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

Análise do Impacto de Cenários Implícitos na Confiabilidade de Sistemas Computacionais

Alexandre Vaz Roriz

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Genaína Nunes Rodrigues

Brasília
2015

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

Coordenadora: Prof.ª Dr.ª Alba C. M. A. Melo

Banca examinadora composta por:

Prof.ª Dr.ª Genaína Nunes Rodrigues (Orientadora) — CIC/UnB
Prof. Dr. Nabor das Chagas Mendonça — UNIFOR
Prof. Dr. George Luiz Medeiros Teodoro — CIC/UnB

CIP — Catalogação Internacional na Publicação

Roriz, Alexandre Vaz.

Análise do Impacto de Cenários Implícitos na Confiabilidade de Sistemas Computacionais / Alexandre Vaz Roriz. Brasília : UnB, 2015. 111 p. : il. ; 29,5 cm.

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

1. cenários implícitos, 2. sistemas concorrentes, 3. dependabilidade, 4. confiabilidade

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

Análise do Impacto de Cenários Implícitos na Confiabilidade de Sistemas Computacionais

Alexandre Vaz Roriz

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

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

Prof. Dr. Nabor das Chagas Mendonça
UNIFOR

Prof. Dr. George Luiz Medeiros Teodoro
CIC/UnB

Prof.ª Dr.ª Alba C. M. A. Melo
Coordenadora do Mestrado em Informática

Brasília, 6 de abril de 2015

Dedicatória

Dedico esse trabalho especialmente à minha pequena Clarisse, que me acompanhou na reta final desse trabalho, e também à minha família que compreendeu a minha ausência durante o período de dedicação a esse Mestrado.

Agradecimentos

"Combati o bom combate, terminei a minha carreira, guardei a minha fé" (II Timóteo 4:7)
Em primeiro lugar, não poderia ser de outra forma, agradeço a Deus que me acompanhou nessa caminhada participando tanto dos meus momentos de descontração, como nos de intensos trabalhos, sempre me dando forças para continuar.

Também agradeço à minha linda esposa Juliana, que sempre me apoiou, desde a decisão de iniciar essa jornada, com todo o carinho e amor que eu precisei. Ju, essa conquista é nossa!!!

À minha doce Clarissinha, tenho que agradecer por fazer, desde o dia nove de junho de 2014, os meus dias sempre especiais. Filhina, papai te ama!!!

À minha mãe, o agradecimento que não caberia nesse espaço. A grande responsável pelo que sou hoje. Mamãe, o seu exemplo de vida é a minha maior inspiração!!!

Aos meus irmãos, meu precioso afilhado Mateus e demais familiares, obrigado pelo carinho de sempre.

Aos meus colegas de curso, MUITO OBRIGADO!!! O clima de camaradagem permanente foi determinante para que eu chegasse até aqui.

Por último, mas não menos importante, à minha orientadora Prof.ª Genaína, meu mais sincero agradecimento. Sempre serei grato pela sua incansável dedicação e por nunca deixar de acreditar (até quando eu mesmo não acreditava!!) no sucesso do trabalho. Você se tornou uma referência, tanto como profissional, quanto como pessoa. De coração, MUITO OBRIGADO!!!

Resumo

O aumento da complexidade dos sistemas computacionais é uma tendência que precisa ser acompanhada pela busca por técnicas de desenvolvimento orientadas em garantir o nível de confiabilidade exigido, desde os estágios iniciais do ciclo de desenvolvimento de software. Nesse contexto, passa a ser requisito essencial que a modelagem seja feita de forma confiável, refletindo da melhor maneira possível o comportamento esperado para o sistema.

A modelagem em cenários possibilita, ainda nas fases iniciais do desenvolvimento, a realização de análises de dependabilidade para verificar o atendimento a requisitos não funcionais do sistema. Configurando então uma técnica interessante para a construção de sistemas confiáveis.

Entretanto, a modelagem em cenários traz a possibilidade da ocorrência de cenários implícitos, que são cenários que não estavam previstos inicialmente mas surgem da integração entre os diversos cenários especificados. Esses novos cenários podem ser desejados, quando não comprometem o sistema, ou indesejados, quando levam o sistema a um estado de erro. Nesse último caso, é exigida a tomada de ações corretivas para o modelo visando a eliminação ou atenuação desses efeitos negativos.

Sendo assim, para a construção de um modelo confiável, faz-se necessário não somente identificar os eventuais cenários implícitos, mas avaliar o impacto desses sobre o sistema a ser modelado, de modo a fundamentar e direcionar as ações para se alcançar o nível de dependabilidade adequado.

Esse trabalho apresenta uma metodologia para a avaliação quantitativa e qualitativa do impacto dos cenários implícitos sobre a confiabilidade de um sistema que está sendo modelado. Um estudo de caso é apresentado mostrando o emprego da metodologia para a definição de uma configuração mais confiável em um sistema de rede de câmeras inteligentes. Os resultados mostram que a metodologia é útil para a compreensão de como os cenários implícitos impactam a confiabilidade do modelo, fornecendo subsídios para a tomada de decisões que resultem em um sistema mais confiável.

Palavras-chave: cenários implícitos, sistemas concorrentes, dependabilidade, confiabilidade

Abstract

The increasing of the complexity in computing systems is a trend that needs be accompanied by the search for development techniques oriented to ensure the required reliability level, since the early stages of the software development cycle. In this context, it becomes a essential requirement that the modeling be done in a reliable way, reflecting the expected behavior for system.

The scenario-based specification allows, even at early stages of the development, the realization of dependability analysis to verify the compliance with no-functional requirements of the system. Being considered an interesting technique for constructing reliable systems.

However, the scenarios modeled brings the possibility of the occurrence of implied scenarios, which are scenarios that were not initially foreseen but arise from interaction between the others specified scenarios. These new scenarios can be allowed, when they do not compromise the system, or unwanted, when they take the system to an error state. In the latter case, it is required to take corrective action in order to eliminate or reduce these negative effects. Therefore, to construct a reliable model, it is necessary not only identify any implied scenarios, but also to assess the impact of these on the system to be modeled in order to ground and direct the actions in order to achieve the appropriate dependability level.

This work presents a methodology for quantitative and qualitative analysis of the impact of implied scenarios on the reliability of a system being modeled. A case study is presented showing the use of the methodology for defining a more reliable configuration in a network system of smart cameras. The results show that the method is useful for understanding how the implied scenarios impact the reliability of the model, providing support for decisions that result in a more reliable system.

Keywords: implied scenarios, concurrent systems, dependability, reliability

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O aumento da complexidade dos sistemas computacionais é uma tendência que precisa ser acompanhada pela busca por técnicas de desenvolvimento orientadas em garantir o nível de confiabilidade exigido, desde os estágios iniciais do ciclo de desenvolvimento de software. Nesse contexto, passa a ser requisito essencial que a modelagem seja feita de forma confiável, refletindo da melhor maneira possível o comportamento esperado para o sistema.

A modelagem por cenários tem sido considerada uma forma eficaz de se modelar e analisar o comportamento de um sistema ainda no início do seu ciclo de desenvolvimento [1]. Uma das vantagens desse formato é permitir uma participação maior dos *stakeholders*, dado que o comportamento do sistema será obtido da integração entre os diversos cenários de uso que refletem as funcionalidades desejadas. Os cenários são especificados utilizando o Diagrama de Sequência da UML (*Unified Modeling Language*) [2] ou diagramas MSCs (*Message Sequence Charts*) [3] e apresentam as trocas de mensagens entre os componentes do sistema, caracterizando sua arquitetura.

São várias as abordagens que utilizam a modelagem em cenários para realizar análise de confiabilidade de software [4–6]. Entretanto, estes trabalhos, não têm, até aqui, considerado nesta análise o impacto de um fenômeno denominado cenários implícitos. Cenários implícitos, que são, de forma breve, comportamentos do sistema que surgem a partir da composição paralela[1] dos componentes, cujos comportamentos são modelados nos cenários previamente especificados [7, 8]. Isso ocorre pelo fato de cenários serem visões parciais do sistema modelado, apesar de se ter em mente um comportamento global ao se modelar esses cenários. Dessa forma, quando esses cenários são integrados podem surgir novos comportamentos que não haviam sido anteriormente previstos [8].

[1] Entende-se por composição o processo de construir a arquitetura do sistema a partir do comportamento de seus componentes, utilizando para tanto métodos formais e suas técnicas de composição em paralelo [9].

Esses novos cenários podem ser desejáveis, cujo efeito obtido é aceito e integrado ao modelo, ou indesejáveis, cujo efeito gera consequências negativas, podendo levar o sistema a falhas [8]. Uma análise de confiabilidade da arquitetura realizada sobre uma modelagem em cenários que não considera esse aspecto pode levar a uma percepção equivocada do comportamento real do sistema de software. Isso pode incorrer em erros detectados somente em ciclos tardios no ciclo de desenvolvimento de software, onerando o custo de correção. Ou ainda, detectando erros somente quando o sistema já estiver em produção.

1.1 Problema

Uma modelagem baseada em cenários que deixe de considerar a presença e o efeito dos cenários implícitos pode ter um efeito nocivo. A confiabilidade de uma modelagem baseada em cenários pressupõe a identificação dos eventuais cenários implícitos presentes e, consequente correção do projeto para eliminar ou reduzir os efeitos negativos ocasionados por eles. Entretanto, esse processo de correção envolve a tomada de decisões que devem ser baseadas, sobretudo, nas consequências da presença dos cenários indesejados para o sistema. É necessário saber não só informações quantitativas, como, por exemplo, a frequência de execução desse cenário, mas também qualitativas, como o impacto que o defeito ocasionado pode ter aos usuários.

1.2 Solução proposta

Frente a esse problema, a solução proposta é apresentar uma metodologia que possibilite avaliar o impacto que os cenários implícitos exercem tanto de forma quantitativa, como de forma qualitativa sobre a confiablidade de uma modelagem.

A Figura 1.1 apresenta, de forma simplificada, as etapas da metodologia proposta neste trabalho. Conforme é mostrado, parte-se da modelagem em cenários e com a ferramenta LTSA-MSC [31] se identifica os eventuais cenários implícitos negativos que serão o alvo das análises qualitativas e quantitativas. Na análise qualitativa, o domínio da aplicação é considerado para a correlação de possíveis defeitos do sistema, com a ocorrência dos cenários implícitos. Busca-se assim, dar um significado semântico para o que esses novos cenários representam e afetam a confiabilidade do sistema. Já na análise quantitativa, são realizados experimentos para verificar o quanto a confiabilidade do modelo é afetada quando esses cenários implícitos negativos são considerados.

A composição dessas duas vertentes (quantitativa e qualitativa) possibilita o levantamento de informações capazes de subsidiar com clareza as decisões a serem tomadas para a correção do modelo no caminho de alcançar o nível de confiança desejado.

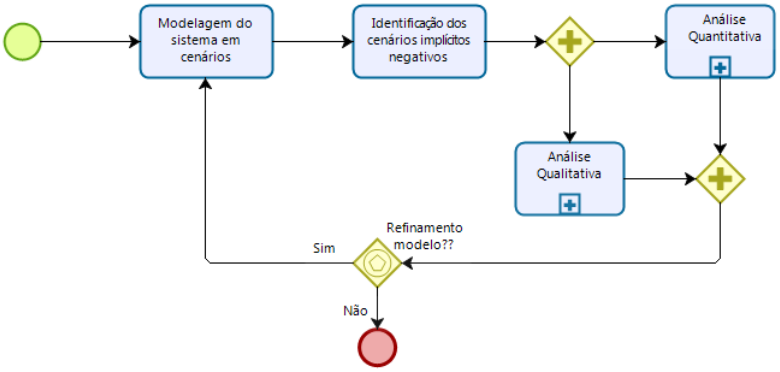


Figura 1.1: Metodologia proposta simplificada

1.3 Contribuições

Este trabalho oferece as seguintes contribuições:

- Estudo da natureza dos defeitos gerados por cenários implícitos sobre a confiabilidade;
- Identificação do impacto quantitativo que os cenários implícitos identificados podem causar na confiabilidade de um modelo;
- Orientação de refatoramento de modelos para a obtenção de maiores níveis de confiabilidade;
- Realização de estudo de caso em um simulador de câmeras inteligentes (*smart cams*).

Ademais, a metodologia proposta nesse trabalho foi apresentada na seguinte publicação:

- A. V. Roriz, G. N. Rodrigues, and L. A. Laranjeira, *"Analysis of the Impact of Implied Scenarios on the Reliability of Computational Concurrent Systems,"* In 2014 Eighth Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 2014, pp. 105–114.

1.4 Trabalhos relacionados

No campo de análise de confiabilidade, os trabalhos [10] e [11] estão bem relacionados com essa pesquisa. Visto que ambos abordam o cálculo da previsão de confiabilidade de um modelo gerado nos estágios iniciais do ciclo de vida de um sistema utilizando técnicas de verificação formal.

Em [10] a análise se deu a partir da modelagem do sistema em cenários utilizando diagramas pHMSC (*probabilistic high message sequence charts*), os quais se caracterizam por utilizar pesos probabilísticos em cada aresta do diagrama. Com a modelagem inserida em uma ferramenta de verificação formal, são realizados experimentos para o cálculo da confiabilidade geral do sistema sem considerar a presença dos cenários implícitos. Em seguida, realiza-se a busca por cenários implícitos e posterior classificação em positivo ou negativo e, calcula-se novamente a confiabilidade do sistema considerando a não ocorrência dos cenários implícitos negativos. Busca-se, assim, verificar o impacto quantitativo da presença de cenários implícitos sobre a confiabilidade do sistema. Todavia, não ocorre a análise qualitativa deste impacto.

Já em [11] a análise se dá a partir da modelagem do sistema utilizando diagramas comportamentais da UML (sequência e atividade). Em seguida o modelo é inserido na ferramenta de verificação formal PRISM, traduzindo os diagramas para a linguagem própria da ferramenta, possibilitando a realização de diversos experimentos a respeito da confiabilidade do sistema e de seus componentes. Nesse trabalho não foi realizada nenhuma análise envolvendo cenários implícitos.

No campo de cenários implícitos, Uchitel *et al.* [8] apresenta uma metodologia para a identificação de cenários implícitos a partir de uma modelagem em cenários. Essa metodologia é utilizada neste trabalho para a etapa de identificação de cenários implícitos negativos que serão submetidos às análises propostas. Em [31], Uchitel apresentou a ferramenta LTSA-MSC que automatiza sua metodologia.

Os trabalhos recentes desenvolvidos por Al-Azzani *et al.* [12–14] apresentam a utilização das técnicas de detecção de cenários implícitos propostas por Uchitel em prol da geração de casos de testes de segurança e avaliação arquitetural. Especificamente, [12] e [13] trataram da proposição de uma metodologia para a geração de casos de testes de segurança a partir dos cenários implícitos descobertos. Já em [14] é proposta uma integração das análises de cenários implícitos e condições de corrida, como forma de detectar e analisar sistematicamente vulnerabilidades relacionadas à segurança no nível de arquitetura.

No trabalho [15], os cenários implícitos são detectados utilizando engenharia reversa a partir dos *traces* de execução de um sistema em operação, para que possam ser obtidos novos casos de testes, eventualmente, ainda não explorados.

Esses quatro últimos trabalhos citados mostram aplicações do conhecimento da existência de cenários implícitos, possibilitando oportunidades para a utilização das técnicas de análise propostas na presente pesquisa.

1.5 Estrutura do documento

O restante do trabalho está organizado da seguinte maneira:

- No **Capítulo 2** são apresentados os principais pontos teóricos que serão utilizandos ao longo do trabalho. São introduzidos conceitos como Dependabilidade, *Model Checking* e Cadeias de Markov;
- Cenários implícitos e os seus principais fundamentos são abordados no **Capítulo 3**;
- No **Capítulo 4** é apresentada a metodologia para a análise do impacto dos cenários implícitos;
- O **Capítulo 5** traz o estudo de caso realizado compreendendo o emprego da metodologia proposta para a definição da arquitetura mais confiável de um sistema de rede de câmeras inteligentes.
- As considerações finais e propostas para trabalhos futuros estão presentes no **Capítulo 6**.

Capítulo 2

Fundamentação Teórica

Nesse capítulo serão apresentados os principais conceitos necessários para a compreensão do trabalho.

2.1 Falha, erro e defeito

O serviço entregue por um sistema é o comportamento dele percebido por seus usuários ou outros sistemas que se comunicam para a utilização de suas funcionalidades. Quando o comportamento do sistema difere do esperado, indicando que o serviço não foi entregue de forma correta, tem-se a ocorrência de um defeito no sistema (*failure*) [16].

Um defeito é a externalização de um estado de erro do sistema. A causa do erro é uma falha (*fault*) de algum componente ou rotina do sistema [17].

Uma falha que ainda não tenha provocado erro ao sistema, seria o exemplo de um algoritmo mal projetado que levaria o sistema a um estado de *dead-lock* se determinada combinação de dados for inserida, é denominada em [18] falha dormente (*dormant fault*). Se o sistema receber a entrada de dados que o leva ao estado de *dead-lock*, é dito que a falha foi ativada.

Em [18] são apresentados vários exemplos desse relacionamento *"falha* × *erro* × *defeito"*. Um desses exemplos é mostrado a seguir:

- Uma implementação mal sucedida de uma instrução, resulta na introdução de uma falha dormente no sistema; caso essa instrução seja utilizada em determinada execução, a falha se torna ativa e produz um erro. Caso o erro afete o serviço entregue pelo sistema, seja alterando o conteúdo correto de uma informação fornecida ou atrasando sua entrega, um defeito estará ocorrendo.

Cabe destacar que a tradução para o português dos termos *failure* e *fault* é uma questão ainda aberta na comunidade. Alguns autores nacionais traduzem *failure* como

falha e *fault* como falta. Nesse trabalho, como pode ser notado, optou-se pela tradução proposta em [19], na qual *failure* é traduzido para defeito e *fault* para falha. O argumento é que se fosse para utilizar a tradução alternativa, a área de pesquisa de tolerância a falhas deveria se chamar tolerância a faltas, pois *failures* não podem ser toleradas, mas sim as *faults*.

2.2 Dependabilidade

O termo dependabilidade, tradução para *dependability*, é definido em [20], como a habilidade de um sistema oferecer um serviço pelo qual se pode justificadamente confiar. Essa definição destaca a necessidade de se justificar o nível de confiança do sistema. Em [18] é apresentada uma definição alternativa, mais direta e didática, na qual dependabilidade de um sistema é a habilidade de evitar que seus serviços tenham defeitos mais frequentes e severos que o aceitável. Nesse caso, é apresentado um critério para que um sistema seja considerado dependável, tradução para *dependable*.

Ainda em [18], dependabilidade é apresentado também como um conceito que integra os seguintes atributos de qualidade de software:

- Disponibilidade (*availability*): prontidão para prover um serviço correto;
- Confiabilidade (*reliability*): continuidade da prestação de serviços corretos por um período de tempo;
- Segurança crítica (*safety*): ausência de consequências catastróficas para o usuário e ao ambiente;
- Integridade (*integrity*): ausência de alterações indevidas no sistema;
- Manutenibilidade (*maintainability*): facilidade para se modificar ou reparar o sistema.

Para que um sistema possa alcançar um determinado nível de dependabilidade devem ser utilizados, de forma combinada, um conjunto de métodos e técnicas divididas nas seguintes categorias [19, 20]:

- Tolerância a falhas: buscam fornecer o serviço esperado mesmo na presença de falhas;
- Remoção de falhas: buscam a verificação da presença e remoção de falhas;
- Prevenção a falhas: buscam impedir a ocorrência ou introdução de falhas;

- Previsão de falhas: buscam realizar estimativas sobre presença e conseqüências de falhas.

A necessidade ou não da aplicação dos métodos e técnicas de cada uma das categorias é resultado de uma análise de dependabilidade a ser efetuada sobre o sistema. Essa análise é realizada sobre a necessidade de cada atributo. Nesse trabalho, a metodologia proposta avalia o atributo confiabilidade e, por ser utilizada no início do ciclo de desenvolvimento do sistema, utilizará, preponderadamente, técnicas de previsão de falhas.

2.3 Análise de dependabilidade

Várias técnicas de análise de dependabilidade têm sido desenvolvidas nos últimos anos. Essas técnicas são usadas para a previsão, verificação e melhoria dos atributos de dependabilidade definidos pelos requisitos do sistema. Elas ajudam a responder questões como: Um sistema de controle de vôo está habilitado para tolerar um determinado número de falhas simultâneas nos equipamentos? Quando ocorrer uma interrupção, em quanto tempo o sistema estará recuperado? O sistema está habilitado para prover o serviço correto durante um determinado período de tempo? Entre outras [21].

Neste trabalho, as análises de dependabilidade utilizarão técnicas de previsão e serão baseada em modelos, implicando em utilizar um modelo[1] do sistema para avaliar a previsão das propriedades desejadas. Particularmente, será utilizado um modelo probabilístico empregando a técnica de *model checking*, que é uma técnica que explora todos os estados possíveis do sistema para verificar determinadas propriedades, que refletem os requisitos do sistema [23].

O modelo probabilístico será composto utilizando Cadeias de Markov [23] e as propriedades a serem avaliadas serão descritas na linguagem PCTL (*Probabilistic Computational Tree Logic*) [24]. Para a realização do *model checking*, será utilizada a ferramenta PRISM [25], a qual possibilita verificar o atendimento das propriedades. A Figura 2.1 apresenta uma síntese desse processo.

2.3.1 Cadeias de Markov

De modo resumido, Cadeias de Markov são modelos nos quais os sistemas são representados por uma sequência de estados em que cada transição recebe uma probabilidade para seguí-la. Essa probabilidade depende apenas do estado atual. Se o parâmetro de

[1] Modelo de um sistema é uma abstração com intuito de compreender o seu comportamento antes da sua construção [22].

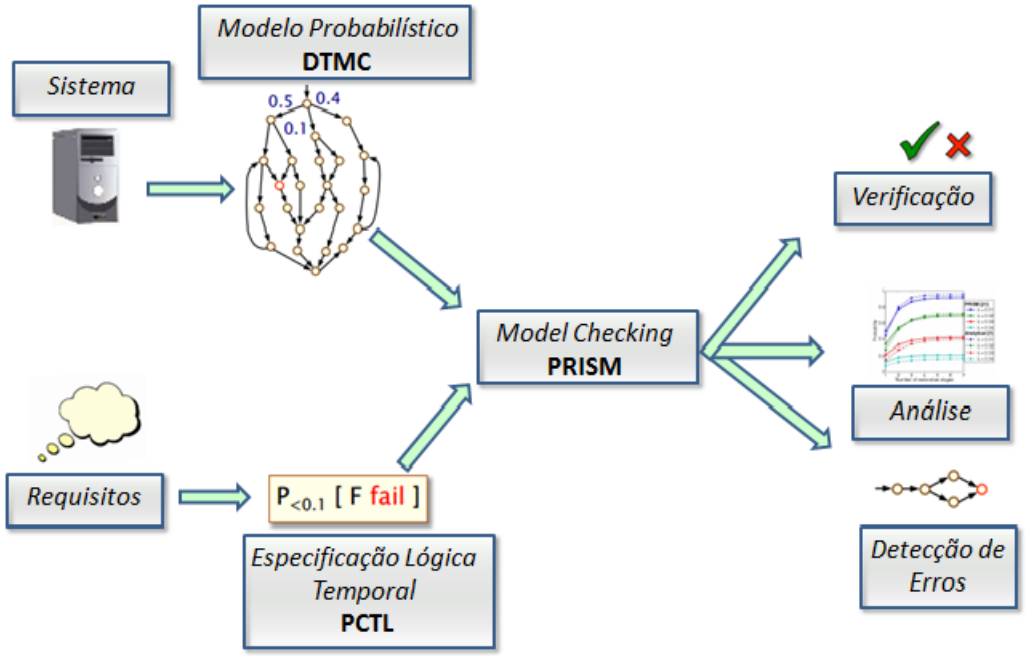


Figura 2.1: Processo de *model checking* - Figura adaptada de [23]

tempo for discreto, o modelo é uma DTMC (*discrete-time Markov Chain*), caso seja contínuo, será uma CTMC (*continuous-time Markov Chain*). Neste trabalho, serão utilizadas apenas DTMCs.

