta (QP2). Em terceiro, a viabilidade da análise de dependabilidade orientada a objetivos como parte de um ciclo de auto-adaptação foi demonstrada pelo uso de uma verificação probabilística de modelos parametrizada , cuja fórmula resultante avalia a confiabilidade de diferentes alternativas em atingir objetivos diversos do sistema (QP3). Por último, mas não menos importante, nossa proposta contemplou com sucesso a geração automatizada de um modelo DTMC a partir de um CRGM, o que constitui uma importante contribuição para a usabilidade da proposta dado que reduz o custo e esforço da análise e diminui a incidência de erro humano nas atividades de modelagem (QP4).

Algumas melhorias relevantes para essa proposta e potenciais novas contribuições envolvendo a análise de dependabilidade orientada a objetivos constituem nosso trabalho futuro:

- **Aplicação e avaliação da proposta num caso de estudo:** como um trabalho futuro imediato, gostaríamos de aplicar o GODA a um caso de estudo real para o qual a dependabilidade seja um requisito de primeira classe e cujas restrições de recursos da plataforma móvel sejam consideradas.

- **Recompensas:** A estrutura de recompensas na linguagem PRISM pode ser usada para se estender o GODA com a verificação do consumo de recursos restritos, tal qual a bateria.

- **Verificação de confiabilidade e outras métricas relacionadas à dependabilidade:** A linguagem PCTL permite a especificação de uma ampla gama de propriedades que poderiam ser verificadas, incluindo propriedades restritas no tempo. Gostaríamos de explorar a análise de outras métricas e atributos de dependabilidade que possam ser verificados em tempo de execução.

- **Heurísticas para a escalabilidade:** A formalização de heurísticas para se diminuir a complexidade espaço-temporal da geração da fórmula paramétrica e sua avaliação, tal qual a composição de uma fórmula global a partir de fórmulas individuais referentes a ramos de execução em interleaving é um potencial trabalho futuro para a geração automatizada CRGM para DTMC.

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. 2, 11, 13, 21, 26

[2] 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. 14

[3] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. 3, 16, 37, 68

[4] 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.

[5] 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.

[6] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994. 66

[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. 1, 9, 10, 21

[8] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*, volume 5 of *International Series in Software Engineering*. Springer, October 1999. 20

[9] 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, 22, 26, 30, 35, 37

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

[11] Anthony Finkelstein 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. 11

[12] Ariel Fuxman, Lin Liu, John Mylopoulos, Marco Pistore, Marco Roveri, and Paolo Traverso. Specifying and analyzing early requirements in tropos. *Requir. Eng.*, 9(2):132–150, May 2004. 24

[13] Carlo Ghezzi and Amir Molzam Sharifloo. Model-based verification of quantitative non-functional properties for software product lines. *Inf. Softw. Technol.*, 55(3):508–524, March 2013. 16, 17, 58

[14] Lars Grunske. Specification patterns for probabilistic quality properties. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 31–40, New York, NY, USA, 2008. ACM. 53

[15] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Param: A model checker for parametric markov models. In *CAV*, pages 660–664, 2010. 6, 18

[16] S. Hallsteinsen, K. Geihs, N. Paspallis, F. Eliassen, G. Horn, J. Lorenzo, A. Mamelli, and G. A. Papadopoulos. A development framework and methodology for self-adapting applications in ubiquitous computing environments. *J. Syst. Softw.*, 85(12):2840–2859, December 2012. 2

[17] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994. 17

[18] Jennifer Horkoff and Eric Yu. Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requirements Engineering*, 18(3):199–222, 2013. 12

[19] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009. 16, 17

[20] M. Kwiatkowska and D. Parker. Advances in probabilistic model checking. In T. Nipkow, O. Grumberg, and B. Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 126–151. IOS Press, 2012. 16, 17

[21] Oxford University Computing Laboratory. PRISM case studies. http://www.prismmodelchecker.org/casestudies/, 2015. [Online; accessed 25-january-2015]. 16

[22] Oxford University Computing Laboratory. PRISM language manual. http://www.prismmodelchecker.org/manual/ThePRISMLanguage/Introduction, 2015. [Online; accessed 25-january-2015]. 17

[23] Oxford University Computing Laboratory. PRISM web site. http://www.prismmodelchecker.org/, 2015. [Online; accessed 25-january-2015]. 16

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

[25] 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. 7

[26] 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. 2, 22, 24

[27] Mirko Morandini, Duy Cu Nguyen, Anna Perini, Alberto Siena, and Angelo Susi. Tool-supported development with tropos: The conference management system case study. In *Proceedings of the 8th International Conference on Agent-oriented Software Engineering VIII*, AOSE'07, pages 182–196, Berlin, Heidelberg, 2008. Springer-Verlag. 6, 59

[28] 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. 2, 18, 72

[29] OnGuardHelp. OnGuardHelp Safety App. `http://www.onguardhelp.com/how-it-works/`, 2015. [Online; accessed 25-january-2015].

[30] Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007. 18

[31] Genaína Nunes Rodrigues, Vander Alves, Renato Silveira, and Luiz A. Laranjeira. Dependability analysis in the ambient assisted living domain: An exploratory case study. *Journal of Systems and Software*, 85(1):112 – 131, 2012. Dynamic Analysis and Testing of Embedded Software. 3, 17, 39, 58

[32] 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. 23, 24

[33] 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.

[34] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262, 2001. 1, 9, 21

[35] A. van Lamsweerde and L. Willemet. Inferring declarative requirements specifications from operational scenarios. *Software Engineering, IEEE Transactions on*, 24(12):1089–1114, Dec 1998. 20

[36] Eric Siu-Kwong Yu. Modelling strategic relationships for process reengineering. 1996. UMI Order No. GAXNN-02887 (Canadian dissertation). 9, 20

[37] E.S.K. Yu. Modeling organizations for information systems requirements engineering. In *Requirements Engineering, 1993., Proceedings of IEEE International Symposium on*, pages 34–41, Jan 1993. 1, 20

[38] 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. 4, 13