Uma Cadeia de Markov de Tempo Discreto (DTMC) é definida formalmente como uma tupla $M = (S, \mathbf{P}, i_{init}, AP, L)$, onde [23]:

- S é um conjunto de estados finitos;
- $\mathbf{P}$: $S \times S \rightarrow [0, 1]$ é uma função de transição na qual todos os estados s satisfazem:

$$\sum_{s' \in S} \mathbf{P}(s, s') = 1$$

- $i_{init} : S \rightarrow [0, 1]$ é a distribuição de probabilidades inicial, onde $\sum_{s \in S} i_{init}(s) = 1$;
- AP é um conjunto de proposições atômicas relativas aos estados do sistema;
- $L : S \rightarrow 2^{AP}$ é uma função que relaciona os estados do sistema às proposições atômicas.

Na Figura 2.2 é apresentado um exemplo, presente em [23], que representa uma Cadeia de Markov de Tempo Discreto modelando um protocolo de comunicação fictício no qual a probabilidade de uma mensagem ser enviada com sucesso é 90%. Nesse protocolo, quando a mensagem não chega ao seu destino correto, há uma nova tentativa de envio. Analisando

o modelo, verifica-se que no estado inicial *"start"*, a mensagem é gerada e o único sucessor possível que é o estado *"try"*. Nesse se dá a tentativa do envio da mensagem. Se ela for perdida (probabilidade de 10%), o estado *"lost"* é alcançado, e tem como única opção de caminho (probabilidade de 100%) a volta ao estado *"try"*, no qual ocorrerá uma nova tentativa de envio da mensagem. Caso a mensagem seja enviada com sucesso, o estado *"delivered"* é alcançado, e, em seguida, o protocolo retorna ao seu estado inicial.

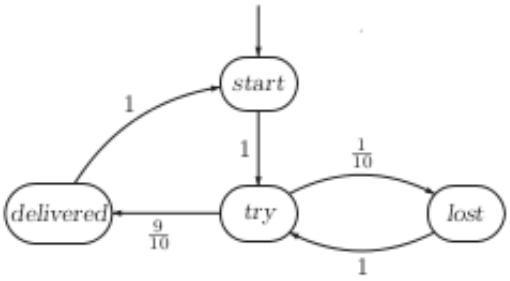


Figura 2.2: Exemplo de um protocolo de comunicação modelado utilizando Cadeia de Markov Discreta (DTMC) [23]

Para uma análise de dependabilidade usando Cadeias de Markov o sistema pode ser modelado associando a cada transição a probabilidade de execução correta de cada componente envolvido. Essa probabilidade seria a confiabilidade do componente. A confiabilidade global do sistema seria, então, calculada pela probabilidade do sistema alcançar o estado final de sucesso, situação na qual, todos os componentes foram executados com sucesso. Essa forma de modelagem foi utilizada em [11] para a análise de dependabilidade realizada.

2.3.2 *PRISM Model Checker*

Para fazer as verificações exigidas na análise de dependabilidade que utiliza *model checking*, será utilizada a ferramenta *PRISM*.

A ferramenta PRISM realiza verificação de modelos probabilísticos e foi desenvolvida na Universidade de Birmingham (Inglaterra). Ela provê suporte a modelos em DTMC, CTMC e MDP (*Markov Decision Process*). Publicações diversas, bem como estudos de casos, sobre a PRISM podem ser encontradas em [26].

A PRISM possui três ambientes para operação:

1. para a modelagem do sistema seguindo uma linguagem própria da ferramenta;
2. para a realização de uma simulação do modelo;
3. para a especificação de propriedades de dependabilidade usando a linguagem PCTL (*Probabilistic Computational Tree Logic*).

No ambiente de modelagem, o sistema pode ser representado diretamente pelo seu modelo, contendo suas transições de estado, ou pelos modelos dos seus componentes, que quando integrados de forma paralela, representarão o modelo do sistema.

Na Listagem 2.1 é apresentado um trecho da modelagem de um sistema na PRISM. A linha 1 do código indica o modelo probabilístico a ser utilizado. No caso é o DTMC. Nas linhas 3 a 5 estão definidas constantes que representam as probabilidades para a tomada das decisões sobre os caminhos a serem seguidos na execução do modelo; no caso de uma análise de confiabilidade, elas representariam as confiabilidades dos componentes. Na linha 7 ocorre a definição do módulo que está sendo modelado. No caso, está sendo modelado diretamente o sistema e não os seus componentes. Nessa linha está expressa a quantidade de estados que o sistema (representado pela variável *sys_st*) pode atingir (0 a 15) e ainda o estado inicial.

```
1  dtmc
2
3  const double R_SS;
4  const double R_CT;
5  const double R_DB;
6
7  module SistemaExemplo sys_st : [0..15] init 0;
8  [On] sys_st = 0 -> R_SS:(sys_st'=1) + (1-R_SS):(sys_st'=15);
9
10 [Pressure] sys_st = 1 -> R_DB:(sys_st'=2) + (1-R_DB):(sys_st'=15);
11 [Off] sys_st = 2 -> R_SS:(sys_st'=9) + (1-R_SS):(sys_st'=15);
12 [Data] sys_st = 3 -> R_CT:(sys_st'=4) + (1-R_CT):(sys_st'=15);
13 ...
14 [Pressure] sys_st = 14 -> R_DB:(sys_st'=5) + (1-R_DB):(sys_st'=15);
15
16 endmodule
```

Listagem 2.1: Trecho de uma modelagem utilizando a linguagem da PRISM.

As linhas seguintes do modelo são preenchidas com a representação das transições entre os diversos estados que o sistema pode alcançar. O formato de cada representação é o seguinte:

$$[ação]<estadoAtual> \rightarrow <probabilidade> : <estadoFuturo>$$

A tag $[ação]$ indica a mensagem que será executada e utilizada para a sincronização dos módulos. Já tag $<estadoAtual>$ indica um estado o qual serve como condição para a realização da transição. Ao ser realizada a transição, ela obedecerá o valor contido na tag $<probabilidade>$ para alcançar um estado futuro indicado na tag $<estadoFuturo>$.

No caso apresentado na Listagem 2.1, tomando a linha 11 como exemplo, tem-se que se o sistema estiver no estado 2, ele pode alcançar o estado 9, com a probabilidade R_SS, ou o estado 15, com a probabilidade $(1 - R\_SS)$. Importante destacar que a mensagem

[*Off*] será usada para a sincronização com os módulos dos outros componentes seguindo as regras da linguagem CSP (*Communicating Sequential Process*) [9]. No caso apresentado, está expresso o modelo direto do sistema, portanto não foi necessária operações de sincronização entre os módulos.

No ambiente para a simulação, é possível a execução do modelo e a identificação de eventuais *dead-locks*.

Já no ambiente de especificação de propriedades para o modelo, a PRISM possibilita a descrição de propriedades usando lógicas temporais como PCTL (*Probabilistic Computation Tree Logic*), CSL (*Continuous-Time Stochastic Logic*), LTL (*Linear Temporal Logic*), entre outras, e a sua posterior verificação. Pode ser exigido o valor de determinada propriedade ou se o modelo a satisfaz.

As modelagens neste trabalho serão verificadas por propriedades especificadas em PCTL.

2.3.3 *Probabilistic Computational Tree Logic*(PCTL)

Probabilistic Computational Tree Logic (PCTL) é uma linguagem, definida originalmente em [24], utilizada para modelar propriedades em modelos probabilísticos. Ela foi estendida da linguagem *Computational Tree Logic* (CTL) [27] que não contemplava o emprego em modelos probabilísticos.

Uma fórmula escrita em PCTL indica uma condição para o sistema atingir um dos estados. A interpretação é se o sistema violou ou não a propriedade expressa em PCTL.

Os detalhes para a construção de propriedades PCTL podem ser encontrados em [24]. Para exemplificar a sintaxe dessa linguagem, seguem algumas propriedades e respectivos significados:

- `P>=1 [F "terminate"]`: o algoritmo eventualmente terminará no estado de sucesso com probabilidade igual a 1;
- `P<0.1 [ F<=100 num_errors > 5]`: a probabilidade que pelo menos 5 erros ocorram nas primeiras 100 unidades de tempo é menor que 0.1;
- `S=? [ queue_size / max_size > 0.75 ]`: qual a probabilidade da fila ultrapassar 0.75 da sua capacidade.

Neste trabalho serão utilizadas fórmulas em PCTL que especificam a propriedade de alcançabilidade (tradução para *reachability*) dos modelos. Esse tipo de propriedade permite verificar a probabilidade do modelo alcançar, a partir de um determinado estado, um outro estado em número limitado ou ilimitado de passos.

2.4 Caracterização de defeitos

De acordo com o conceito apresentado na Seção 2.1, um defeito ocorre quando o serviço entregue pelo sistema não corresponde àquele que era desejado.

Essa definição permite estabelecer que um defeito afeta a confiabilidade de um sistema. Dessa forma, em uma análise de dependabilidade com foco na confiabilidade, os defeitos do sistema devem ser avaliados, como forma de compreender, dentre outras coisas, como a confiabilidade está sendo afetada e quais ações a serem tomadas.

Em [18] é apresentada uma relação de quatro critérios para a caracterização de defeitos. São eles:

1. Quanto ao domínio:

   (a) **Conteúdo**: são aqueles defeitos em que o conteúdo da informação entregue na interface do serviço é diferente do esperada, podendo ocorrer em conjuntos numéricos ou não numéricos;

   (b) **Tempo**: são aqueles defeitos em que o conteúdo da informação, apesar de correto, é entregue fora do tempo esperado, podendo ser antecipado ou atrasado;

1.1. Quando os defeitos ocorrem tanto com relação ao conteúdo como ao tempo, ainda cabe a classificação:

   (a) **Defeitos de interrupção**: quando o serviço é interrompido;

   (b) **Defeitos erráticos**: quando o serviço é entregue, mas erradamente, tanto no tempo como no conteúdo.

2. Quando à detectabilidade:

   (a) **Detectáveis**: quando existem mecanismos que detectam os defeitos;

   (b) **Indetectáveis**: quando não existem esses mecanismos e os defeitos ocorrem de forma silenciosa;

3. Quanto à consistência:

   (a) **Consistentes**: o serviço incorreto é percebido de modo idêntico a todos os usuários;

   (b) **Inconsistentes**: o serviço incorreto é percebido de modos diferentes pelos usuários, podendo haver usuários que estão recebendo o serviço correto. Os erros ocorrem de forma arbitrária;

4. Quanto às consequências:

(a) **Defeitos irrelevantes**: quando o custo da ocorrência do defeito é irrelevante;

(b) **Defeitos catastróficos**: quando o custo da ococrrência do defeito é diversas vezes maior que o benefício provido pela entrega correta do serviço.

Capítulo 3

Cenários Implícitos

Neste capítulo serão apresentados os principais pontos teóricos a respeito de cenários implícitos. Para ilustrar os principais conceitos deste capítulo, será utilizado um sistema fictício apresentado no trabalho de Alur *et al.* [7]. Trata-se da descrição de um sistema que regula a quantidade de combustível utilizada por uma usina nuclear. No caso, dois componentes (P1 e P2) gerenciam remotamente as quantidades de Urânio (UR) e Ácido Nítrico (AN) a serem consumidos. À medida que for sendo necessário, mais informações sobre esse sistema serão apresentadas.

3.1 Modelagem por cenários

Cenários descrevem como componentes, ambiente e usuários interagem de modo a prover funcionalidades de um sistema [28]. Em uma modelagem por cenários, os diferentes cenários são combinados para expressar o comportamento global do sistema.

Os cenários podem ser descritos de várias formas, de textos narrativos até diagramas UML[1] [2].

Neste trabalho, seguindo o exemplo de outros vários, como [8, 29, 30], será utilizada a notação MSC (*Message Sequence Charts*) [3] para realizar a especificação em cenários.

Nesse caso, cada cenário é descrito em um bMSC (*basic Message Sequence Chart*), ilustrado na Figura 3.1, que consiste em um conjunto de componentes (no caso da Figura 3.1, Processo1, Processo2 e Processo3) que trocam mensagens(no caso da Figura 3.1, msgA, msgB e msgC) de forma ordenada, entre si.

Formalmente, um bMSC é definido como uma estrutura $(E, L, I, \lambda, <, tgt)$ em que:

- E é um conjunto de eventos particionados em um conjunto S de eventos enviados e um conjunto R de eventos recebidos;

[1] Especificamente, diagramas de sequência.

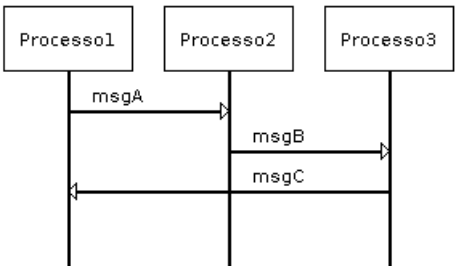


Figura 3.1: Exemplo básico de bMSC

- L é um conjunto finito de rótulos;
- I é um conjunto finito de componentes;
- $\lambda : E \rightarrow L \times I$ é uma função que mapeia eventos para seus rótulos e componentes, definindo-se $i(E)$ como o conjunto de eventos e de modo que $\lambda(e) = (I, i)$;
- $<$ é o conjunto das ordens $<_i \subseteq (i(E) \times i(E))$ em que $i \in I$;
- $tgt : S \rightarrow R$ é uma função que mapeia os eventos enviados nos eventos recebidos.

Ainda sobre a MSC, para integrar os cenários e modelar o comportamento global do sistema, são utilizados os diagramas hMSC (*high level Message Sequence Chart*), que são representados por um grafo, como pode ser verificado na Figura 3.2, no qual os nós são os diagramas bMSC.

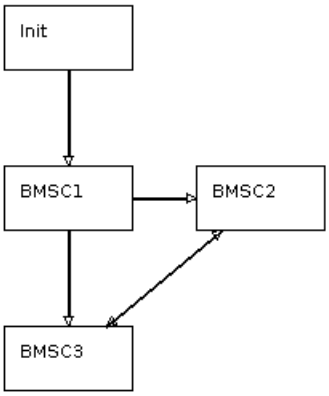


Figura 3.2: Exemplo básico de um hMSC

É importante destacar que, além de bMSCs, os nós podem conter outros hMSCs, possibilitando uma relação de hierarquia entre hMSCs.

As arestas do hMSC indicam as ordens aceitáveis para a execução dos cenários, permitindo o reúso de bMSCs e a especificação de bifurcações e laços de repetição (*loops*).

Um hMSC é definido formalmente por um grafo na forma (N, A, s_0) onde:

- N é um conjunto de nós;
- $A \subseteq (N \times N)$ é um conjunto de arestas;
- $s_0 \in N$ é o nó inicial

A sintaxe MSC é definida com detalhes em [3].

Considerando o já citado sistema que gerencia a quantidade de combustível em uma usina nuclear, a Figura 3.3 apresenta os dois cenários (M1 e M2) modelados para o sistema. De acordo com a especificação desse sistema, o componente P1 realiza a ação de incrementar as quantidades dos combustíveis Urânio e Ácido Nítrico em uma unidade. Essa ação é modelada pelas mensagens "*incUranio*" ou "*incAcidoNitrico*". Por sua vez, o componente P2 realiza a ação de dobrar as quantidades existentes dos combustíveis. Essa ação é modelada pelas mensagens "*doubleUranio*" ou "*doubleAcidoNitrico*".

Dado que os componentes P1 e P2 atuam remotamente, os cenários apresentados relatam situações onde os componentes atuam em ordens diferentes. No cenário M1, inicialmente, P1 envia as mensagens, incrementando os valores de ambos ingredientes, e em seguida, P2 dobra as quantidades. No cenário M2 ocorre o contrário. Primeiro P2 duplica as quantidades e depois P1 incrementa. Ao fim das duas transações, um requisito crucial do sistema é que as quantidades de Urânio e Ácido Nítrico sejam iguais, sob a pena de ocorrer um acidente de graves proporções.

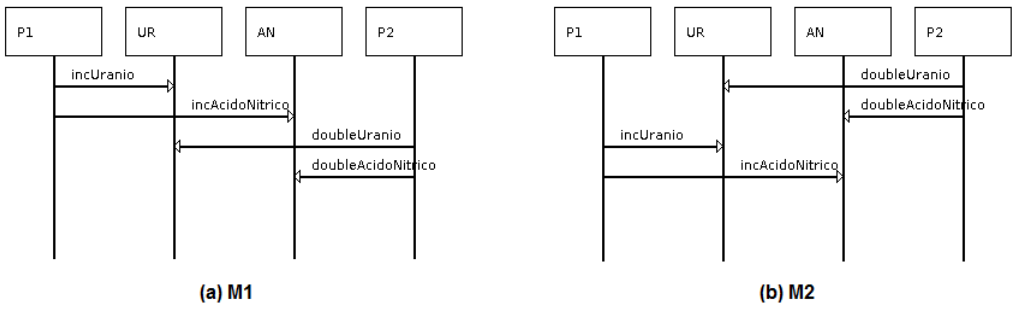


Figura 3.3: Exemplo de cenários

A Figura 3.4 apresenta o diagrama hMSC esboçando a possibilidade da ocorrência paralela entre dos dois cenários (M1 e M2), simulando o comportamento concorrente dos componentes P1 e P2.

Cabe destacar foi utilizada a ferramenta LTSA-MSC [31] para a confecção dessa modelagem.

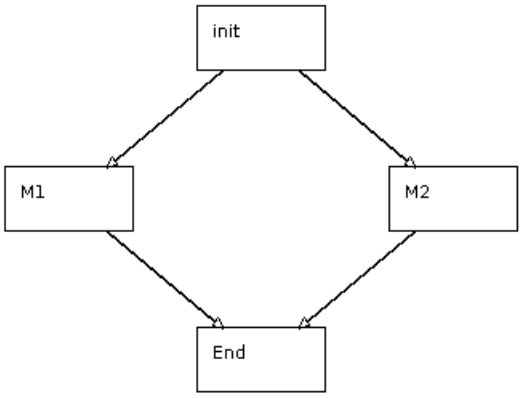


Figura 3.4: Diagrama hMSC para a especificação do sistema de gerência de combustível da usina nuclear

3.2 Modelo comportamental

Com a especificação dos cenários concluída, as análises a serem realizadas exigem a integração dos cenários para a obtenção do modelo comportamental do sistema, também chamado de modelo arquitetural.

Com os modelos comportamentais, técnicas e ferramentas de análise e verificação de propriedades podem ser utilizadas, visto que eles apresentam todos os estados possíveis do sistema. Essas práticas possibilitam a descoberta de eventuais falhas do projeto [32].

No trabalho [1] de Uchitel *et al.* é apresentado um algoritmo que integra os diversos cenários para a obtenção de um modelo que descreva os comportamentos desejados do sistema. Esse modelo é expresso na notação FSP (*Finite State Process*) [33], que é uma expressão textual para os diagramas LTS(*Labelled Transition System*) [42], que representam graficamente as transições entre os estados do sistema.

Cabe destacar que outros processos para obter o modelo comportamental a partir de modelagem em cenários já foram propostos em trabalhos como [47–49].

3.2.1 *Labelled Transition System(LTS)*

Proposto em [42], os diagramas LTS (*Labbeled Transition System*) evidenciam as transições entre os diversos estados de um sistema. Essas transições são rotuladas com as mensagens trocadas entre os componentes do sistema, representando os eventos que levam às mudanças de estado.

Formalmente, o diagrama LTS é definido como uma estrutura (S, L, Δ, q) em que:

- S é um conjunto de estados;
- $L = \alpha(P) \cup \tau$ é um conjunto de rótulos onde $\alpha(P)$ é o alfabeto de P e τ refere-se às ações internas que não são observáveis pelo ambiente de um LTS.

- $\Delta \subseteq (S \setminus \pi, \epsilon \times L \times S)$ define as transições, e respectivos rótulos, entre os estados. Nenhuma transição pode iniciar de estado de erro, π, ou final, ϵ.
- $q \in S$ indica o estado inicial.

Diagramas LTS são utilizados para a representação de sistemas concorrentes, possibilitando a modelagem individual de cada componente e a posterior integração utilizando a composição paralela similar à definida na reconhecida linguagem formal CSP (*Communicating sequential processes*) [9], na qual as mensagens dos componentes são intercaladas (*interleaving*), considerando a sincronização entre aquelas de mesmo rótulo.

Sejam P_1 e P_2 dois diagramas LTS, em que $P_i = (S_i, L_i, \Delta_i, q_i)$, a composição paralela desses dois componentes, descrita por $P_1 \parallel P_2$, é formalmente definida como:

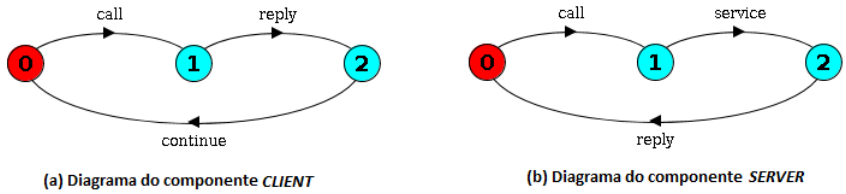
- Um LTS $(S_1 \times S_2, L_1 \times L_2, \Delta, q_1 \times q_2)$, onde Δ é a menor relação que satizfaz às seguintes condições:

$$\frac{P_1 \xrightarrow{l} P_1'}{P_1 \parallel P_2 \xrightarrow{l} P_1' \parallel P_2} \qquad (l \notin \alpha(P_2))$$

$$\frac{P_2 \xrightarrow{l} P_2'}{P_1 \parallel P_2 \xrightarrow{l} P_1 \parallel P_2'} \qquad (l \notin \alpha(P_1))$$

$$\frac{P_1 \xrightarrow{l} P_1' \quad P_2 \xrightarrow{l} P_2'}{P_1 \parallel P_2 \xrightarrow{l} P_1' \parallel P_2'} \qquad (l \neq \tau)$$

Para ilustrar o processo de composição paralela em diagramas LTS, serão utilizados os modelos de dois componentes (*Client* e *Server*), expressos na Figura 3.5. A composição paralela desses dois modelos leva em conta as mensagens comuns "*call*"e "*reply*", que deverão ser executadas de forma sincronizada. Enquanto as outras duas ("*service*"e "*continue*") poderão ser executadas independentemente.



(a) Diagrama do componente *CLIENT* (b) Diagrama do componente *SERVER*

Figura 3.5: Diagramas LTS utilizados para a ilustração do processo de composição paralela

O LTS a ser obtido contará com o seguinte conjunto de estados, obtidos a partir da combinação entre os estados dos dois componentes considerados: (0,0), (0,1), (0,2), (1,0),

(1,1), (1,2), (2,0), (2,1), (2,2). Partindo do estado (0,0), ambos componentes executam a mensagem "*call*", o que os leva ao estado 1 de cada modelo. Portanto, o estado (1,1) seria alcançado. A partir do estado (1,1), o componente Server pode executar a mensagem "*service*", que levaria o modelo ao estado (1,2). Entretanto, o componente Client não poderia executar a mensagem "*reply*", que levaria o modelo ao estado (2,1), devido ao fato dela ser uma das mensagens que estão sincronizadas e só poderá ser executada quando o componente Server estiver no estado 2, ou seja, a partir do estado (1,2). No estado (2,2) o componente Client pode executar a mensagem "*continue*", que leva o modelo ao estado (0,2), já o componente Server não poderia executar a mensagem "*reply*", pois como já visto, o componente Client teria que estar no estado 1, pois é desse estado que a mensagem é executada no seu LTS. A Figura 3.6 mostra o diagrama LTS obtido da composição paralela dos dois componentes apresentados.

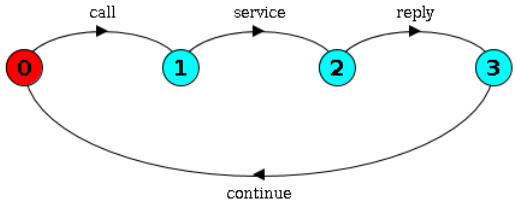


Figura 3.6: Diagrama LTS resultante da composição paralela entre os componentes *Client* e *Server*

A Figura 3.7 mostra o modelo comportamental do sistema utilizado como exemplo nesse capítulo usando um diagrama LTS. Na Seção 3.2.4 será abordado o processo de obtenção desse diagrama.

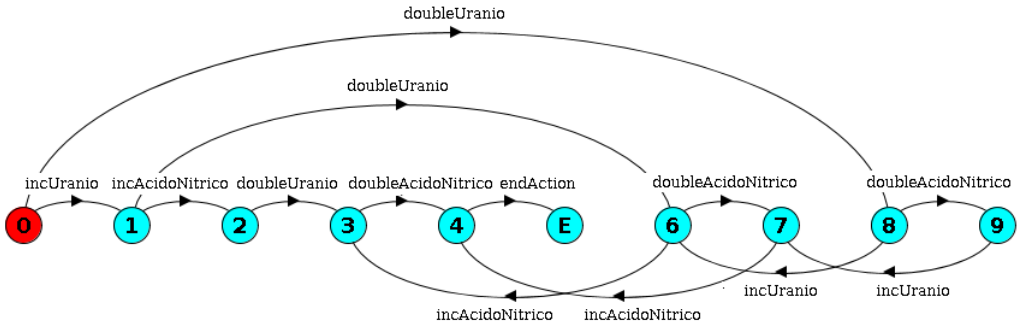


Figura 3.7: Diagrama LTS representando o modelo comportamental do sistema exemplo

3.2.2 Notação *Finite State Process(FSP)*

Pode ser percebido que representação de sistemas em diagramas LTS possuem a restrição de não serem adequados quando o sistema apresenta muitos estados. Nesses casos, a

análise do diagrama se torna impraticável, dada a poluição visual causada pelos inúmeros estados e transições. Como forma de atacar esse problema, Magee *et al.* porpuseram em [33] uma notação algébrica chamada *Finite State Process (FSP)* para especificar de forma textual os diagramas LTS.

A FSP é uma notação de especificação com uma semântica bem definida, baseada na do LTS. Cada expressão em FSP pode ser mapeada para um diagrama LTS, e vice e versa, aproveitando inclusive as propriedades.

Cada sentença FSP apresenta a indicação dos estados consecutivos e as mensagens que orientam essas transições. O operador "|"indica a possibilidade de escolha entre dois caminhos, enquanto o operador "||"representa a composição paralela, segundo o especificado na CSP. Já o operador "`END`"indica que o sistema alcançou o estado final de sucesso.

A Listagem 3.1 apresenta em FSP o modelo comportamental da Figura 3.7.

```
1  ArchitectureModel = Q0,
2   Q0 = (incUranio -> Q1 | doubleUranio -> Q8),
3   Q1 = (incAcidoNitrico -> Q2 | doubleUranio -> Q6),
4   Q2 = (doubleUranio -> Q3),
5   Q3 = (doubleAcidoNitrico -> Q4),
6   Q4 = (endAction -> Q5),
7   Q5 = END,
8   Q6 = (incAcidoNitrico -> Q3 | doubleAcidoNitrico -> Q7),
9   Q7 = (incAcidoNitrico -> Q4),
10  Q8 = (incUranio -> Q6 | doubleAcidoNitrico -> Q9),
11  Q9 = (incUranio -> Q7).
```

Listagem 3.1: Modelo comportamental do sistema de controle de combustível da usina expresso em FSP

3.2.3 *Labelled Transition Systems Analyser (LTSA)*

Em [39], Magee *et al.* apresentaram a ferramenta de verificação de modelos *Labelled Transition System Analyser(LTSA)* que explora modelagens de sistemas concorrentes para checar propriedades especificadas.

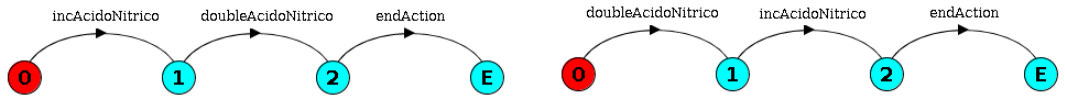
A LTSA possibilita a geração de diagramas LTS a partir dos respectivos modelos expressos em FSP. Desenvolvida em Java, a LTSA possui uma arquitetura que permite a adição de *plugins* para o acréscimo de mais funcionalidades [34]. Em [31] foi apresentado um *plugin* que disponibiliza uma interface gráfica para a modelagem do sistema em MSC e a identificação de eventuais cenários implícitos.

3.2.4 Obtenção do modelo comportamental segundo Uchitel *et al.* [1]

No algoritmo proposto em [1] para a integração dos cenários especificados e obtenção do modelo comportamental, cada componente da especificação é modelado em um LTS, para em seguida, serem integrados usando a operação de composição paralela.

Na geração dos diagramas LTS de cada componente, as mensagens trocadas pelo componente dentro dos cenários se tornam as transições entre os estados. Nesse sentido, um diagrama LTS por componente é composto para cada cenário. Esses diagramas, referentes a cada cenário, são então integrados seguindo as regras de composição estipuladas no hMSC. Ou seja, caso a ordem definida entre os bMSCs seja sequencial, essa será a forma de integração dos LTS. Caso exista algum paralelismo entre os bMSCs, a integração será a de composição paralela.

No caso do sistema utilizado como exemplo nesse capítulo, os cenários M1 e M2 estão dispostos de forma paralela, logo a composição será feita de forma paralela. Para ilustrar esse processo de obtenção do LTS de cada componente, a Figura 3.8 mostra os dois diagramas LTS do componente AN referentes, respectivamente, aos cenários M1 e M2. Eles serão compostos de forma paralela para a formação do diagrama LTS mostrado na Figura 3.9(c).



(a) Diagrama LTS referente ao cenário M1

(b) Diagrama LTS referente ao cenário M2

Figura 3.8: Diagrama LTS representando o modelo comportamental do sistema exemplo

A Figura 3.9 mostra os diagramas LTS dos 4 componentes do sistema, que deverão ser compostos de forma paralela para a obtenção do modelo comportamental já ilustrado na Figura 3.7.

3.3 Cenários implícitos

Combinações inesperadas no modo como os componentes interagem podem levar ao surgimento de comportamentos que não estavam previstos. Tais comportamentos, estudados inicialmente por Alur *et al.* [7] são denominados "cenários implícitos" e surgem principalmente porque os componentes foram modelados considerando uma visão local do que está acontecendo no sistema, e não uma visão do comportamento global do sistema. Dessa forma, quando os componentes são sincronizados de forma paralela, representando

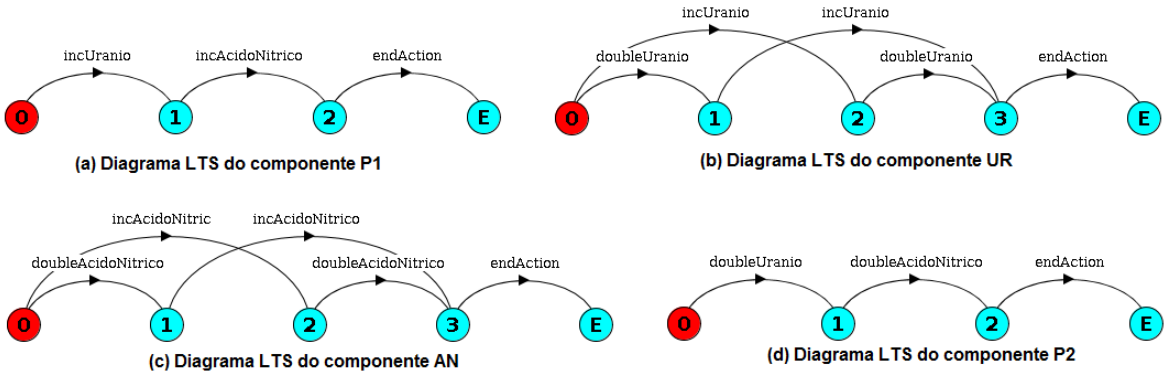


Figura 3.9: Diagrama LTS representando o modelo comportamental do sistema exemplo

a execução concorrente, as mensagens são intercaladas (fenômeno denominado *interleaving*) gerando novas sequências, que não estavam presentes em nenhum dos outros cenários previamente concebidos.

Voltando ao sistema que gerencia o combustível em uma usina, verifica-se que a composição realizada para a geração do modelo comportamental do sistema, implicou no aparecimento de um novo cenário, mostrado na Figura 3.10. No caso, as mensagens enviadas por P1 e P2 surgem intercaladas, de uma forma que não estava prevista inicialmente, ou seja, um comportamento inusitado, que surgiu da composição entre os dois cenários inicialmente previstos. O resultado dessa execução seria um desequilíbrio entre as quantidades dos combustíveis, ferindo um requisito crucial, o que poderia resultar em um acidente.

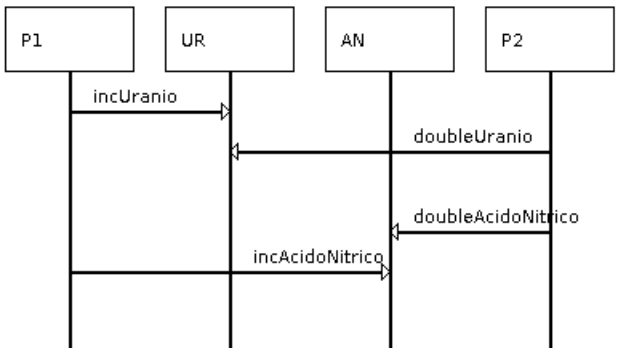


Figura 3.10: Exemplo de cenário implícito presente na especificação

Importante destacar que, isoladamente, os dois cenários executam corretamente, seguindo a ordem adequada. Entretanto, na composição paralela, as mensagens são intercaladas (ocorrência de "*interleaving*"). No caso apresentado, a nova sequência descoberta implicou em uma situação problemática para o sistema, visto que o resultado seria um acidente. Em [36], foi proposto o conceito de *cenários implícitos negativos*, que seriam aqueles cenários indesejados pelos usuários. Portanto, o cenário obtido é um cenário im-

plícito negativo, o que indica ao projetista a necessidade de um revisão do modelo. Por outro lado, se o cenário implícito não trouxer problemas, se trata de um *cenário implícito positivo*, e devem ser incluídos na modelagem do sistema, de modo que passem a estar previstos.

3.4 Detecção dos cenários implícitos

Os cenários implícitos devem ser identificados sob a pena de poderem resultar em erros a serem descobertos em etapas posteriores do ciclo de desenvolvimento ou até mesmo por usuários finais.

Conforme comentado na Seção 3.2.1, um processo automatizado para a detecção de cenários implícitos foi proposto por Uchitel *et al.* em [8]. Na ocasião, foi desenvolvida a ferramenta LTSA-MSC, um *plugin* para a LTSA. Nesse processo de detecção, os cenários implícitos são identificados um a um, caracterizando um processo iterativo, o qual cada cenário descoberto é submetido a uma análise para verificação quanto ao seu acolhimento no modelo, no caso de se tratar de um cenário implícito positivo, ou descarte, no caso dos cenários implícitos negativos. Mais informações sobre o tratamento dado aos cenários implícitos negativos encontrados são apresentadas na Seção 3.5.

Então, com a LTSA-MSC parte-se da modelagem em cenários, expressa em MSC, obtêm-se o modelo comportamental do sistema expresso em FSP ou no diagrama LTS, e identificam-se os eventuais cenários implícitos presentes na modelagem. Ao final, busca-se obter um modelo que não apresente cenários implícitos.

A Figura 3.11 mostra a interface da LTSA-MSC no momento que um dos cenários implícitos do sistema de gerência de combustíveis foi identificado.

A ferramenta LTSA-MSC será utilizada neste trabalho para a identificação dos cenários implícitos a partir de uma especificação em cenários, conforme será visto no Capítulo 4.

Na Figura 3.12 são apresentados os dois cenários implícitos negativos detectados pela LTSA-MSC.

Os detalhes sobre o processo de detecção de cenários implícitos podem ser encontrados em [8].

3.5 Obtenção das *Constraints*

No processo descrito por Uchitel, os cenários implícitos positivos são agregados ao modelo e passam a fazer parte da modelagem, deixando de ser implícitos, passando a serem explícitos. Por outro lado, os negativos devem ser evitados por prejudicarem o funcionamento desejado do sistema.

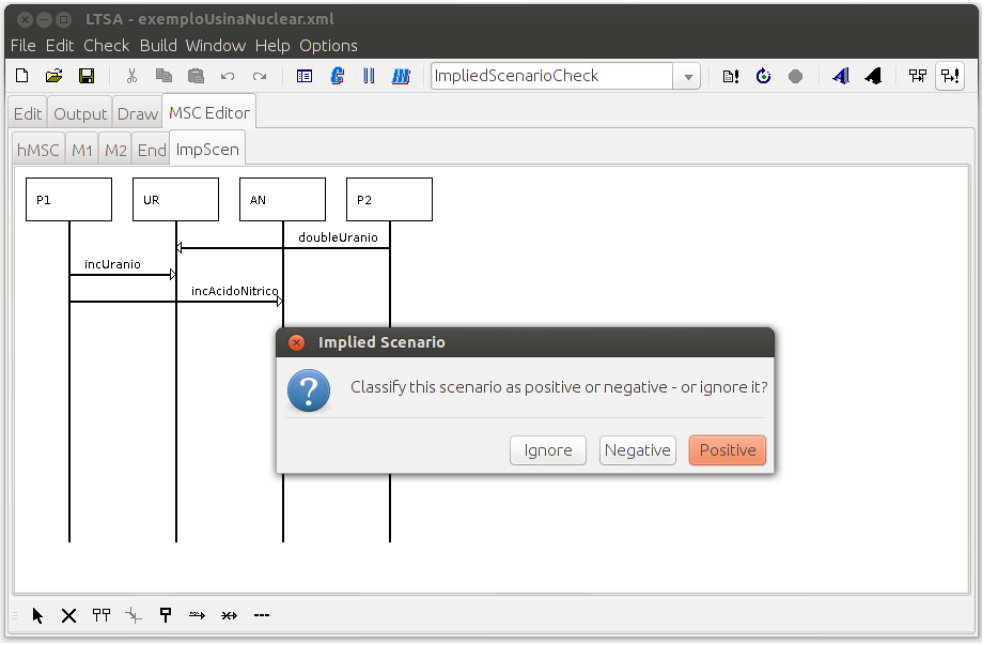


Figura 3.11: Interface da ferramenta LTSA-MSC no momento da detecção de um cenário implícito

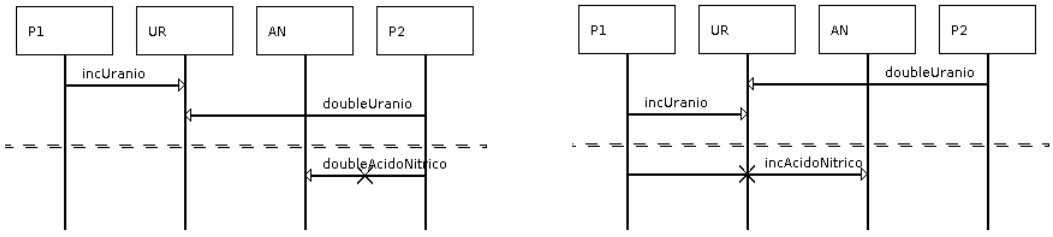


Figura 3.12: Cenários implícitos negativos detectados para o sistema de gerência de combustível na usina

A cada descoberta de um cenário implícito negativo, o processo prevê que este não seja mais executado, dando a chance de outros serem descobertos. A forma utilizada para evitar a execução do cenário implícito sem alterar o modelo é a utilização do recurso de sincronização paralela. Isto é, a partir do cenário implícito negativo detectado, é criado um diagrama LTS para ser composto junto ao modelo original. Esse diagrama é concebido de forma a impedir que o LTS execute determinada sequência que caracteriza o cenário, devido ao sincronismo das mensagens semelhantes.

A Figura 3.13 apresenta a estrutura do diagrama LTS que impede a ocorrência do primeiro cenário implícito negativo detectado para o sistema em questão. Isto é, a sequência: $incUranio \rightarrow doubleUranio \rightarrow doubleAcidoNitrico$.

Esses diagramas criados para a sincronização são chamados por Uchitel de *Constraints*.

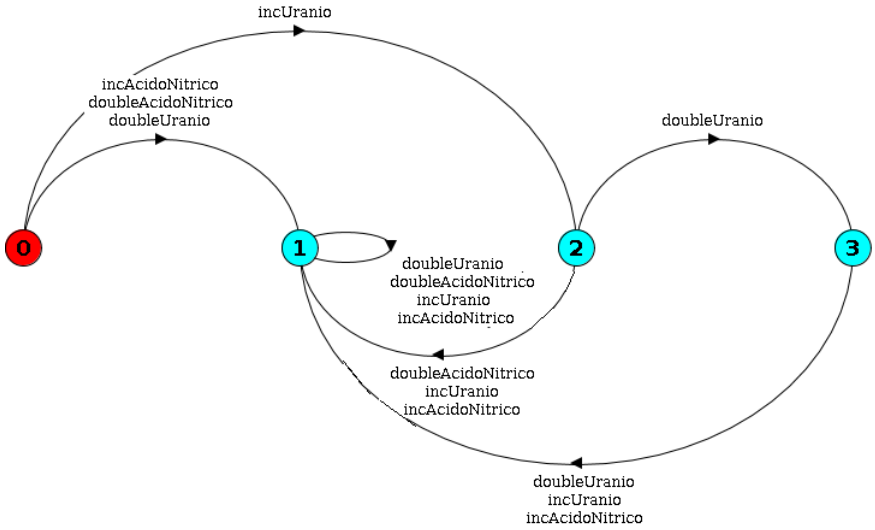


Figura 3.13: *Constraint* concebida para impedir a ocorrência do primeiro cenário implícito negativo para o sistema de gerência de combustível.

Caso todos os cenários implícitos sejam detectados, e consequentemente, todas as *Constraints* concebidas, deixando o modelo comportamental do sistema livre da ocorrência de cenários implícitos, o diagrama resultante é chamado de *Constrained Model*. Na Figura 3.14 é apresentado o *Constrained Model* referente ao sistema de gerência de combustível utilizado nesse capítulo. Verifica-se que as sequências: "*incUranio* → *doubleUranio* → *doubleAcidoNitrico*" e "*doubleUranio* → *incUranio* → *incAcidoNitrico*" não estão presentes.

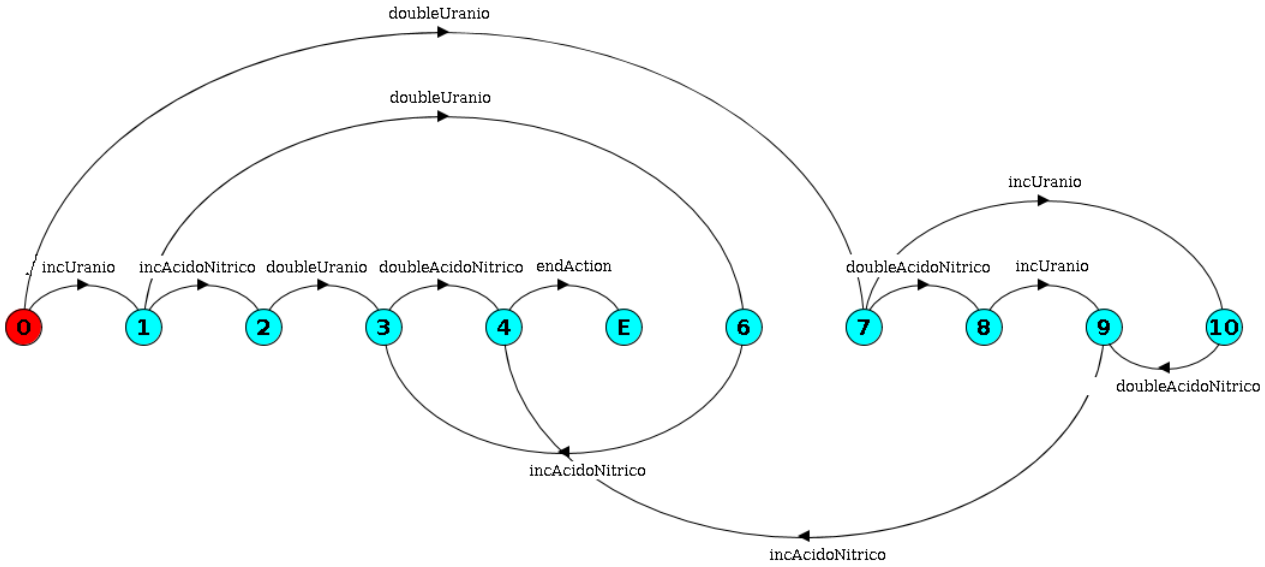


Figura 3.14: *Constrained Model* concebido para o sistema de gerência de combustível.

Capítulo 4

Metodologia

Neste capítulo será apresentada a principal contribuição desse trabalho, que é a metodologia proposta para a análise do impacto de cenários implícitos sobre a confiabilidade de um sistema. Essa análise busca caracterizar a forma como os cenários implícitos afetam o modelo tanto qualitativamente como quantitativamente [44].

4.1 Metodologia

A Figura 4.1 apresenta o fluxo completo da metodologia proposta nesse trabalho. Cabe destacar que as etapas iniciais do processo ("Modelagem em MSC", "Síntese do modelo comportamental em FSP" e "Detecção dos cenários implícitos negativos") são aplicações de trabalhos publicados por Uchitel *et al.* [1, 8, 35, 36] e já abordadas nas Seções 3.1, 3.2, 3.4, respectivamente. Após a etapa de identificação dos cenários implícitos, sucedem-se as etapas de análise qualitatva e quantitativa.

Com relação à análise qualitativa, são propostos dois passos. O primeiro seria uma análise do domínio da aplicação para a correlação de possíveis defeitos no sistema com os cenários implícitos considerados. Com a lista dos defeitos associados, eles são caracterizados conforme os critérios propostos em [18], já abordados na Seção 2.4.

Na análise quantitativa, o primeiro passo é a tradução do modelo comportamental expresso em FSP, obtido via utilização da ferramenta LTSA-MSC, para o modelo probabilístico expresso na linguagem da ferramenta PRISM, já apresentada na Seção 2.3.2. Esse modelo probabilístico receberá como parâmetros as confiabilidades individuais dos componentes. Em seguida, o modelo comportamental é analisado para a identificação dos pontos onde os cenários implícitos negativos considerados ocorrem. Essa identificação será utilizada para as análises posteriores nas quais os pontos do modelo que levam à ocorrência desses cenários serão alterados para caracterizar a ocorrência de erro ao serem executados. As etapas seguintes são as análises propriamente ditas que poderão revelar in-

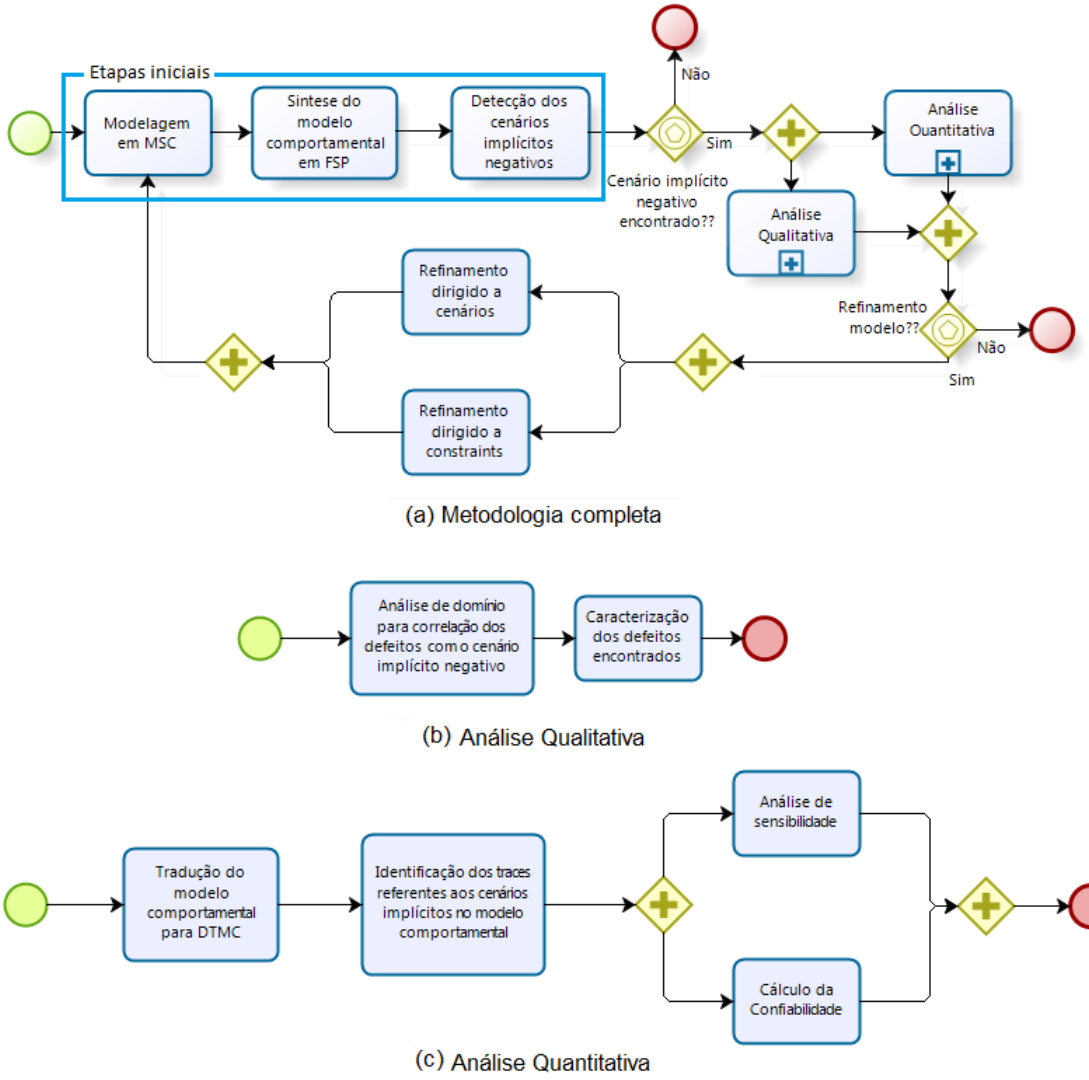


Figura 4.1: Metodologia proposta

formações quantitativas de como os cenários implícitos afetam a confiabilidade. A análise de sensibilidade é importante quando não estão disponíveis as informações sobre as confiabilidades individuais dos componentes. Nesse caso, busca-se identificar como a confiabilidade de cada componente afeta a confiabilidade global do sistema. Com isso, poderá ser identificado quais componentes precisariam de maior atenção quanto aos mecanismos para elevar a sua confiabilidade individual. Na etapa do cálculo da confiabilidade, são utilizados os valores disponíveis para as confiabilidades dos componentes. Nessas duas etapas, é utilizada uma propriedade PCTL, apresentada na Seção 2.3.3, para o cálculo da confiabilidade do modelo.

Com a realização das análises qualitativa e quantitativa, espera-se que as informações estejam disponíveis para as eventuais ações de refinamento arquitetural focadas em au-

mentar a confiabilidade do modelo. São propostas duas alternativas: o refatoramento do modelo, com a alteração da comunicação entre os componentes; ou a manutenção do modelo seguida da implementação de medidas para evitar os efeitos advindos dos cenários implícitos. Para a representação dessas duas medidas e possibilidade de comparação dos resultados obtidos, serão repetidos os passos da análise quantitativa para cada uma das opções, sendo que no caso da implementação de mecanismos para evitar a ocorrência dos cenários, será utilizado o *Constrained Model*, apresentado na Seção 3.5.

Nas seções seguintes, a metodologia será detalhada, etapa por etapa.

4.2 Modelagem do sistema em MSC

- ***Entrada:*** Especificação do sistema.
- ***Saída:*** Modelagem em cenários, utilizando a linguagem MSC.

Conforme adiantado na Seção anterior, nessa etapa será aplicada a modelagem em cenários segundo a abordagem proposta por Uchitel *et al.* [1]. Para orientar a apresentação da metodologia será utilizado um sistema fictício, inicialmente proposto em [8], e aproveitado em outros trabalhos, como [10] e [15], para demonstrar a ocorrência de cenários implícitos. Trata-se de um sistema que gerencia a pressão do vapor de uma caldeira (livre tradução para *boiler*). Seu objetivo é manter a pressão em determinado patamar. O sistema é composto por um sensor que verifica a pressão do vapor e envia as informações para um módulo de controle que aciona um mecanismo para ativar ou desativar o sistema de aquecimento de água, responsável pela elevação ou não da pressão do vapor produzido.

A Figura 4.2 ilustra o referido sistema.

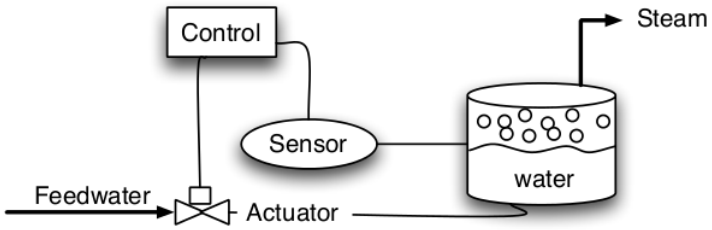


Figura 4.2: Ilustração do sistema de gerência da caldeira, utilizado para apresentar a metodologia [8]

Para o sistema em questão, serão considerados 4 componentes para a modelagem. São eles: Controlador(*Control*), Sensor, Atuador(*Actuator*) e Base de Dados(*Database*).

Os componentes estarão interagindo para a implementação das funcionalidades de inicialização(*Initialize*), registro(*Register*), análise(*Analysis*), finalização(*Terminate*) e desligamento(*End*).

As Figuras 4.3 e 4.4 [30] apresentam a modelagem em cenários, diagrama hMSC e respectivos bMSCs, utilizada para esse sistema.

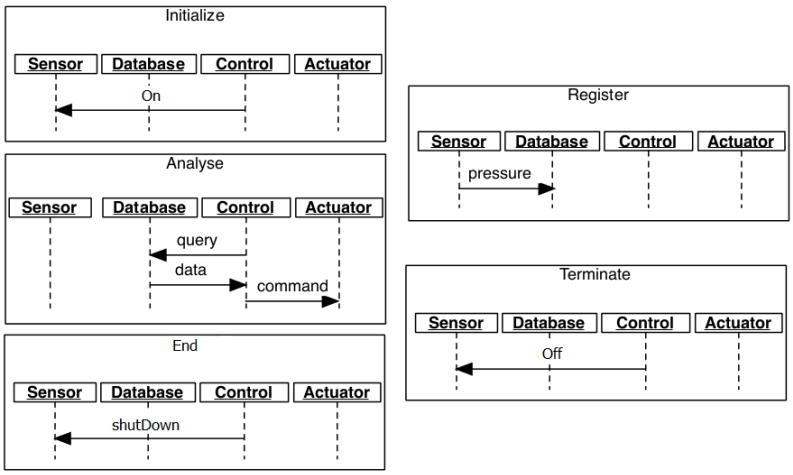


Figura 4.3: Cenários do sistema de gerência de caldeira [30]

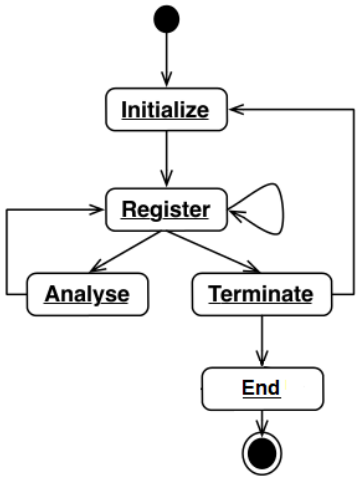


Figura 4.4: Diagrama hMSC do sistema de gerência de caldeira [30]

Com a modelagem em cenários realizada, passa-se para a próxima etapa da metodologia.

4.3 Geração do modelo comportamental

- ***Entrada:*** Modelagem em cenários.

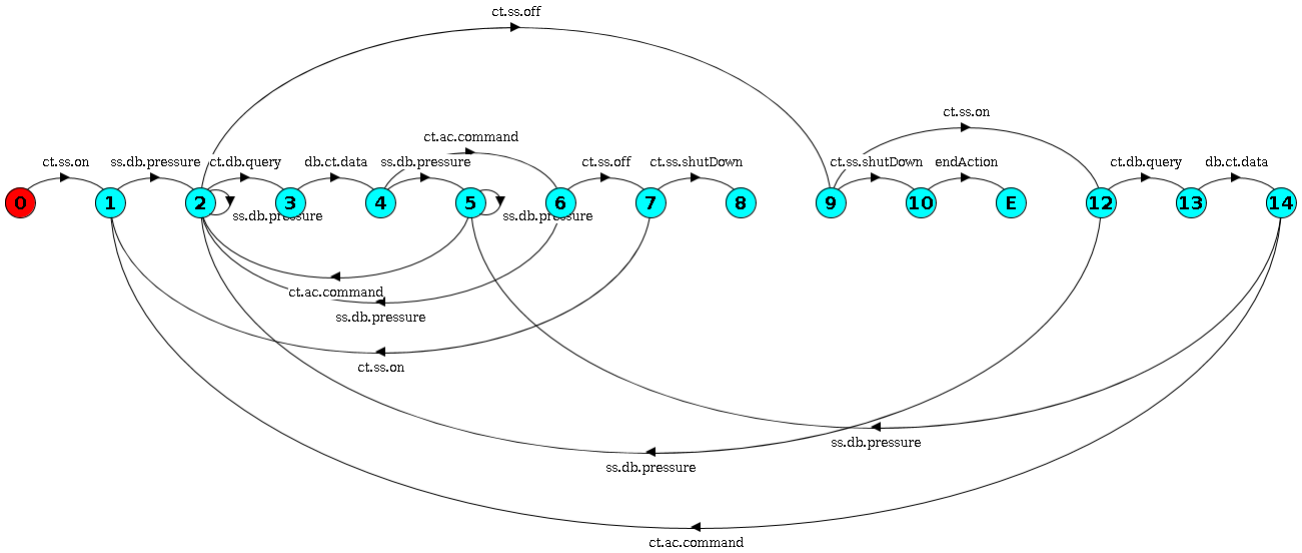


Figura 4.5: Diagrama LTS para o modelo comportamental do sistema de gerência da caldeira

- ***Saída:*** Modelo comportamental expresso em FSP.

Com o sistema modelado em cenários na LTSA-MSC, utilizando uma funcionalidade da ferramenta, obtém-se o modelo comportamental do sistema, representado na notação FSP. Nesse modelo são expressos todos os caminhos possíveis de execução do sistema. O modelo é apresentado na Listagem 4.1.

```
1  ArchitectureModel = Q0,
2  Q0 = (ct.ss.on -> Q1),
3  Q1 = (ss.db.pressure -> Q2),
4  Q2 = (ss.db.pressure -> Q2 | ct.db.query -> Q3 | ct.ss.off -> Q9),
5  Q3 = (db.ct.data -> Q4),
6  Q4 = (ss.db.pressure -> Q5 | ct.ac.command -> Q6),
7  Q5 = (ct.ac.command -> Q2 | ss.db.pressure -> Q5),
8  Q6 = (ss.db.pressure -> Q2 | ct.ss.off -> Q7),
9  Q7 = (ct.ss.on -> Q1 | ct.ss.shutDown -> Q8),
10 Q8 = STOP,
11 Q9 = (ct.ss.shutDown -> Q10 | ct.ss.on -> Q12),
12 Q10 = (endAction -> Q11),
13 Q11 = END,
14 Q12 = (ss.db.pressure -> Q2 | ct.db.query -> Q13),
15 Q13 = (db.ct.data -> Q14),
16 Q14 = (ct.ac.command -> Q1 | ss.db.pressure -> Q5).
```

Listagem 4.1: Modelo comportamental do sistema de gerência de caldeira expresso em FSP

Na Figura 4.5 é apresentada a representação gráfica desse modelo expresso em um diagrama LTS.

Cabe ressaltar que a LTSA-MSC só possibilita a representação do modelo comportamental no diagrama LTS para sistemas com número de estados reduzido. Testes com a ferramenta mostraram que quando o sistema possui mais que 40 estados não é possível a criação do referido diagrama.

O modelo comportamental obtido nessa etapa será utilizado na etapa de análise quantitativa.

4.4 Identificação dos cenários implícitos negativos

- ***Entrada:*** Modelagem em cenários.
- ***Saída:*** Cenários implícitos negativos encontrados.

O propósito dessa etapa é a obtenção dos cenários implícitos cujos efeitos sobre a confiabilidade serão analisados. Será utilizada a funcionalidade da LTSA-MSC para essa identificação.

A Figura 4.6 mostra o primeiro cenário implícito encontrado pela ferramenta e submetido para classificação entre negativo ou positivo.

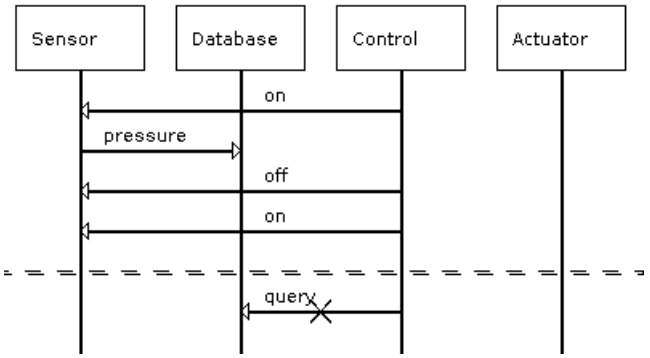


Figura 4.6: Cenário implícito negativo identificado para o sistema de gerência de caldeira

Com o auxílio das Figuras 4.3 e 4.4, verifica-se que se trata de um cenário o qual a última mensagem "*query*", pertencente ao cenário *Analyse*, enviada pelo componente Controlador ao componente *Database*, é antecedida diretamente por uma "*on*", pertencente ao cenário *Initialize*, enviada do componente Controlador ao componente Sensor. Esse cenário contraria a especificação fornecida já que de acordo como hMSC da Figura 4.4, o cenário *Analyse* deve sempre ser precedido do cenário *Register*, impossibilitando a sequência de mensagens ilustradas nesse cenário encontrado. Ou seja, se trata de um cenário passível de ocorrência devido ao funcionamento concorrente de todos os componentes, mas que não estava previsto na modelagem original.

No trabalho que originalmente explorou esse sistema [8], o cenário foi classificado como negativo, visto que ele representaria uma situação na qual haveria leitura de dados potencialmente desatualizados, pois a mensagem "*query*", que consulta o arquivo de dados, não teria aguardado a mensagem "*pressure*" enviada do componente Sensor ao componente *Database* para capturar o valor atual.

Uma análise da arquitetura modelada revela o motivo da ocorrência desse cenário. O componente Controlador não tem a informação de quando o Sensor registra dados no *Database*, então, se é para ser feita uma consulta no componente *Database* após o dado ter sido inserido pelo menos uma vez, é necessário confiar no componente *Database* para habilitar ou desabilitar consultas quando apropriado. Entretanto, como o componente *Database* não pode informar quando o componente Sensor coletou informações, também não pode distinguir o primeiro registro de dados dos demais. Desta forma, não se pode habilitar ou desabilitar consultas apropriadamente [8].

Após a classificação do cenário e inclusão dessa informação na LTSA-MSC, a busca por outros cenários implícitos deve continuar até o esgotamento de novos cenários, pois a metodologia visa identificar o impacto de todos os cenários implícitos do modelo.

As execuções seguintes passam a revelar outros cenários implícitos negativos que possuem a mesma terminação (mensagem "*on*" seguida imediatamente pela mensagem "*query*") mas com variações das mensagens anteriores. A Figura 4.7 mostra dois desses cenários implícitos negativos encontrados.

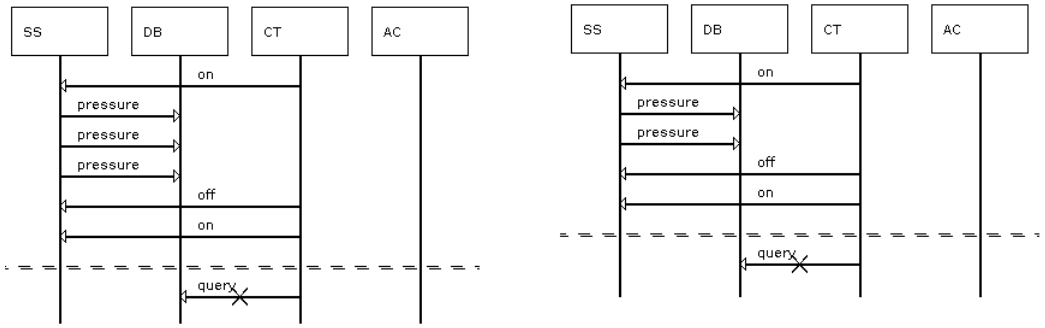


Figura 4.7: Outros cenários implícitos negativos identificados para o sistema de gerência de caldeira

Tendo em vista essas inúmeras combinações de mensagens produzirão inúmeras possibilidades de cenários implícitos com a mesma terminação, torna-se necessária uma análise do modelo comportamental para verificar em que ponto se dá a ocorrência desses cenários implícitos. A forma gráfica do modelo comportamental, expresso na Figura 4.5 facilita a análise e mostra que esse conjunto de cenários implícitos ocorre na transição entre os estados **12** e **13**. Com essa ação busca-se isolar o ponto de ocorrência desse conjunto de

cenários implícitos negativos que serão avaliados. Essa informação será utilizada na etapa abordada na Seção 4.6.2.

Com a detecção dos cenários implícitos, passa-se às análises qualitativa e quantitativa de como eles impactam a confiabilidade.

4.5 Análise qualitativa

- ***Entrada:*** Cenários implícitos negativos identificados.
- ***Saída:*** Possíveis defeitos originados e respectiva caracterização.

Para o entendimento do impacto que os cenários implícitos podem causar na confiabilidade do sistema, a metodologia propõe que sejam caracterizados os defeitos que seriam originados dos cenários implícitos. Essa caracterização seguiria os quatro critérios propostos no significativo trabalho apresentado por Avizienis *et al.* em [18], já abordado na Seção 2.4:

- *domínio*(tempo ou conteúdo);
- *detectabilidade*(detectável ou indetectável);
- *consistência*(consistente ou inconsistente);
- *impacto* (irrelevante a catastrófico).

Na metodologia apresentada, propõe-se uma organização semelhante à Tabela 4.1, na qual são listados alguns defeitos relacionados à confiabilidade causados pelos cenários implícitos identificados, que serão os alvos dessa análise.

Defeito	**Domínio**	**Detectabilidade**	**Consistência**	**Consequência**
Leitura de dados desatualizados	tempo	detectável	inconsistente	grave
Acionamento equivocado do componente Atuador	conteúdo	detectável	inconsistente	grave

Tabela 4.1: Defeitos considerados e suas classificações

Analisando-se o defeito relativo à leitura de dados desatualizados, tem-se que a especificação é clara ao exigir que a leitura do valor da pressão (mensagem "*query*" enviada do componente Controlador ao *Database*) deve ser realizada imediatamente após à gravação de um valor na base (mensagem "*pressure*" enviada do componente Sensor ao *Database*). Entretanto, foi revelada a possibilidade da mensagem "*query*" ser enviada imediatamente após a mensagem "*on*", sem que a mensagem "*pressure*" tivesse sido enviada entre elas.

Essa situação resultaria na leitura de valores possivelmente desatualizados, o que poderia resultar em equívoco na tomada de decisões pelo componente Controlador, responsável por acionar o componente Atuador para o controle da pressão na caldeira, fato que incorreria no outro defeito considerado.

Em cenários implícitos negativos, defeitos de domínio podem ter natureza de tempo e/ou de conteúdo. Isso acontece pois determinadas mensagens tem sua ordem de execução trocada em virtude da intercalação (*interleaving*) entre as mensagens enviadas entre os componentes. Essa troca pode-se caracterizar como um atraso ou antecipação do envio da mensagem e pode gerar um defeito de conteúdo dos dados enviados. O fato da mensagem "*query*" ser enviada imediatamente após a mensagem "*on*", e não após a mensagem "*pressure*" ilustra esse fato, e gera um defeito no domínio de tempo, relativo à leitura de dados desatualizados, e outro no domínio de conteúdo, relativo ao acionamento equivocado do componente Atuador.

No campo da detectabilidade, é possível utilizar mecanismos para detectar quando defeitos oriundos de cenários implícitos acontecem. Por exemplo, verifica-se que as leituras (*query*) da pressão devem ser imediatamente precedidas pela gravação dos valores de pressão no componente *Database*. Do ponto de vista de implementação, podem ser utilizadas técnicas de instrumentação para se detectar essas informações por meio de assertivas.

Quanto à questão da consistência, é preciso relembrar a forma que surgem os cenários implícitos. Eles surgem devido à composição paralela entre os componentes para a criação do modelo comportamental. Nesse processo, as mensagens de cada modelo são intercaladas conforme a natureza paralela distribuída do sistema de software. Em virtude dessas intercalações, novas ordens de execução das mensagens trocadas pelo sistema podem ocorrer de forma assíncrona e aleatória. Dessa forma, o defeito originado seria observado de forma inconsistente pelo usuário do sistema. Dado que esse tipo de defeito é um dos mais difíceis de serem diagnosticados, essa é uma contribuição crucial da detecção de cenários implícitos negativos para a análise de confiabilidade de sistemas.

Por fim, com relação ao impacto, dado que o requisito principal do sistema é, justamente, manter a pressão em determinado nível, uma decisão equivocada do componente Controlador, baseada, por exemplo, em informações desatualizadas, poderia ter consequências graves.

Portanto, essa análise evidencia o significado desses cenários implícitos para a confiabilidade do sistema e fornece insumos para a tomada de decisão quanto a um refinamento arquiteural, a ser tratado na próxima Seção.

4.6 Análise quantitativa

Após a caracterização dos defeitos originados pelos cenários implícitos, passa-se à análise quantitativa, que será detalhada nas seções a seguir.

4.6.1 Tradução de FSP para a linguagem do PRSIM

- ***Entrada:*** Modelo comportamental expresso em FSP.
- ***Saída:*** Modelagem seguindo a linguagem da PRISM.

Para essa primeira etapa da análise quantitativa, utiliza-se a descrição em FSP do modelo comportamental do sistema, mostrado na Listagem 4.1, para gerar o modelo probabilístico do sistema e ser inserido na ferramenta PRISM, para a utilização das técnicas de *model checking* referidas na Seção 2.3.

No trabalho [37] é apresentada uma rotina para traduzir um diagrama LTS para uma sentença na linguagem PRISM. Dada a correspondência biunívoca entre a notação FSP e a os diagramas LTS, aplica-se essa rotina para o modelo comportamental do sistema. Dessa forma, a tradução consiste em implementar uma transição para cada par (estado,evento) do FSP de modo que a sentença FSP

$$<estado> = (e_1 \rightarrow Q_1 | \ldots | e_n \rightarrow Q_n)$$

seja traduzida para as ações em PRISM da seguinte forma:

$$[T(e_1)]\ sys\_st = T(<estado>) \rightarrow (sys\_st' = T(Q_1));$$

$$\vdots$$

$$[T(e_n)]\ sys\_st = T(<estado>) \rightarrow (sys\_st' = T(Q_n));$$

de modo que sys_st é a variável para o controle do estado do sistema em questão no PRISM e T é a função bijetora de tradução que deve adaptar os nomes de estados e eventos em FSP à sintaxe da linguagem expressiva do PRISM.

Os parâmetros probabilísticos a serem inseridos no modelo comportamental representarão os valores das confiabilidades individuais de cada componente do sistema. Assim, o cálculo da confibilidade global do sistema será em função das confiabilidades dos componentes. Na tradução, é utilizada a confiabilidade do componente que recebe a mensagem.

A Figura 4.8 mostra a tradução de uma das linhas do modelo comportamental do sistema em questão. Cabe destacar que foi definido um estado de erro do sistema

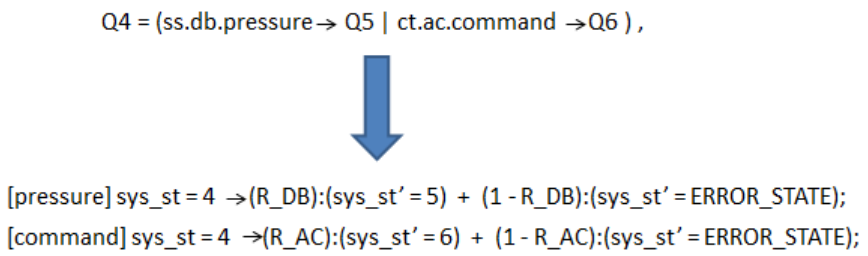


Figura 4.8: Trecho do código em FSP sendo traduzido para a linguagem da PRISM

($ERROR\_STATE$) que é alcançado quando um componente destino falha, tendo, então, a probabilidade sempre igual a $(1 - R\_X)$. Para automatização dessa tarefa, foi desenvolvido um componente em Java.

A Listagem 4.2 é a modelagem em questão já na linguagem para inserção no PRISM.

```
1  dtmc
2
3  const double R_SS; //Confiabilidade do componente Sensor
4  const double R_CT; //Confiabilidade do componente Controlador
5  const double R_AC; //Confiabilidade do componente Atuador
6  const double R_DB; //Confiabilidade do componente Base de Dados
7  const int FINAL_STATE = 11; //Estado final de sucesso do sistema
8  const int ERROR_STATE = 15; //Estado de erro do sistema
9
10 module ArchitectureModel
11
12  sys_st : [0..15] init 0;
13  [CTSSOn] sys_st = 0 -> R_SS:(sys_st'=1) + (1-R_SS):(sys_st'=ERROR_STATE);
14
15  [SSDBPressure] sys_st = 1 -> R_DB:(sys_st'=2) + (1-R_DB):(sys_st'=ERROR_STATE);
16
17  [CTSSOff] sys_st = 2 -> R_SS:(sys_st'=9) + (1-R_SS):(sys_st'=ERROR_STATE);
18  [CTDBQuery] sys_st = 2 -> R_DB:(sys_st'=3) + (1-R_DB):(sys_st'=ERROR_STATE);
19  [SSDBPressure] sys_st = 2 -> R_DB:(sys_st'=2) + (1-R_DB):(sys_st'=ERROR_STATE);
20
21  [DBCTData] sys_st = 3 -> R_CT:(sys_st'=4) + (1-R_CT):(sys_st'=ERROR_STATE);
22
23  [CTACCommand] sys_st = 4 -> R_AC:(sys_st'=6) + (1-R_AC):(sys_st'=ERROR_STATE);
24  [SSDBPressure] sys_st = 4 -> R_DB:(sys_st'=5) + (1-R_DB):(sys_st'=ERROR_STATE);
25
26  [SSDBPressure] sys_st = 5 -> R_DB:(sys_st'=5) + (1-R_DB):(sys_st'=ERROR_STATE);
27  [CTACCommand] sys_st = 5 -> R_AC:(sys_st'=2) + (1-R_AC):(sys_st'=ERROR_STATE);
28
29  [SSDBPressure] sys_st = 6 -> R_DB:(sys_st'= 2) + (1-R_DB):(sys_st'=ERROR_STATE);
30  [CTSSOff] sys_st = 6 -> R_SS:(sys_st'=7) + (1-R_SS):(sys_st'=ERROR_STATE);
31
32  [CTSSOn] sys_st = 7 -> R_SS:(sys_st'=1) + (1-R_SS):(sys_st'=ERROR_STATE);
33  [CTSSShutDown] sys_st = 7 -> (sys_st'=FINAL_STATE);
34
35  [CTSSOn] sys_st = 9 -> R_SS:(sys_st'=12) + (1-R_SS):(sys_st'=ERROR_STATE);
36  [CTSSShutDown] sys_st = 9 -> (sys_st'=FINAL_STATE);
37
38  [CTDBQuery] sys_st = 12 -> R_DB:(sys_st'=13) + (1-R_DB):(sys_st'=ERROR_STATE);
```

```
39 [SSDBPressure] sys_st = 12 -> R_DB:(sys_st'=2) + (1-R_DB):(sys_st'=ERROR_STATE);
40
41 [DBCTData] sys_st = 13 -> R_CT:(sys_st'=14) + (1-R_CT):(sys_st'=ERROR_STATE);
42
43 [CTACCommand] sys_st = 14 -> R_AC:(sys_st'=1) + (1-R_AC):(sys_st'=ERROR_STATE);
44 [SSDBPressure] sys_st = 14 -> R_DB:(sys_st'=5) + (1-R_DB):(sys_st'=ERROR_STATE);
45
46 endmodule
```

Listagem 4.2: Modelagem do sistema de gerência de caldeira para o PRISM

4.6.2 Identificação dos traces referentes aos cenários implícitos negativos

- ***Entrada:*** Modelagem na linguagem da PRISM; Traces que representam os cenários implícitos negativos.
- ***Saída:*** Alterações a serem realizadas no modelo da linguagem da PRISM.

Como o intuito da metodologia é analisar os efeitos dos cenários implícitos, serão realizados experimentos comparando as situações com e sem os efeitos desses cenários. Para o experimento em que não se considera os efeitos, será utilizado o modelo comportamental original do sistema. Já para o experimento que se considerará o efeito, são realizadas alterações no modelo de modo que ao serem executados os traces referentes aos cenários implícitos negativos, o sistema atinja o estado de erro.

Dessa forma, recuperando a informação obtida na Seção 4.4, na qual foram identificados os pontos de ocorrência dos cenários impícitos negativos no modelo comportamental, para representar esse efeito, o modelo PRISM será alterado para que o estado seguinte à sequência **12** $\rightarrow$ **13** seja o estado de erro do sistema. A Listagem 4.3 mostra o trecho a ser alterado na modelagem PRISM.

```
1
2   [DBCTData] sys_st = 13 -> R_CT:(sys_st'=14) + (1-R_CT):(sys_st'=ERROR_STATE); //Modelo
        original
3
4   [DBCTData] sys_st = 13 -> (sys_st'=ERROR_STATE); //Modelo alterado
```

Listagem 4.3: Trecho a ser alterado na modelagem para a representação do efeito do cenário implícito considerado

Portanto, a comparação quanto ao efeito da presença dos cenários implícitos será obtida com os resultados do modelo original e o alterado, que a leva a execução do cenário implícito determinado ao estado de erro.

4.6.3 Análise de sensibilidade

- ***Entrada:*** Modelo inserido na PRISM;
- ***Saída:*** Curva indicando a variação da confiabilidade do sistema em função da confiabilidade dos componentes.

Para verificar o impacto quantitativo da presença de cenários implícitos, a metodologia propõe a realização de uma análise de sensibilidade do modelo.

A importância da análise de sensibilidade é o fato dela revelar como se comporta a confiabilidade do sistema em relação à variação da confiabilidade de cada componente. Com essa análise, é possível identificar quais componentes impactam mais a confiabilidade do sistema, consequentemente, aqueles componentes que precisam de níveis maiores de confiabilidade.

A análise se dá realizando cálculos para a obtenção da confiabilidade global do modelo variando a confiabilidade de um componente por vez, fixando os outros com valor constante, igual a 1. Dessa forma são obtidas curvas para a variação na confiabilidade de cada componente. A curva com maior inclinação será a do componente cuja alteração na sua confiabilidade produzirá o maior impacto sobre a confiabilidade do sistema.

Para a estimativa da confiabilidade do modelo, utiliza-se a seguinte propriedade em PCTL:

$$P = ?\ [F\ (\text{sys\_st}= \text{FINAL\_STATE})]$$

que representa a probabilidade do sistema atingir o estado final de sucesso, no caso do modelo apresentado na Listagem 4.2, o estado 11.

Na Figura 4.9 é mostrada a análise de sensibilidade do modelo considerando a presença dos cenários implícitos negativos. Portanto, foi utilizado o modelo contendo a alteração prevista na Listagem 4.3.

O resultado apresentado na Figura 4.9 mostra que o componente Sensor (R_SS) é o que tem menor influência na confiabilidade global do sistema. As curvas dos componentes Atuador (R_AC) e Controlador (R_CT) se sobrepuseram. Por outro lado, o componente *Database* (R_DB) apresentou-se como o de maior impacto para a confiabilidade do sistema. Dada a demonstração sobre a destacada influência do componente *Database* sobre a confiabilidade geral, ele seria o candidato prioritário a receber mecanismos de tolerância e prevenção a falhas, de modo a aumentar a sua confiabilidade individual.

No entanto, um fator fundamental a ser observado é o fato de que, ainda que todos os componentes do sistema da caldeira tenham uma confiabilidade individual dos componentes a 100%, a confiabilidade global do sistema está limitada em torno do valor de 70%, conforme pode-se observar na Figura 4.9. Isso evidencia de forma quantitativa o efeito do cenário implícito que surge da interação entre os componentes.

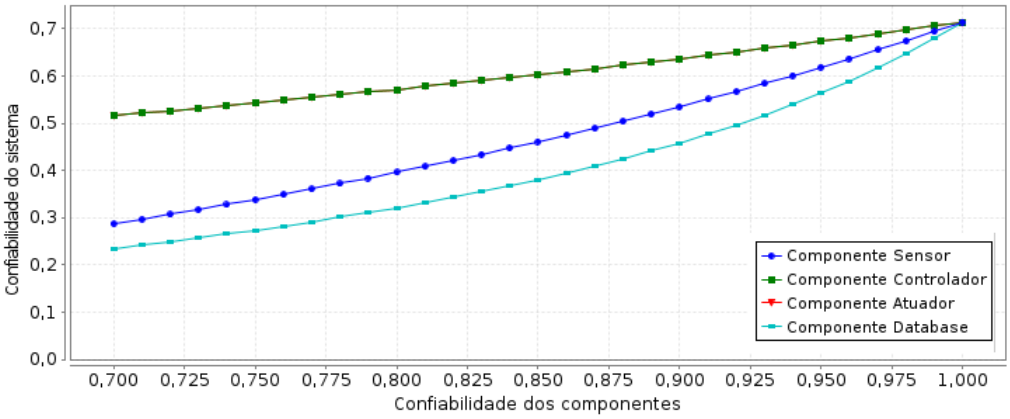


Figura 4.9: Análise de sensibilidade sobre o sistema considerando a presença dos cenários implícitos negativos detectados

4.6.4 Cálculo da confiabilidade

- ***Entrada:*** Modelo inserido na PRISM; Valores das confiabilidades individuais dos componentes.
- ***Saída:*** Valor da confiabilidade do sistema.

Para casos em que informações sobre as confiabilidades individuais dos componentes estejam disponíveis, o cálculo da confiabilidade do sistema pode ser obtido diretamente.

Para o sistema em questão, serão utilizados os valores para a confiabilidade determinados em [30], presentes na Tabela 4.2.

Componente	**Valor da confiabilidade**
Sensor	0.99
Controlador	0.95
Actuator	0.99
Database	0.999

Tabela 4.2: Valor da confiabilidade dos componentes [30]

Para a obtenção do valor previsto para a confiabilidade do modelo, repete-se o mesmo cálculo realizado na Seção anterior, mas agora com os valores específicos de confiabilidade de cada componente. Portanto, será utilizada, novamente, a propriedade PCTL:

$$P = ? \; [F \; (\text{sys\_st=FINAL\_STATE})]$$

.

Então, considerando os valores de confiabilidade descritos na Tabela 4.2, usando a modelagem que desconsidera o efeito dos cenários implícitos negativos, ou seja, a Listagem 4.2, chegou-se ao valor 85,40% para a confiabilidade do sistema.

Quando os efeitos dos cenários implícitos são considerados, utilizando a modificação expressa na Listagem 4.3, chegou-se ao valor de 65,46%. Essa a diferença entre os resultados das duas análises representa o efeito dos cenários implícitos sobre a confiabilidade.

O impacto de cerca de 20% sobre a confiabilidade do modelo é considerável. Quanto maior for essa probabilidade, maior será o impacto do cenário sobre a confiabilidade. Entretanto, para uma tomada de decisão quanto a uma alteração do modelo, é imprescindível que seja compreendido o que isso representa dentro da semântica do sistema, conforme discutido na Seção 4.5.

4.7 Refinamento arquitetural

Seguindo o fluxograma da Figura 4.1, após a realização das análises qualitativa e quantitativa, o projetista tem informações consistentes para definir sobre a necessidade ou não de um refinamento arquitetural. Caso haja a necessidade, pode se seguir dois caminhos:

1. Alterar a modelagem para que os cenários implícitos passem a não ocorrer. Nesse caso, os cenários iniciais sofreriam alguma alteração, sendo essa opção chamada de *refinamento dirigido a cenários*;

2. Manter a modelagem original, mas implementar mecanismos para impedir o efeito dos cenários implícitos. Os mecanismos implementados fariam o papel das *constraints* propostas em [8] e já apresentadas na Seção 3.5. Essa opção será chamada de *refinamento dirigido a constraints*.

4.7.1 Refinamento dirigido a cenários

Nesse caso, a opção é pela alteração do modelo buscando a eliminação dos cenários implícitos considerados. Dado que o surgimento desses cenários implícitos está relacionado com a falta de comunicação entre os componentes Controlador e *Database* causada pela sobreposição de instâncias de execução do sistema, uma possibilidade de solução seria a adição de uma mensagem que fizesse o papel de sincronizar a execução em determinado trecho do modelo de forma a impedir que ocorra intercalação entre instâncias de execução, impedindo assim o surgimento de sequências não esperadas.

No caso, a opção escolhida é adicionar uma mensagem de sincronização (mensagem *Syn*) entre os componentes *Database* e Sensor nos cenários *Initialize* e *Terminate*, conforme ilustrado na Figura 4.10. Dessa forma, a mensagem seria sincronizada e essa região estaria restrita a apenas uma instância de execução.

Com essa nova configuração, atingiu-se com sucesso o objetivo de não haver mais cenários implícitos presentes. Cabe destacar que essa é apenas uma solução possível para a eliminação dos cenários implícitos. A decisão sobre qual alternativa seguir recai sobre o projetista.

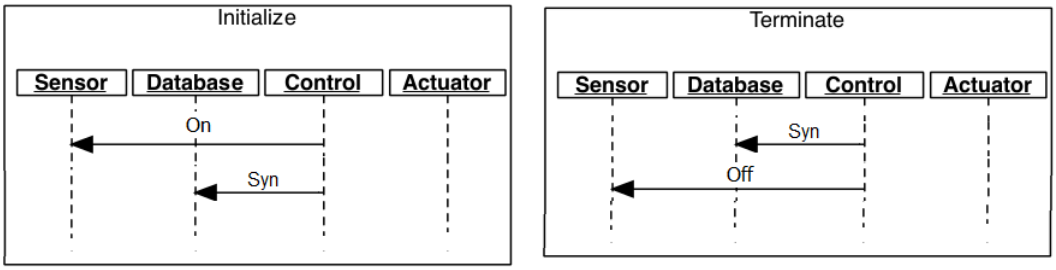


Figura 4.10: Cenário refinados

Ao gerar um novo modelo comportamental e o traduzí-lo para o PRISM utilizando as mesmas confiabilidades dos componentes utilizado no experimento anterior, se chega à confiabilidade de 85,8%.

4.7.2 Refinamento dirigido a *constraints*

Para representar a situação na qual a modelagem foi mantida, mas foram implementados mecanismos para impedir a ocorrência dos cenários implícitos detectados, serão utilizados modelos atuando como *constraints*, que ao serem compostos paralelamente com o modelo original, impedem que os cenários implícitos sejam executados. Na Seção 3.5 foram apresentados os fundamentos desse processo.

Um modelo que mantenha a mesma especificação em cenários, mas seja livre da ocorrência dos cenários implícitos é chamado por Uchitel *et al.* em [8], conforme visto na Seção 3.5, de *Constrained Model*

Considerando que no sistema em questão, o cenário implícito é causado pela ocorrência da mensagem "*query*" antes da mensagem "*pressure*", um modelo para restringir essa ocorrência deve trazer a ordem de execução desejada, isto é: "*pressure*" sempre antes de "*query*".

A listagem 4.4 mostra o modelo da *constraint* que restringe a ocorrência dos cenários implícitos considerados. Ao sincronizar paralelamente com o modelo original, será obtido o *Constrained Model*.

```
1 Constraint = Q0,
2 Q0 = (ct.ss.on->Q1 | {ss.db.pressure, ct.db.query}->Q0),
3 Q1 = (ct.ss.on->Q1 | ss.db.pressure->Q0).
4
5 ||ConstrainedModel = (ArchitectureModel || Constraint).
```

Listagem 4.4: Modelagem em FSP da geração do Constrained Model

É importante destacar que a LTSA-MSC possui a funcionalidade de automatizar a geração do *Constrained Model*. Esse recurso torna-se imprescindível para sistemas mais complexos.

Conforme já dito, o *Constrained Model* simularia uma ação de refinamento arquitetural, que impedisse os efeitos daquele cenário implícito, como o lançamento de uma exceção ao ocorrer algum caminho indesejado ou a implementação de um mecanismo de serialização para essas duas mensagens (*pressure* e *query*). Mas, seria mantida a mesma modelagem em cenários.

Realizando-se os cálculos para a previsão da confiabilidade desse modelo, continuando a usar o mesmo perfil de confiabilidade dos componentes utilizados nos experimentos anteriores, chegou-se ao valor de 86,80%.

4.8 Análise dos resultados

A Figura 4.11 pode ser utilizada para uma análise dos resultados obtidos a partir dos experimentos. Ela apresenta quatro curvas das confiabiliades do sistema representando cada um dos modelos avaliados. São eles:

- cenários implícitos não são considerados;
- cenários implícitos são considerados e levam o sistema a erro;
- refinamento do modelo dirigido a *constraints*(utilização do *Constrained Model*) e;
- refinamento dirigido a cenários (modelo refatorado).

A análise dos dados revela que há uma sobreposição das três curvas que tratam os cenários implícitos, seja desconsiderando a presença, ou seja pelas duas alternativas de refinamento arquitetural, e se diferenciam daquela a qual os cenários implícitos levam o sistema a erro, havendo uma diferenciação à medida que o valor da confiabilidade dos componentes chega ao valor máximo. Observa-se que em todos os casos, a confibiabilidade do sistema em estudo tem um comportamento assintótico.

Dessa forma, observa-se uma distância entre as curvas da modelagem original, que desconsidera os cenários e a que considera e leva o sistema a erro. A primeira curva seria, então, a representação de um modelo com a falsa sensação de estar em um patamar

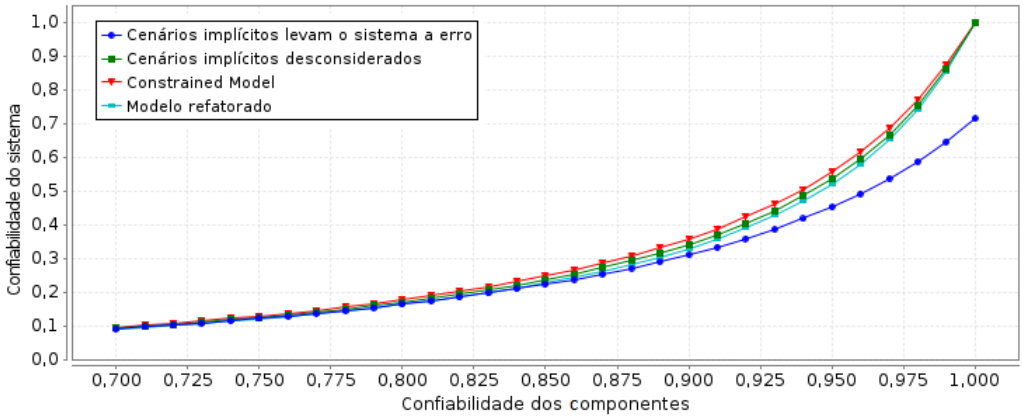


Figura 4.11: Comparação do comportamento da confiabilidade do sistema com relação às confiabilidades dos componentes para 4 situações

de confiabilidade, quando na realidade, o comportamento obtido é o da curva que considera o efeito negativo dos cenários implícitos detectados. Nessa que os erros só serão descobertos quando o sistema estiver em execução. Pode-se comparar ao fato de haver faltas dormentes (*dormant faults*) no sistema. Portanto, por meio dessa curva, observa-se que não será possível atingir na prática uma confiabilidade de 100% do sistema, ainda que seus componentes apresentem uma confiabilidade de 100%. Esse ponto pode ser explicado pelo fato das falhas oriundas de cenários implícitos se deverem a interações que emergem da composição entre os componentes.

Para que o sistema possa, de fato, apresentar sua confiabilidade próxima ao apresentado na primeira curva, ou seja, aquela que desconsidera a presença de cenários implícitos, seriam necessárias ações de refinamento arquitetural, evidenciado pelo resultados nas curvas que as representam. Observa-se na Figura 4.11 que há uma sobreposição entre duas curvas que representam as ações de refinamento. Isso mostra que, nesse caso, a aplicação de qualquer uma das opções de refinamento arquitetural (orientado a cenários ou orientado a *constraints*) tem efeito similiar do ponto de vista quantitativo de confiabilidade.

Complementando a análise, com relação à análise qualitativa, há que se destacar, em particular, o fato dos defeitos oriundos de cenários implícitos terem um comportamento inconsistente. Defeitos desse tipo são de difícil previsão e reprodução, dificultando a obtenção do diagnóstico em tempo de execução. Por meio da metodologia proposta nesse trabalho, torna-se possível tratar esses defeitos desde o estágio inicial do ciclo do desenvolvimento de software.

Capítulo 5

Estudo de Caso

Nesse capítulo será apresentado um estudo de caso sobre a aplicação da metodologia apresentada no Capítulo 4.

O estudo de caso seguirá a estrutura defendida em [43] como sendo a adequada para a estudos empíricos em Engenharia de Software. Dessa forma, o estudo de caso deverá deixar bem definidos os seguintes aspectos:

- Contexto da pesquisa;
- Hipóteses a serem validadas;
- Configuração do experimento;
- Análise e apresentação de dados;
- Resultados e conclusões;
- Ameaças à validade.

5.1 Contexto da pesquisa

Conforme já tratado em pontos anteriores desse trabalho, essa pesquisa foca na análise do impacto dos cenários implícitos sobre a confiabilidade de modelos. Nesse estudo de caso, busca-se, especificamente, utilizar essa análise como subsídio para definições arquiteturais de um sistema com vias a alcançar um maior nível de dependabilidade.

5.2 Descrição do sistema *"Smart Cams"*

O sistema a ser analisado é uma rede de câmeras inteligentes (*Smart Cams*) que compartilham entre si a responsabilidade pelo rastreamento de objetos determinados.

Trata-se de um projeto de pesquisa entre universidades da Áustria[1] e da Inglaterra[2] apresentado preliminarmente em [40] e, integralmente, em [41].

Esse sistema é caracterizado pela ausência de um controle central que gerenciaria a topologia da rede e o relacionamento entre as câmeras. As tarefas são compartilhadas entre as próprias câmeras que estabelecem entre si uma comunicação para esse fim.

O fato do relacionamento entre as câmeras ser definido e operado entre elas, permite a alteração do arranjo da rede de forma dinâmica, inclusive com a adição ou remoção de câmeras ao sistema.

O algoritmo proposto para orientar a forma como as câmeras vão se revezar na tarefa de rastrear os objetos visa uma utilização eficiente de recursos, buscando otimizar a comunicação entre os componentes. Os autores definem essa abordagem como sendo sócio-econômica por seguirem os mecanismos de mercado [40].

De forma resumida, segue a descrição do funcionamento do sistema:

(a) cada câmera do sistema se comunica com as câmeras vizinhas, definidas assim por estarem fisicamente próximas de modo que possam trocar informações. Ou seja, as câmeras não serão todas vizinhas uma das outras necessariamente;

(b) cada câmera gerencia duas listas: uma de rastreamento (que indica os objetos sob sua responsabilidade) e outra de busca (que indica os objetos que ela deve buscar). A presença de um objeto na sua lista de busca significa que não existem câmeras vizinhas rastreando o objeto no momento ou, que a responsável pelo rastreamento está prestes a perdê-lo de vista, em outras palavras, o objeto está prestes a deixar o campo de visão da câmera;

(c) quando um novo objeto é inserido no sistema, um agente central envia às câmeras a ordem para que o coloquem em suas listas de busca. Essa é a única ação desse agente central: informar às câmeras a inserção ou retirada de objetos, a serem rastreados, do sistema;

(d) quando esse novo objeto entra no campo de visão de alguma câmera, essa envia às vizinhas mensagens para que o retirem de suas listas de busca. Nesse momento ela passa a ser a responsável pelo rastreamento do objeto, incluindo-o na sua lista de rastreamento;

(e) quando essa câmera está prestes a perder um objeto, ou seja, ele está saindo do seu campo de visão, ela envia mensagens às câmeras vizinhas para que o objeto seja

[1] Alpen-Adria-Universität Klagenfurt
[2] University of Birmingham

incluído na respectivas listas de busca. Esse momento representa o início do "leilão" que será realizado;

(f) as câmeras que receberam a mensagem e possuem o objeto no seu campo de visão enviam mensagens para a responsável indicando que o objeto está no campo, e portanto vão participar do leilão;

(g) a câmera responsável inicia então o leilão requerindo das câmeras que estão visualizando o objeto os chamados índices de confiança (valores que representam o quanto o objeto está visível para determinada câmera);

(h) após o recebimento dos índices de confiança das câmeras participantes do leilão, a atual responsável transmite a responsabilidade à câmera que apresentou o maior índice, significando que seria a melhor posicionada para rastrear o objeto;

(i) a câmera que recebeu a responsabilidade inclui o objeto na sua lista de rastreamento e envia mensagem às vizinhas para que retirem o objeto das respectivas listas de busca;

(j) quando uma câmera entra no sistema, ela envia mensagens em *broadcast* a fim de obter respostas de quais seriam as câmeras vizinhas, alterando, assim, a topologia da rede de forma dinâmica.

Dado o que foi apresentado, constatam-se as seguintes características para o sistema:

- é decentralizado, não possui um nó central de controle;
- as câmeras não tem conhecimento da topologia completa da rede, mas sim da vizinhança;
- é possível acrescentar ou remover câmeras em tempo de execução;
- existe alto grau de adaptabilidade e flexibilidade para a disposição das câmeras.

Em [41] pode-se encontrar maiores informações sobre o sistema, inclusive a configuração do *hardware* das câmeras.

5.3 Hipóteses

A proposta desse estudo de caso é verificar o impacto que os cenários implícitos podem ter sobre a definição de um arranjo das câmeras. Deseja-se escolher o melhor arranjo, que será chamado de arquitetura, com base na confiabilidade das comunicações entre os componentes do sistema.

Essa análise foi motivada por uma pesquisa conjunta com uma equipe da Universidade de Birminghan (UK). Serão apresentadas duas arquiteturas contendo quatro câmeras e a mais adequada sob o ponto de vista da confiabilidade deverá ser a escolhida. A equipe da universidade britânica avaliará a mesma configuração para decidir qual seria a mais interessante sob o foco das vulnerabilidades de segurança expressas nos cenários implícitos. A análise conjunta sob os focos de confiabilidade e de segurança definiria a melhor escolha a ser realizada.

Diante do exposto até o momento, são hipóteses a serem validadas nesse estudo de caso:

Hipótese I *A presença de cenários implícitos afeta de forma diferenciada cada arquitetura, constituindo peso importante para a definição da arquitetura mais adequada.*

Hipótese II *Os defeitos gerados pelos cenários implícitos afetam de forma relevante o sistema.*

Hipótese III *É possível melhorar a confiabilidade do modelo tratando os cenários implícitos detectados. Visto o impacto que eles proporcionam ao sistema.*

Conforme já comentado nesse capítulo, para a tentativa de validação das hipóteses listadas, será utilizada a metodologia apresentada nesse trabalho. De modo mais específico, a *Hipótese I* será validada com a análise quantitativa do impacto de cenários implícitos. Dadas as características já elencadas do sistema, com destaque para o alto grau de concorrência, a possibilidade da presença de cenários implícitos pode ser grande. Tendo em vista que o arranjo das câmeras pode afetar o grau de concorrência, pressume-se que a probabilidade da ocorrência de cenários implícitos possa ser alterada. Com relação à *Hipótese II*, a análise qualitativa dos defeitos trará essas informações. Contando que os defeitos originados por cenários implícitos são, invariavelmente, inconsistentes, ou seja, com ocorrência imprevisível, dificultando a previsão e reprodução, espera-se que o impacto sobre o sistema seja relevante.

A *Hipótese III* será validada com a análise dos cenários implícitos negativos identificados. Sabendo a origem da ocorrência será possível propor implementações que alterem os resultados para a confiabilidade do sistema.

5.4 Configuração do experimento

Para a realização desse estudo de caso foi disponibilizado um simulador desse sistema, implementado na linguagem Java *(21 classes com 7695 linhas de código)*. A análise de

dependabilidade realizada sobre o simulador visa trazer essa tarefa para os estágios iniciais do desenvolvimento.

A implementação do simulador se deu como parte inicial do projeto do sistema e foi apresentado em [40]. Ele reproduz as iterações do algoritmo que orienta a troca de informações entre os componentes do sistema. Dessa forma, a análise do comportamento do simulador fornecerá subsídios para a melhoria do sistema real. A Figura 5.1 apresenta a tela do simulador.

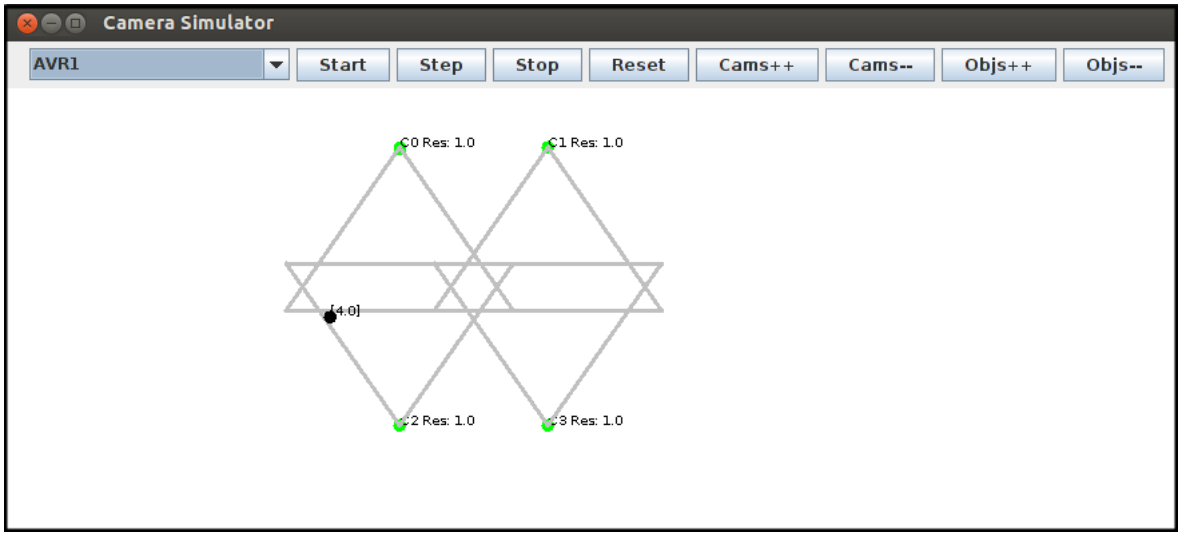


Figura 5.1: Interface do simulador do sistema.

Para a especificação dos cenários a serem modelados, foi realizada uma análise do código do simulador para a identificação das mensagens trocadas entre os componentes. A lista a seguir mostra aquelas que serão utilizadas na modelagem dos cenários:

(a) *StartSearch*: Mensagem enviada às câmeras vizinhas para que adicionem o objeto em suas listas de busca;

(b) *Sees*: Mensagem que indica o momento em que o objeto entra no campo de visão da câmera;

(c) *StartTracking*: Mensagem que indica o momento em que uma câmera passa a ser a responsável por rastrear o objeto.

(d) *Looses*: Mensagem enviada às câmeras vizinhas pela câmera que atualmente rastreia o objeto, indicando o momento em que ele está prestes a deixar o campo de visão.

(e) *Found*: Mensagem enviada pelas câmeras que receberam a mensagem *"Looses"* e possuem o objeto no respectivo campo de visão. O destino da mensagem é a câmera enviou a mensagem *"Looses"*;

(f) *AskConfidence*: Mensagem enviada a uma câmera perguntando o índice de confiança dela com relação ao objeto;

(g) *SendConfidence*: Mensagem utilizada para responder à pergunta *AskConfidence* enviada, enviando o índice de confiança como argumento.

(h) *AllowsTrack*: Mensagem enviada a uma câmera informando que ela é a nova responsável por rastrear o objeto;

As arquiteturas a serem avaliadas estão expressas na Figura 5.2. Ambas contam com quatro câmeras e com a mesma configuração quanto às câmeras vizinhas, isto é, aquelas que se comunicam entre si, conforme a Tabela 5.1.

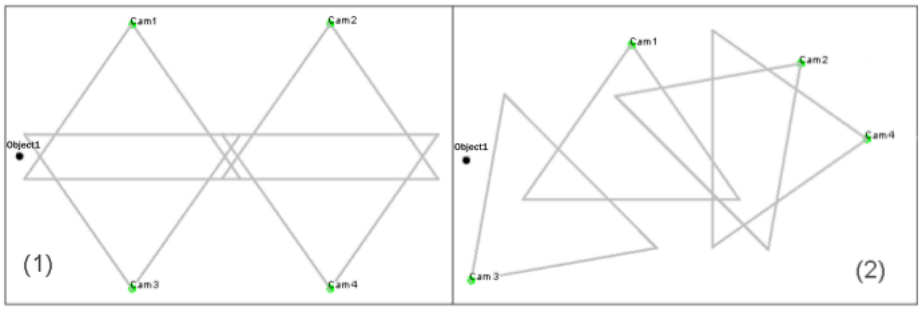


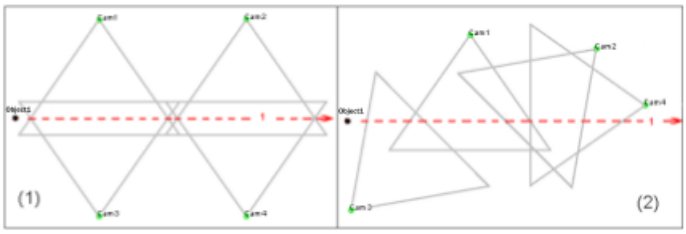
Figura 5.2: arquiteturas a serem analisadas

Câmeras	**Respectivos vizinhos**
1	2,3
2	1,4
3	1
4	2

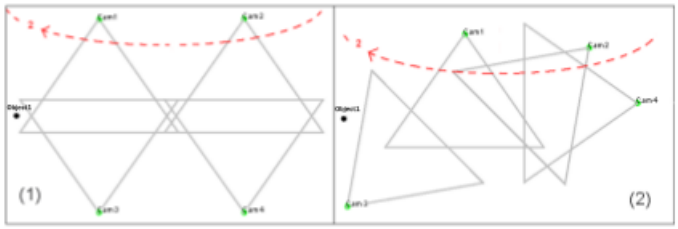
Tabela 5.1: Configuração da vizinhança de cada câmera

A definição das câmeras vizinhas seguiu uma sugestão apresentada em [38]. Entende-se que a utilização de uma mesma configuração de vizinhança nas duas arquiteturas avaliadas é importante para um melhor diagnóstico de qual opção seria a mais interessante, visto que o que estaria em questão seria apenas a diferença entre as faixas cobertas pelos campos de visão de cada câmera. Entretanto, nada impediria que a análise fosse realizada com configurações de vizinhança diferentes.

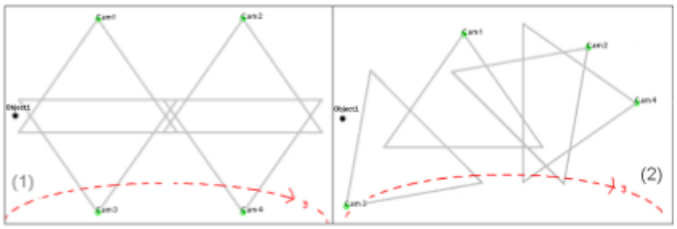
Para a modelagem do sistema em cenários foi proposto que se capturassem cenários a partir da trajetória do objeto na rede de câmeras. Então, foram configurados cinco



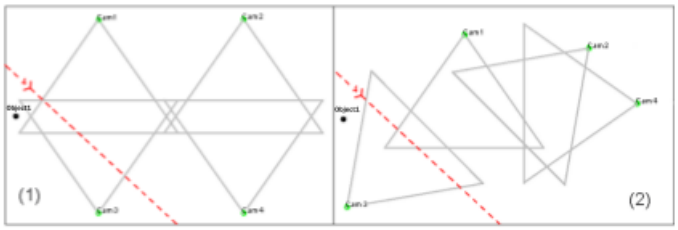
(a) Objeto atravessa o campo de visão das quatro câmeras tanto na Arch1 como na Arch2.



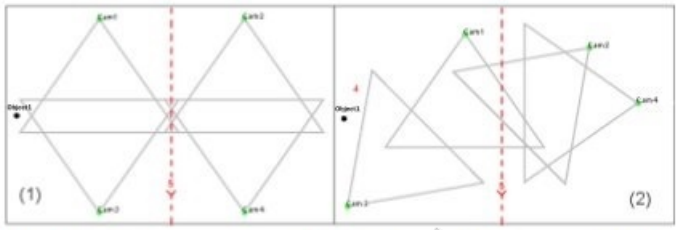
(b) Objeto atravessa o campo de visão das Câmeras 1 e 2 na Arch1 e das câmeras 4, 2 e 1 na Arch2.



(c) Objeto atravessa o campo de visão das câmeras 3 e 4 na Arch1 e das câmeras 3, 2 e 4 na Arch2.



(d) Objeto atravessa o campo de visão das câmeras 1 e 3 tanto na Arch1 quanto na Arch2.



(e) Objeto atravessa o campo de visão das quatro câmeras na Arch1 e das câmeras 1 e 2 na Arch2.

Figura 5.3: Cenários utilizados na modelagem. Eles foram aproveitados de [38]

cenários básicos para cada arquitetura dispostos de forma paralela. Cada cenário rep-

resenta a trajetória de um objeto entre as câmeras, indicado pela linha tracejada em vermelho. A Figura 5.3 apresenta os diversos cenários utilizados, aproveitados de [38].

A Figura 5.4 apresenta o diagrama bMSC para o cenário 1 da primeira alternativa de arquitetura. O conjunto completo dos diagramas bMSC dos cenários especificados está disponível no Anexo A.

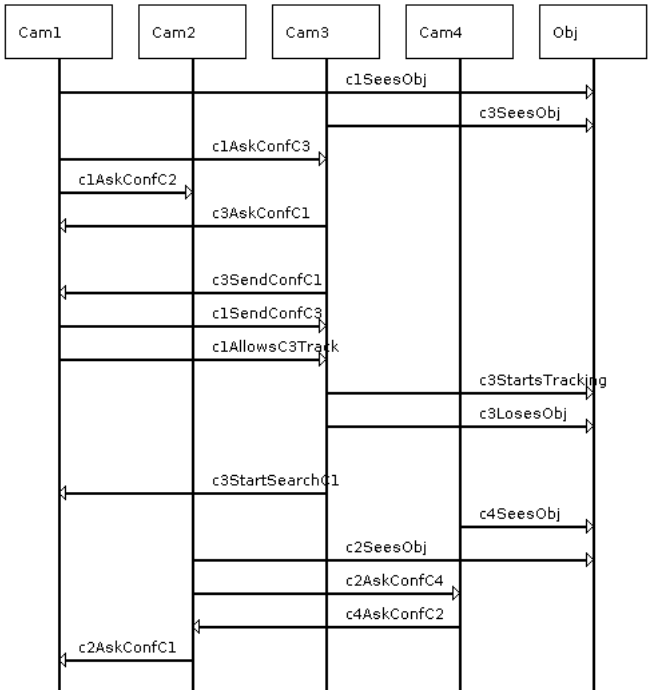


Figura 5.4: Representação em MSC do cenário 1 da arquitetura 1.

5.5 Análise e apresentação de dados

Com base nas informações levantadas sobre o sistema, passa-se à aplicação da metodologia proposta.

A primeira etapa é a obtenção do modelo comportamental a partir dos cenários especificados e inseridos na ferramenta LTSA-MSC. Em seguida são identificados os cenários implícitos. A partir daí passa-se a avaliar o impacto desses novos cenários na confiabilidade do modelo realizando-se as análises quantitativa e qualitativa. E, finalmente, as propostas de melhoria arquitetural.

As seções seguintes detalham as etapas.

5.5.1 Obtenção do modelo comportamental

Para cada uma das arquiteturas a serem avaliadas é gerado o modelo comportamental descrito em FSP a partir dos cenários especificados.

Dado o tamanho dos modelos gerados, 95 estados para a arquitetura 1 e 70 para a arquitetura 2, para uma melhor organização, optou-se por deixá-los disponíveis nos Anexos B.1 e B.2, respectivamente.

5.5.2 Identificação dos cenários implícitos

Com o modelo comportamental de cada arquitetura, passa-se à identificação dos cenários implícitos.

Conforme já apresentado nas Seções 3.4 e 4.4, o processo de identificação de cenários implícitos na LTSA-MSC é iterativo. A cada execução é apresentado um cenário para que o usuário informe se é um cenário positivo ou negativo. Já que o atributo a ser avaliado está sendo a confiabilidade, os cenários serão classificados como negativos quando o comportamento contrariar o especificado para o sistema, desviando do correto funcionamento.

Realizada a busca, foram encontrados resultados diferentes para as duas arquiteturas. Para a arquitetura 1 foram encontrados um total de quatro cenários implícitos, sendo todos negativos. Enquanto que para a arquitetura 2, foram encontrados 52 cenários implícitos, sendo que apenas 1 classificado como negativo, e o restante todos positivos.

Na Figura 5.5 é apresentado um dos cenários implícitos negativos identificados para a arquitetura 1, que para fins de facilitar a organização, será denominado IS1. Analisando-se o cenário, verifica-se que o objeto entra no campo de visão das Câmeras 1 e 3, disparando as mensagens *"Sees"* referentes. Seguindo o protocolo, essas câmeras iniciam as pesquisas aos seus vizinhos, com a mensagem *"AskConfidence"*. Conforme a Tabela 5.1, a Câmera 1 deve enviar às Câmeras 2 e 3, enquanto a Câmera 3 deve enviar à Câmera 1. Entretanto, verifica-se que a Câmera 3 inicia o rastreamento do objeto (mensagem *"StartTracking"*) sem aguardar a resposta da sua vizinha, a Câmera 1, que tinha o objeto em seu campo de visão, e lhe responderia com uma mensagem *"Send Confidence"*, enviando o seu índice de confiança.

Essa sequência de mensagens contraria a especificação, visto que as câmeras devem tomar a decisão sobre o rastreamento ou não, após receber as informações de todas as câmeras vizinhas. Fato que impacta a confiabilidade do sistema.

Os outros três cenários implícitos negativos para a arquitetura 1 (IS2, IS3 e IS4) são variações do apresentado na Figura 5.6. O fato gerador é o mesmo, com a diferença que é a Câmera 2 que inicia o rastreamento sem ter recebido a resposta das suas vizinhas sobre

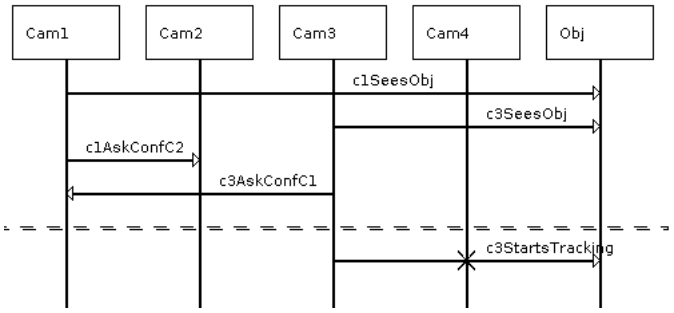


Figura 5.5: Cenário implícito negativo encontrado para a arquitetura 1(IS1)

o índice de confiança. O que varia com relação aos três cenários é a ordem de mensagens anteriores.

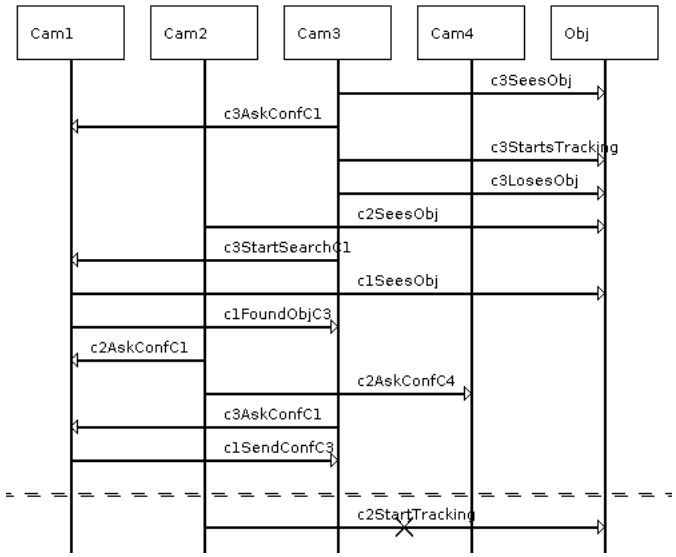


Figura 5.6: Segundo cenário implícito negativo(IS2) identificado para a arquitetura 1

Já o cenário implícito negativo referente à arquitetura 2, denominado IS5, é mostrado na Figura 5.7, a qual apresenta a Câmera 2 iniciando o rastreamento do objeto (mensagem *"StartTracking"*) sem receber o índice de confiança da Câmera 4 que possui o objeto em seu campo de visão, fato evidenciado pela ocorrência da mensagem *"Sees"* enviada por ela. Verifica-se então que se trata de um cenário implícito com a mesma natureza dos que foram identificados para a arquitetura 1.

Na Figura 5.8 é apresentado o diagrama MSC que ilustra um cenário implícito identificado para a arquitetura 2 que foi classificado como positivo por não expressar nenhuma situação que prejudicasse a confiabilidade do sistema, mas apenas mensagens em ordens

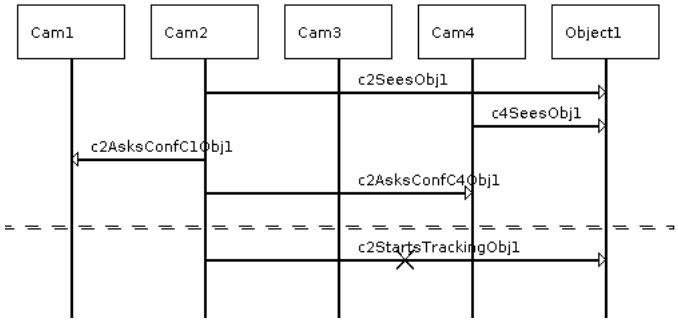


Figura 5.7: Cenário implícito negativo encontrado para a arquitetura 2(IS5)

que não estavam previstas na modelagem inicial. Verifica-se que a ordem das mensagens segue o previsto na concepção do modelo.

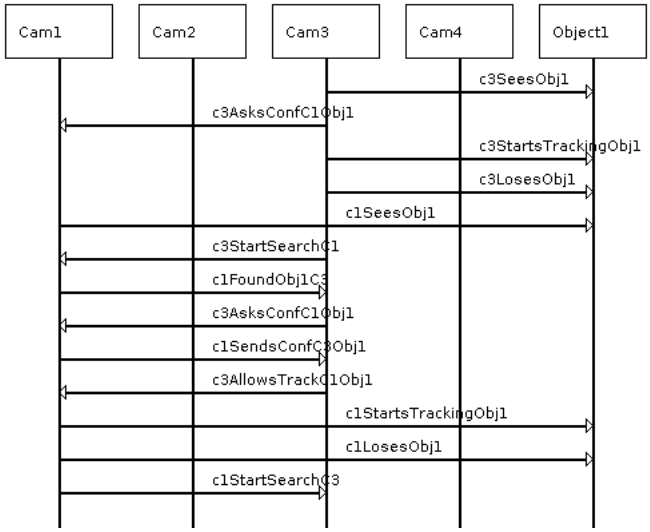


Figura 5.8: Cenário implícito positivo encontrado para a arquitetura 2

Percebe-se que os cinco cenários implícitos ocorrem quando faltam informações das câmeras sobre a presença ou não do objeto no campo de visão das suas vizinhas.

Esses cenários negativos poderiam resultar em uma tomada de decisão equivocada do sistema, levando uma câmera a rastrear o objeto sem ter a posição adequada para isso. Esse fato iria contra um dos princípios do sistema que é a utilização eficiente dos recursos disponíveis. Seguramente, afetando a confiabilidade do sistema.

Com os cenários implícitos devidamente identificados, passa-se à avaliação de como os cenários negativos impactam a confiabilidade do sistema. A primeira análise será a qualitativa. Nela serão relacionados defeitos do sistema, portanto observáveis aos usuários, que poderiam resultar da execução desses cenários.

5.5.3 Análise qualitativa

Passando-se à análise qualitativa do efeito desses cenários sobre a confiabilidade do sistema, o primeiro passo é o levantamento de eventuais defeitos, isto é, os comportamentos percebidos pelos usuários que desviam daqueles esperados.

Considerando o domínio de aplicação do sistema, cita-se três potenciais defeitos que estariam relacionados com a ocorrência dos cenários implícitos negativos detectados:

- Câmera com menor índice de confiança é escolhida para rastrear o objeto;
- Duas câmeras vizinhas rastreiam o objeto simultaneamente;
- Objeto não é rastreado por nenhuma câmera apesar de estar no campo de visão de alguma(s).

O fato de uma câmera não aguardar a resposta do índice de confiança de alguma vizinha que eventualmente possua o objeto em seu campo de visão, a leva a desconsiderar que haja outra câmera capaz de rastrear o objeto no momento. Esse fato é maléfico para a confiabilidade do sistema pois implicaria, entre outras coisas, no desperdício de recursos, pois uma câmera com condições de rastrear o objeto estaria sendo excluída da transação, levando a escolha de uma que possa não ter o maior índice de confiança. Vale ressaltar que um dos diferenciais desse sistema é, justamente, a utilização eficiente dos recursos, baseada no fato de que a câmera que estaria rastreando o objeto, seria sempre aquela melhor posicionada, ou seja, com maior índice de confiança.

Considerando que duas câmeras vizinhas tenham, simultaneamente, o comportamento de não aguardar a resposta das vizinhas e iniciar o rastreamento, chega-se à situação na qual câmeras vizinhas estariam rastreando o objeto no mesmo momento, contrariando um dos requisitos do sistema, que impede que câmeras vizinhas rastreiem simultaneamente o mesmo objeto. Cabe destacar que o sistema permite que câmeras rastreiem simultaneamente o objeto, desde que não sejam vizinhas.

O terceiro defeito relacionado seria o fato do objeto não ser rastreado por nenhuma câmera, apesar de estar no campo de visão de alguma(s). Esse caso poderia ocorrer em um momento de transição de responsabilidades. Conforme já apresentado, a especificação preconiza que a câmera que detém a responsabilidade pelo rastreio, no momento em que o objeto está prestes a sair, é quem efetua a operação de escolha de qual câmera, dentre as que possuem o objeto no respectivo campo de visão e lhe enviaram o índice de confiança, deve ser a escolhida para continuar o rastreamento. Caso o comportamento de ignorar a mensagem de resposta seja executado, a câmera responsável no momento pode interpretar que não existe câmera que possua o objeto em seu campo de visão, deixando de realizar a transferência de responsabilidade. Com isso, o objeto ficaria durante alguns instantes sem

uma câmera responsável, apesar de estar em um campo de visão. Essa situação persistiria até que aquela que o possuísse em seu campo de visão o identificasse e negociasse com os vizinhos a obtenção da responsabilidade pelo rastreio.

Feita a identificação dos defeitos que podem ocorrer devido aos cenários implícitos considerados, é apresentada a Tabela 5.2 que contempla a caracterização desses defeitos seguindo os critérios apresentados em [18].

Defeito	Domínio	Detectabilidade	Consistência	Consequência
Câmera com menor índice de confiança é escolhida para rastrear o objeto	conteúdo	detectável	inconsistente	grave
Duas câmeras vizinhas rastreiam o objeto simultaneamente	tempo	detectável	inconsistente	média
Objeto não é rastreado por nenhuma câmera apesar de estar no campo de visão de alguma(s)	conteúdo	detectável	inconsistente	grave

Tabela 5.2: Defeito e respectivas caracterizações

Tendo em vista que, conforme já abordado na Seção 3.3, os cenários implícitos surgem devido à integração dos diversos cenários por meio da composição paralela, ocasionando uma intercalação entre as diversas mensagens dos modelos, a ocorrência ou não do cenário implícito se torna um problema não determinístico. Isto é, sem a possibilidade da previsão determinante. Dessa forma, os defeitos gerados dos cenŕios implícitos são classificados, quanto à consistência, como inconsistentes, trazendo uma grande preocupação, dado que é um comportamento aleatório que poderia acontecer ou não.

5.5.4 Análise quantitativa

Concluída a análise qualitativa, passa-se à avaliação do impacto quantitativo dos cenários implícitos sobre a confiabilidade do sistema.

Conforme previsto na metodologia, para a análise quantitativa, a partir do modelo comportamental obtido na Seção 5.5.1, visando a obtenção do modelo probabilístico, foi acrescentada ao modelo a variável R_C para representar a confiabilidade das câmeras. O componente objeto, dado que é um componente passivo, terá a sua confiabilidade sempre igual a 1. A confiabilidade do sistema será obtida com base nos valores dessa variável.

Os modelos na linguagem do PRISM, que seguiram a tradução apresentada na Seção 4.6.1, estão disponíveis nos Anexos C.1 e C.2, para as arquiteturas 1 e 2, respectivamente.

Para a consideração do efeito dos cenários implícitos negativos sobre a arquitetura 1, serão alteradas as linhas 18, 117, 122 e 125 do modelo PRISM, presentes no Anexo C.1, conforme mostrado na Listagem 5.1.

```
1  ...
2  [c3StartTracking] sys_st = 4 -> (sys_st'=ERROR_STATE);
3  ...
4  [c2StartTracking] sys_st = 86 -> (sys_st'=ERROR_STATE);
5  ...
6  [c2StartTracking] sys_st = 89 -> (sys_st'=ERROR_STATE);
7  ...
8  [c2StartTracking] sys_st = 91 -> (sys_st'=ERROR_STATE);
9  ...
```

Listagem 5.1: Alteração da modelo PRISM da arquitetura 1 para a representação do efeito dos cenários implícitos negativos considerados.

Com essa alteração, quando o sistema chegar nesse estado, será encaminhado a um estado de erro, caracterizando o efeito negativo do cenário.

Para a arquitetura 2, como foi identificado apenas 1 cenário implícito negativo, apenas a linha 42 do modelo PRISM, presente no Anexo C.2 foi alterada conforme indicado na Listagem 5.2.

```
1  ...
2  [c2StartTracking] sys_st = 15 -> (sys_st'=ERROR_STATE);
3  ...
```

Listagem 5.2: Alteração da modelo PRISM da arquitetura 2 para a representação do efeito do cenário implícito negativo considerado.

Conforme previsto na metodologia, será utilizada a propriedade de alcançabilidade do estado final de sucesso para a caracterização da confiabilidade. Tendo em vista que os estados finais dos modelos das arquiteturas 1 e 2 são 19 e 13, respectivamente, as propriedades PCTL a serem utilizadas são:

$$P = ? \; [F \; (\text{sys\_st}{=}19)]$$

para a arquitetura 1, e:

$$P = ? \; [F \; (\text{sys\_st}{=}13)]$$

para a arquitetura 2.

Utilizando os modelos e propriedades PCTL já apresentadas, são obtidas as Figuras 5.9 e 5.10 que mostram as análises de sensibilidade da confiabilidade do sistema com relação à confiabilidade das câmeras para as duas arquiteturas.

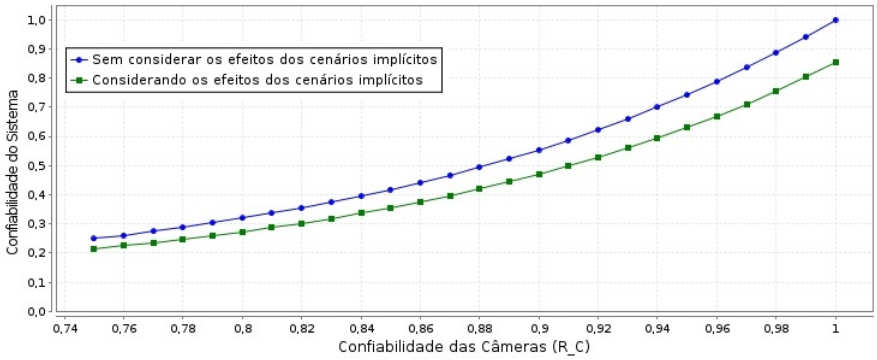


Figura 5.9: Curva comparativa relativa à arquitetura 1

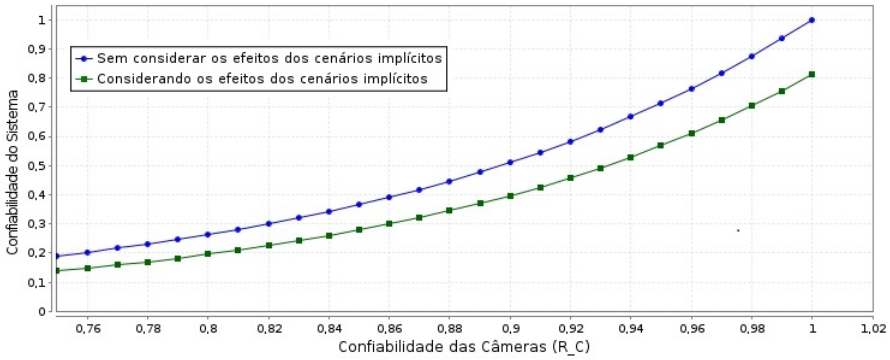


Figura 5.10: Curva comparativa relativa à arquitetura 2

As duas curvas em cada caso representam as situações que consideram ou não os efeitos dos cenários implícitos negativos, visando evidenciar o efeito da presença deles para a confiabilidade do sistema.

Para auxiliar a análise, a Tabela 5.3 mostra resultados discriminados da confiabilidade do sistema para quatro valores da confiabilidade das câmeras (R_C).

Considera IS?	**Arquitetura 1**				**Arquitetura 2**			
	R_C=0.90	R_C=0.95	R_C=0.99	R_C=1.0	R_C=0.90	R_C=0.95	R_C=0.99	R_C=1.0
Não	0.55	0.74	0.94	1.00	0.51	0.71	0.93	1.00
Sim	0.47	0.63	0.80	0.85	0.40	0.57	0.76	0.81

Tabela 5.3: Resultados para as diferentes confiabilidades das câmeras

Analisando os resultados, verifica-se que a arquitetura 1 alcançou valores maiores para a confiabilidade do sistema tanto quando foi considerado os efeitos dos cenários implícitos, como quando não foi.

Para extrair mais informações desses resultados, propõe-se o cálculo da diferença relativa entre os dois valores de confiabilidade para cada valor de R_C em cada arquitetura. A fórmula seria:

$$Dif\_Relativa = \frac{R_{semIS} - R_{comIS}}{R_{semIS}}$$

onde R_{semIS} seria o valor da confiabilidade no experimento que não considera a presença dos cenários implícitos, e R_{comIS}, por sua vez, seria o valor da confiabilidade no experimento que considera a presença dos cenários implícitos.

Os resultados desse cálculo são apresentados na Tabela 5.4:

	Arquitetura 1				**Arquitetura 2**			
	R_C=0.90	R_C=0.95	R_C=0.99	R_C=1.0	R_C=0.90	R_C=0.95	R_C=0.99	R_C=1.0
DifRelativa	0.15	0.15	0.15	0.15	0.21	0.20	0.18	0.19

Tabela 5.4: Resultados para o cálculo da diferença relativa entre os valores de confiabilidade para cada um dos níveis de R_C.

Os valores da diferença relativa revela qual arquitetura sofreu o maior impacto. Fica evidente que a arquitetura 2 foi a mais impactada, visto que os valores da diferença variaram entre 0.19 a 0.21, enquanto, os valores relativos à arquitetura 1 ficaram constantes no valor de 0.15.

Cabe destacar que o impacto foi maior na arquitetura 2 apesar do número menor de cenários implícitos que o apresentado na arquitetura 1. O fato que poderia explicar esse resultado seria a maior área de intersecção dos campos de visão das câmeras quando posicionadas segundo a arquitura 2, o que levaria a uma maior possibilidade de trocas de responsabilidades, ressaltando assim o efeito do cenário implícito negativo.

5.5.5 Refinamento arquitetural

Tendo em vista o impacto quantitativo dos cenários implícitos sobre a confiabilidade do modelo foi alta e as execuções deles consistiriam em defeitos relevantes, qual seria o impacto em cada uma das arquiteturas se fossem implementados mecanismos para impedir que esses cenários implicítos ocorressem?

Conforme preconiza a metodologia, seriam possíveis dois caminhos. Um que altera o modelo original de forma a evitar que os cenários implícitos negativos existam, isto é, o refinamento dirigido a cenários, e o outro que atua mantendo a modelagem original mas implementando mecanismos que impedem que os cenários implícitos ocorram, o refinamento dirigido a *constraints*.

Refinamento dirigido a cenários

Sobre o refatoramento do modelo, o foco é na busca por propostas de correções para que o modelo deixe de apresentar o comportamento expresso nos cenários implícitos negativos detectados.

Nesse sentido, conforme já indicado na Seção 5.5.2, os cinco cenários implícitos negativos detectados ocorrem pela mesma razão: uma câmera sem ter a informação de que uma vizinha possui o objeto em seu campo de visão, desconsidera a resposta relativa ao índice de confiança que que receberia, e inicia o rastreamento. Analisando-se o algoritmo proposto, verifica-se que o envio da mensagem *"Send Confidence"* está condicionada à presença do objeto no campo de visão da câmera que recebeu a mensagem *"Ask Confidence"*, de modo que quando a câmera não possui o objeto em seu alcance, o algoritmo prevê que a câmera não envie a mensagem. Por trás dessa decisão de projeto, estaria a economia de recursos, visto que menos mensagens seriam geradas, reduzindo a comunicação entre as câmeras ao mínimo necessário; entretanto, essas alternativas de envio ou não da mensagem podem gerar problemas à câmera que solicitou essa informação. Dependendo do tempo que levaria a informação a chegar, poderia ser interpretado como a ausência do objeto no campo de visão e a consequente desconsideração do índice de confiança enviado.

Portanto, para evitar esse problema, detectado por um cenário implícito, sugere-se uma alteração no algoritmo de forma que as câmeras vizinhas sempre tenham que responder à mensagem *"Ask Confidence"* independente da presença ou não do objeto no campo de visão. No caso do objeto não estar no campo de visão, a câmera responderia com o índice de confiança igual a zero.

Para ilustrar essa ação de refatoramento, a Figura 5.11 apresenta como ficaria o cenário 2 da arquitetura 1 após essa alteração. Na figura são destacados os locais nos quais seriam adicionados as novas mensagens *"Send Confidence"*. Destaca-se que nos cinco cenários da arquitetura 1 foram encontradas oportunidades para a inclusão dessa mensagem.

Ao ser realizada essa implementação na arquitetura 1, os cenários negativos deixaram de acontecer, entretanto surgiram 12 cenários positivos. Resultado esperado dada a inclusão de novas mensagens.

Para a arquitetura 2, a implementação do refatoramento descrito foi necessária, também, nos cinco cenários. A Figura 5.12 mostra a alteração realizada no cenário 3. Após a implementação, foram identificados 61 cenários implícitos, todos positivos.

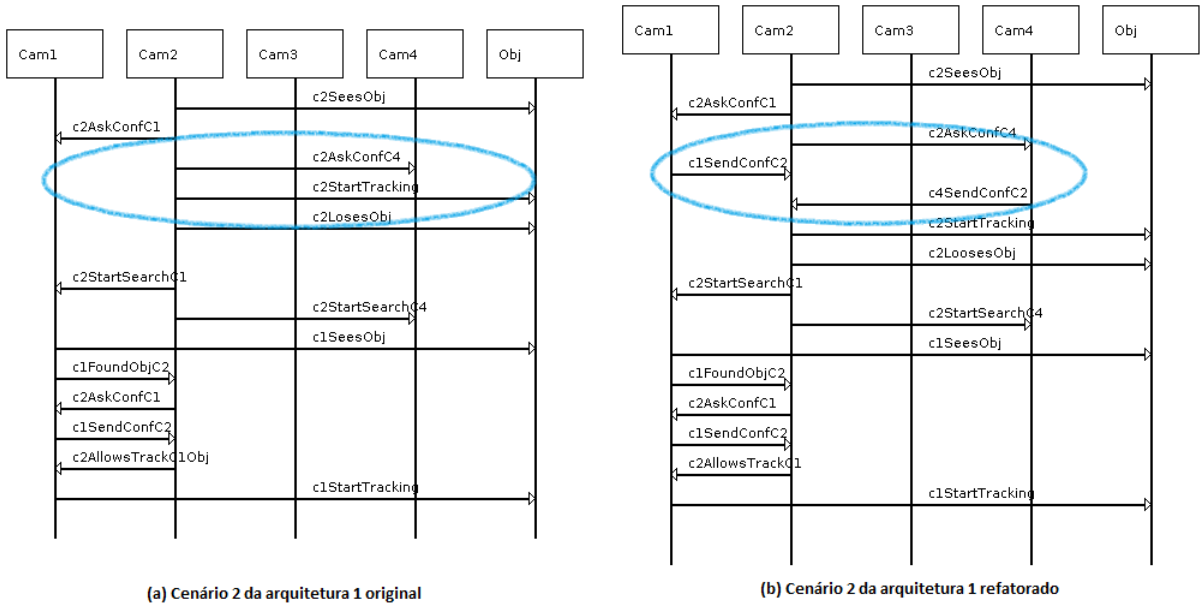


Figura 5.11: Cenário 2 pertencente à arquitetura 1 refatorado

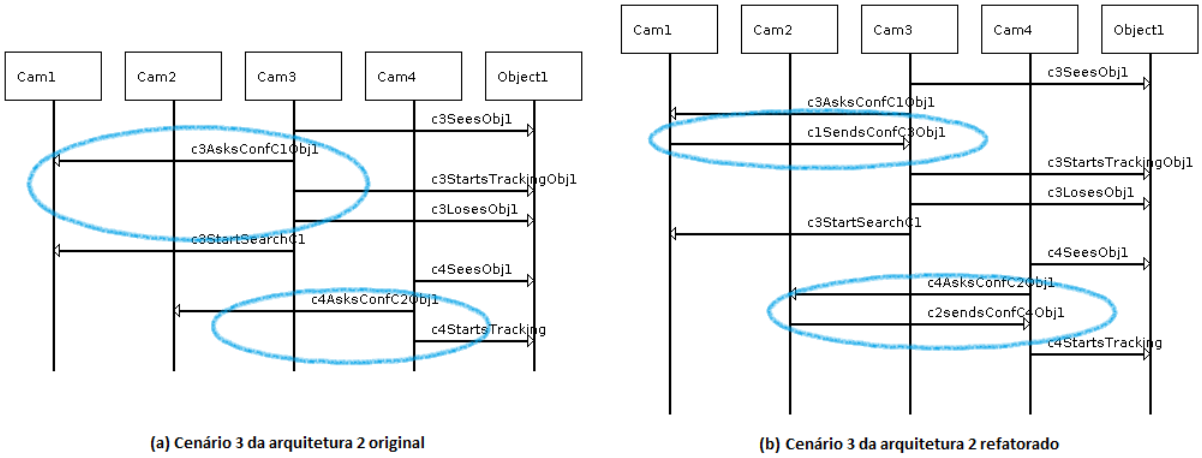


Figura 5.12: Cenário 3 pertencente à arquitetura 2 refatorado

Refinamento dirigido a *constraints* (obtenção do *Constrained Model*)

Conforme já expresso, a LTSA-MSC possui a funcionalidade de gerar as *constraints* referentes a cada cenário implícito negativo detectado. Em seguida, elas são utilizadas para a composição do *Constrained Model*.

Nos Anexos D.1 e D.2, estão presentes os *Constrained Models*, em FSP, para as arquiteturas 1 e 2, respectivamente. Enquanto nos Anexos E.1 e E.2, estão presentes respectivos modelos para o PRISM.

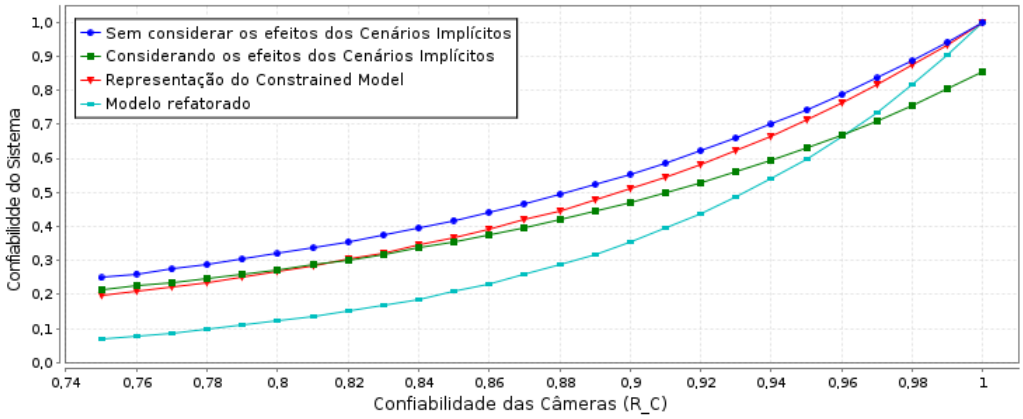


Figura 5.13: Curva comparativa relativa à arquitetura 1

Para a comparação dos resultados, repete-se a execução da análise de sensibilidade para esse modelo.

Análise das ações de refinamento arquitetural

Com as propostas de ações para o refinamento, e as respectivas análises de sensibilidade, podem-se compará-las com as situações originais, apresentadas na análise quantitativa. Seriam quatro situações para cada arquitetura:

- Execução desconsiderando os efeitos dos cenários implícitos negativos;
- Execução considerando os efeitos dos cenários implícitos negativos, levando o sistema a estado de erro quando vierem a ser executados;
- Execução do *Constrained Model* que simula o sistema sem a presença dos cenários implícitos (refinamento dirigido a *constraints*);
- Execução do modelo refatorado que eliminou os cenários implícitos negativos (refinamento dirigido a cenários).

Na Figura 5.13 é apresentado o gráfico comparando as quatro execuções na arquitetura 1.

Já na Figura 5.14 é apresentado o gráfico comparando as quatro execuções na arquitetura 2.

Verifica-se que, em ambas arquiteturas, conforme o esperado, a curva relativa ao *Constrained Model* se aproximou da relativa à situação na qual se desconsidera os cenários implícitos. A curva do *Constrained Model* se acomodou entre as duas curvas originais, evidenciando a melhoria da confiabilidade do sistema. Entretanto, pelo fato da distância

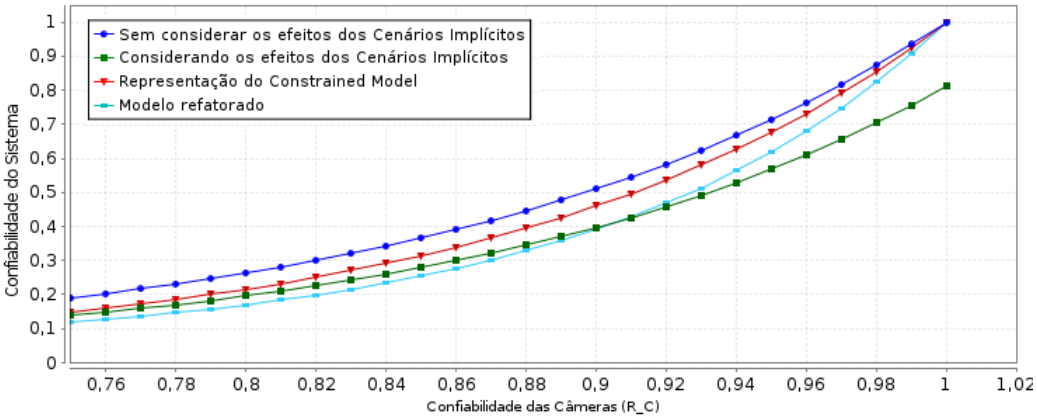


Figura 5.14: Curva comparativa relativa à arquitetura 2

entre as curvas, já destacado na Seção 5.5.4, a aproximação da curva do modelo que evita os cenários implícitos ocorreram em velocidades mais rapidamente na arquitetura 1.

Com relação à curva relacionada ao modelo refatorado, ambas tiveram o mesmo comportamento, mas com diferença considerável entre os valores. Nas duas arquiteturas foi registrado um aumento da sensibilidade da confiabilidade do sistema com relação às confiabilidades das câmeras. Fato evidenciado pela maior inclinação de ambas as curvas com relação às outras três. Na arquitetura 1, o modelo refatorado passa ser interessante, pois melhoraria o grau de confiabilidade com relação ao modelo que não atua sobre os cenários implícitos, somente quando as câmeras possuem valor de confiabilidade acima de 97%. Já na arquitetura 2, o cruzamento das curvas ocorrem quando a confiabilidade das câmeras alcançam por volta de 91% de confiabilidade. Entertanto, em ambas arquiteturas, a opção pela implementação de *constraints* se mostrou mais vantajosa.

5.6 Resultados e conclusões

Neste estudo de caso tinha-se definido três hipóteses a serem validadas: (I) que presença de cenários implícitos afetaria de forma diferenciada os arranjos, (II) que as falhas geradas pelos cenários implícitos afetariam de forma relevante o sistema e, (III) que seria possível melhorar a confiabilidade do modelo tratando os cenários implícitos detectados. Os resultados consolidados são apresentados a seguir:

- Foram encontrados um total de 4 e 52 cenários implícitos para as arquiteturas 1 e 2, respectivamente. Desses, para a arquietura 1 foram todos os quatro negativos. Para a arquitetura 2 foi encontrado apenas um cenário implícito negativo e 51 positivos. Também se diferiram os valores da confiabilidade do modelo. Os valores para a

arquitetura 1 foram sempre superiores aos da arquitetura 2, entretanto, verificou-se um aumento dessa diferença quando os cenários implícitos foram considerados. Fato evidenciado com o cálculo da diferença relativa, que indicou que os valores de confiabilidade da arquitetura 2 apresentaram uma variação proporcionalmente maior que os da arquitetura 1. Uma justificativa para esse resultado seria a alteração na cobertura do campo de visão das câmeras. Quanto maiores forem as zonas de intersecção dos campos de visão, maior será a quantidade de trocas de mensagens entre elas, possibilitando a ocorrência mais frequente dos cenários implícitos que prejudicam a confiabilidade do sistema. Como se pode depreender da Figura 5.2 que mostra os arranjos das câmeras para cada arquitetura, a arquitetura 1 apresenta uma região menor de intersecção que a arquitetura 2, influenciando na quantidade de transações de troca de responsabilidade entre as câmeras. O que justifica a arquitetura 1 ter uma maior confiabilidade e ser menos influenciada pelos cenários implícitos. Cabe destacar que esse resultado ocorreu apesar da arquitetura 1 ter apresentado mais cenários implícitos que a arquitetura 2. Ou seja, realmente os cenários implícitos afetam de forma diferente esse sistema, justificando a necessidade dessa análise em uma fase preliminar à composição de um novo arranjo.

- Na Tabela 5.2 são elencados os defeitos que estão sendo considerados como de possível ocorrência que seriam originados desses cenários implícitos negativos. Dado que esses cenários têm a mesma natureza, os defeitos seriam os mesmos para os dois arranjos. Dentre as classificações em comum, destaca-se o fato de serem inconsistentes, de difícil reprodução e previsibilidade.

- As duas alternativas apresentadas para as ações de refinamento apresentaram resultados diferentes. A opção pela implementação de mecanismos que evitassem os efeitos dos cenários implícitos, representada pela obtenção do *Constrained Model*, resultou na melhoria da confiabilidade nas duas arquiteturas. Os gráficos apresentados nas Figuras 5.13 e 5.14 mostram que as curvas relativas ao *Constrained Model* se aproximam de forma semelhante às curvas que não consideram o efeito dos cenários implícitos nas duas arquiteturas. Já a opção pela alteração do modelo com o objetivo de eliminar a presença dos cenários implícitos negativos levou a resultados diferentes. Há que se levar em conta que a opção tomada para a eliminação dos cenários foi a inclusão de novas mensagens, fato que incorre no aumento do número de transações entre os componentes elevando a importância do do valor da confiabilidade dos componentes para a confiabilidade global do sistema. Dessa forma, o gráfico da arquitetura 1 indica que a opção pelo refatoramento dos cenários só passa a ser interessante quando a confiabilidade das câmeras passa os 97%. Caso o valor

for abaixo desse, os valores de confiabilidade do modelo original, no qual os cenários implícitos levam o sistema ao erro, são mais vantajosos. Já para a arquitetura 2, esse valor seria menor, cerca de 91%. Entretanto, em ambas arquiteturas, o mais vantajoso seria a implementação do refinamento dirigido a *constraints*. Dessa forma, fica validado que a correção dos cenários implícitos negativos resulta em modelos com possibilidade de alcançar níveis melhores de confiabilidade, dependendo, para isso, sobretudo no caso da arquitetura 1, do aumento da confiabilidade das câmeras.

Portanto, tendo em vista os resultados listados, com a modelagem original, a arquitetura 1 se mostrou ser a mais adequada para ser a escolhida sob o ponto de vista da confiabilidade. Caso haja o refatoramento do modelo em prol da eliminação dos cenários implícitos negativos, a arquitetura 2 passaria a ser a mais adequada. Entretanto, outros critérios, como cobertura de visão, poderiam indicar outras escolhas. Mas, há que se colocar que em sistemas críticos o quesito de confiabilidade é crucial, o que justificaria a tomada de decisão levando em conta esse fator.

5.7 Ameaças à validade

Diante do apresentado nas seções anteriores, destacam-se três ameaças que poderiam invalidar ou prejudicar a validade dos resultados alcançados.

- A quantidade de cenários utilizados na modelagem é suficiente para estabelecer um resultado confiável? Realmente, não existiria uma resposta definitiva para essa questão. A forma como se deu a modelagem, levantando cenários de uso aleatórios, abre esse precedente. Entretanto, cabe destacar que a utilização de mais cenários implicaria, necessariamente, no aumento de estados a serem tratados na modelagem, podendo alcançar uma explosão de estados. Inclusive, essa limitação das modelagens formais é abordada em [23], ao argumentar que a modelagem de um sistema no devido grau de abstração é uma tarefa complicada, devido ao risco de se tratar detalhes e incorrerem uma explosão de estados ou utilizar um nível de abstração alto que prejudique a validade dos resultados. Existem trabalhos que tratam esse problema por meio da implementação de heurísticas.

- Com mais cenários na modelagem poderiam ocorrer outros cenários implícitos? Provavelmente sim. Mas esses novos cenários poderiam atuar no sentido de propiciar uma explosão de estados, conforme destacado no item anterior.

- Esses cenários implícitos podem ser reproduzidos no sistema? Devido ao fato de serem inconsistentes, não seria possível prever com exatidão quando eles ocorreriam, mas é certo que eles teriam determinada probabilidade de ocorrência.

Capítulo 6

Conclusões e trabalhos futuros

O trabalho de pesquisa buscou apresentar uma nova metodologia para a análise do impacto da presença de cenários implícitos sobre a confiabilidade de um modelo. Essa análise se dá tanto de forma quantitativa quanto qualitativa, com o intuito de apresentar um diagnóstico consistente. A metodologia foi apresentada utilizando como exemplo um sistema didático de gerência de caldeira que por já ter sido explorado em outros trabalhos, possibilitou a comparação dos resultados.

Foi realizado um estudo de caso empregando a metodologia no projeto de um sistema de câmeras inteligentes de forma a chegar a um diagnóstico sobre qual arranjo de câmeras seria o mais confiável. Além disso, conforme a metodologia prevê, foram identificadas oportunidades de refinamento arquitetural a serem implementadas, a decisão sobre qual alternativa seguir pode ser justificada pela análise de sensibilidade.

Verifica-se com os resultados a importância de se considerar tanto o efeito quantitativo como o qualitativo da presença dos cenários implícitos. Fica clara a possibilidade da existência de um cenário implícito que tenha um alto impacto quantitativo sobre a confiabilidade do sistema, entretanto não tenha um impacto crítico, qualitativo. Assim como o contrário, um cenário implícito que afete pouco quantitativamente, mas a falha que ocasionaria é altamente impactante.

6.1 Trabalhos futuros

Tendo em vista que, conforme apresentado [46], existem outros fenômenos, além dos cenários implícitos, que podem impactar a confiabilidade de um modelo baseado em cenários, como: condições de corrida [29], escolha não local(tradução para *non-local choice*) [47], confluência [47], dentre outros. Uma possibilidade de trabalho futuro seria a integração à metodologia da análise dessas outras anomalias, chamadas de patologias

em [45]. Essa medida agregaria mais valor à metodologia no sentido de obter um modelo com maior nível de confiabilidade.

Uma outra proposta de trabalho seria a realização de um estudo de caso em um sistema que possibilitasse a implementação das medidas de refinamento arquitetural eventualmente propostas, juntamente com a aferição dos resultados. Esse trabalho possibilitaria uma validação objetiva dos ganhos alcançados com a aplicação da metodologia.

Referências Bibliográficas

[1] S. Uchitel, J. Kramer, and J. Magee, *Synthesis of Behavioral Models from Scenarios,* in IEEE Trans. Softw. Eng., vol. 29, no. 2, pp. 99–115, Feb. 2003. ix, 1, 18, 22, 27, 29

[2] Object Management Group. *UML 2.0 Superstructure.* In http://www.omg.org/cgi-bin/doc?ptc/2004-10-02, 2004. 1, 15

[3] ITU-T Recommendation Z.120 Message Sequence Charts (MSC'99). *Technical report,* In ITU Telecommunication Standardization Sector, Geneva, 1996. 1, 15, 17

[4] Sherif M. Yacoub, Bojan Cukic, and Hany H. Ammar. *Scenario-Based Reliability Analysis of Component-Based Software.* In Proc. of the 10th ISSRE, Boca Raton, FL, USA. IEEE, November 1999. 1

[5] Vittorio Cortellessa and Antonio Pompei. *Towards a uml profile for qos: a contribution in the reliability domain.* In Jozo J. Dujmovic, Virgílio A. F. Almeida, and Doug Lea, editors, Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004, Red- wood Shores, California, USA, January 14-16, pages 197–206. ACM, 2004. 1

[6] Carlo Ghezzi and Amir Sharifloo. *Quantitative verification of non-functional requirements with uncertainty.* In Dependable Computer Systems, pages 47–62. Springer Berlin / Heidelberg, 2011. 1

[7] R. Alur, K. Etessami, and M. Yannakakis, *Inference of Message Sequence Charts*, In IEEE Transactions on Software Engineering, vol. 29, no. 7, pp. 623–633, Jul. 2003. 1, 15, 22

[8] Uchitel, S., Kramer, J., and Magee, J. *Detecting implied scenarios in message sequence chart specifications.* In ACM Proceedings of the joint 8th ESEC and 9th FSE, pages 74–82. ACM Press.(2001). xii, 1, 2, 4, 15, 24, 27, 29, 33, 41, 42

[9] Hoare, C. A. R. *Communicating sequential processes.* In Communications of the ACM, 21(8):666–677, (1978). 1, 12, 19

[10] Rodrigues, G. N., Rosenblum, D. and Uchitel, S. *Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems.* In Proc. ETAPS 2005 Conference on Formal Approaches to Software Engineering, pages 111-126. Springer, LNCS 3442. (2005) 3, 4, 29

[11] Rodrigues, G., Alves, V., Silveira, R. Cheungand Laranjeira, L. *Dependability analysis in the ambient assisted living do- main: An exploratory case study.* In Journal of Systems and Software, 85(1):112-131, (2012). 3, 4, 10

[12] S. Al-Azzani and R. Bahsoon, *Semi-automated Detection of Architectural Threats for Security Testing,* in Proceedings of the Doctoral Symposium for ESEC/FSE on Doctoral Symposium, New York, NY, USA, 2009, pp. 25–26. 4

[13] S. Al-Azzani and R. Bahsoon, *Using Implied Scenarios in Security Testing,* in Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems, New York, NY, USA, 2010, pp. 15–21. 4

[14] S. Al-Azzani and R. Bahsoon, *SecArch: Architecture-level Evaluation and Testing for Security*, in 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012, pp. 51–60. 4

[15] F. Cantal de Sousa, N. C. Mendonca, S. Uchitel, and J. Kramer, *Detecting Implied Scenarios from Execution Traces*, in 14th Working Conference on Reverse Engineering, 2007. WCRE 2007, 2007, pp. 50–59. 4, 29

[16] A. Avizienis, J. Laprie, and B. Randell. *"Fundamental Concepts of Dependability."* In IARP/IEEE-RAS Workshop on Robot Dependability, May 2001. 6

[17] J.-C. Laprie. *"Dependable Computing and Fault Tolerance: Concepts and Terminology,"* In Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years, 1995. 6

[18] Avizienis, A., Laprie, J., R, B., L, C., and Member, S. *Basic concepts and taxonomy of dependable and secure computing*, In IEEE Transactions on Dependable and Secure Computing, 1:11–33. (2004). 6, 7, 13, 27, 34, 57

[19] Weber, T. S., *"Tolerância a falhas: conceitos e exemplos."* Intech Brasil, São Paulo, vol. 52, pp. 32-42, 2003. 7

[20] A. Avizienis and J.-C. Laprie, *"Dependable computing: From concepts to design diversity,"* Proceedings of the IEEE, vol. 74, no. 5, pp. 629–638, May 1986. 7

[21] S. Bernardi, J. Merseguer, D. C. Petriu, *"Model-Driven Dependability Assessment of Software Systems"* In Springer-Verlag, 2013, ISBN 978-3-642-39512-3 (e-book)/ ISBN 978-3-642-39511-6 (hardcover). 8

[22] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design.* Prentice Hall, 1991. 8

[23] Baier, C., Kaoten, JP.*Principles of Model Checking.* In The MIT Press, Cambridge, Massachusetts (2008). xii, 8, 9, 10, 66

[24] H. Hansson and B. Jonsson, *"A Logic for Reasoning about Time and Reliability,"* In Formal Aspects of Computing, vol. 6, pp. 102–111, 1994. 8, 12

[25] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM 4.0: Verification of probabilistic real-time systems,* in CAV'11, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, pp. 585–591. 8

[26] O.U.C. Laboratory, *Prism case studies* In http://www.prismmodelchecker.org/casestudies/. 10

[27] E. M. Clarke, E. A. Emerson, and A. P. Sistla, *"Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications,"* In ACM Trans. Program. Lang. Syst., vol. 8, no. 2, pp. 244–263, Apr. 1986. 12

[28] Carroll, J.M. (ed.) *"Scenario-based design: envisioning work and technology in system development"*. Wiley, New York, 1995. 15

[29] R. Alur, G. J. Holzmann, and D. Peled, *"An analyzer for message sequence charts,"* In Tools and Algorithms for the Construction and Analysis of Systems, T. Margaria and B. Steffen, Eds. Springer Berlin Heidelberg, 1996, pp. 35–48. 15, 67

[30] Rodrigues, G. N. *A Model Driven Approach for Software Reliability Prediction.* PhD thesis, University College London. (2008). xiii, xv, 15, 30, 40

[31] S. Uchitel, R. Chatley,J. Kramer, and J. Magee, *"LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios."* In Proc. of the 9thvInternational Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03), LNCS vol. 2619, Springer. 2, 4, 17, 21

[32] E. M. Clarke and J. M. Wing, *"Formal Methods: State of the Art and Future Directions,"* In ACM Comput. Surv., vol. 28, no. 4, pp. 626–643, Dezembro 1996. 18

[33] Magee, J., Kramer, J. and Giannakopoulou, D. *Software architecture directed behaviour analysis.* In Proceedings of the 9th International Workshop on Software Specification and Design, Washington, DC, USA. IEEE Computer Society. (1998) 18, 21

[34] *LTSA - Labelled Transition System Analyser* In http://www.doc.ic.ac.uk/ltsa/ . 21

[35] S. Uchitel, J. Kramer, and J. Magee, *Incremental elaboration of scenario-based specifications and behavior models using implied scenarios.* ACM Trans. Softw. Eng. Methodol., 13(1):37–85.(2004). 27

[36] S. Uchitel, J. Kramer, and J. Magee, *Negative Scenarios for Implied Scenario Elicitation,* in Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering, New York, NY, USA, 2002, pp. 109–118. 23, 27

[37] Goldsmith, M. and Whittaker, P. *A csp frontend for probabilistic tools.* Technical Report FORWARD Deliverable D14, Formal Systems.(2005). 36

[38] S. Al-Azzani, *"Architecture-Centric Testing for Security"* PhD thesis, University of Birmingham, 2013. xiii, 50, 51, 52

[39] Magee, J. and Kramer, J. *Concurrency: State Models and Java Programs.* John Wiley, New York. (1999) 21

[40] Esterle, L., Lewis, P. R., *et al.,A socio-economic approach to online vision graph generation and handover in distributed smart camera networks.* In Proceedings of the Fifth ACM/IEEE International Conference on Distributed Smart Cameras, pages 1–6.(2011) 46, 49

[41] Esterle, L., Lewis, P. R., *et al.,Socio-Economic Vision Graph Generation and Handover in Distributed Smart Camera Networks.* In Proceedings of the ACM Transactions on Sensor Networks, vol. 10, no. 2, pp. 20:1–20:24, Jan. 2014. 46, 47

[42] R. M. Keller, *Formal Verification of Parallel Programs,* in Commun. ACM, vol. 19, no. 7, pp. 371–384, Jul. 1976. 18

[43] D. E. Perry, A. A. Porter, and L. G. Votta, *Empirical Studies of Software Engineering: A Roadmap,* in Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA, 2000, pp. 345–355. 45

[44] Roriz A., Rodrigues G., Laranjeira L. *Analysis of the Impact of Implied Scenarios on the Reliability of Computational Concurrent Systems* In Proc. of 8th Brazilian

Symposium on Software Components, Architectures and Reuse, Maceió, AL, BR. Pages 94-103. October 2014. 27

[45] P. Baker, P. Bristow, C. Jervis, D. King, R. Thomson, B. Mitchell, and S. Burton, *"Detecting and Resolving Semantic Pathologies in UML Sequence Diagrams,"* In Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, New York, NY, USA, 2005, pp. 50–59. 68

[46] H. Dan, R. M. Hierons, and S. Counsell, *"Non-local Choice and Implied Scenarios,"* In 2010 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM), 2010, pp. 53–62. 67

[47] P. Ladkin and S. Leue, *"What Do Message Sequence Charts Mean?"* In Proceedings of the IFIP TC6/WG6.1 Sixth International Conference on Formal Description Techniques, pages 301–316, Boston, USA, 1993. 18, 67

[48] S. Leue and P. B. Ladkin, *"Implementing and Verifying Scenario-Based Specifications Using Promela/XSpin (Extended Abstract),"* In PROCEEDINGS OF THE 2ND WORKSHOP ON THE SPIN VERIFICATION SYSTEM, RUTGERS UNIVERSITY, AUGUST 5, 1996. 18

[49] H. Ben-Abdallah and S. Leue, *"Syntactic analysis of Message Sequence Chart specifications."* In Tech Report 96-12, Department of Electrical and Computer Engineering, University of Waterloo, November 1996. 18

Anexo A

Diagramas bMSC referentes aos cenários modelados

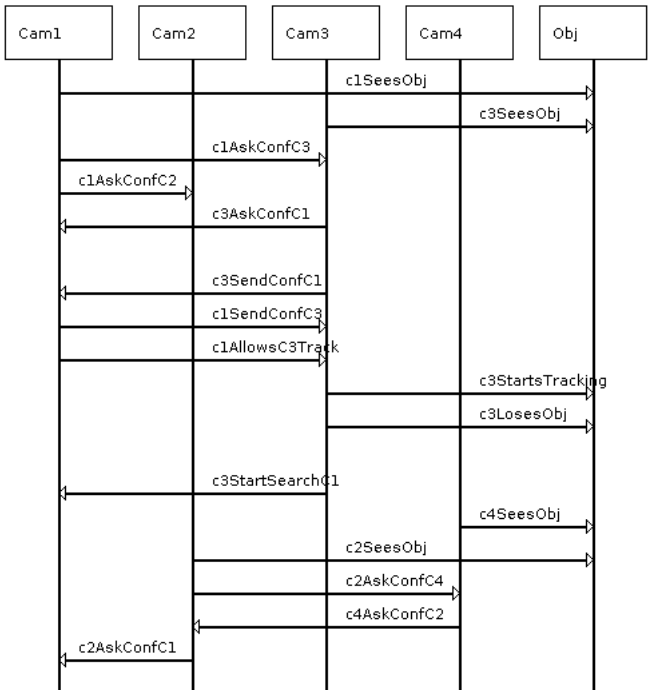


Figura A.1: Diagrama bMSC referente ao cenário 1 da Arquitetura 1

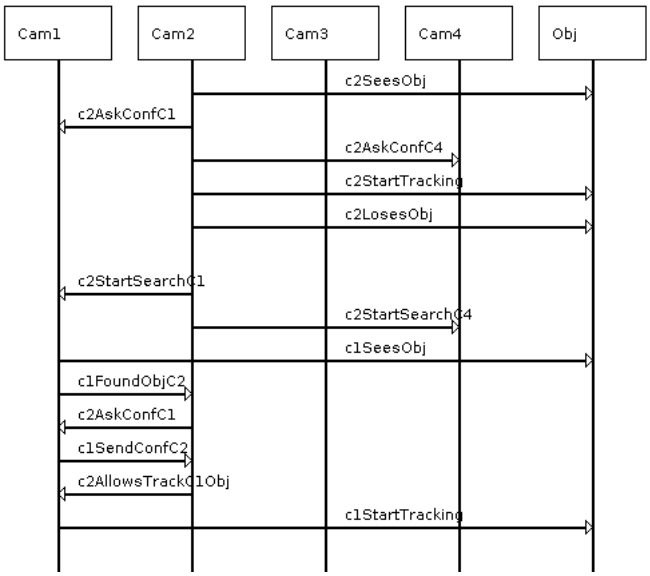


Figura A.2: Diagrama bMSC referente ao cenário 2 da Arquitetura 1

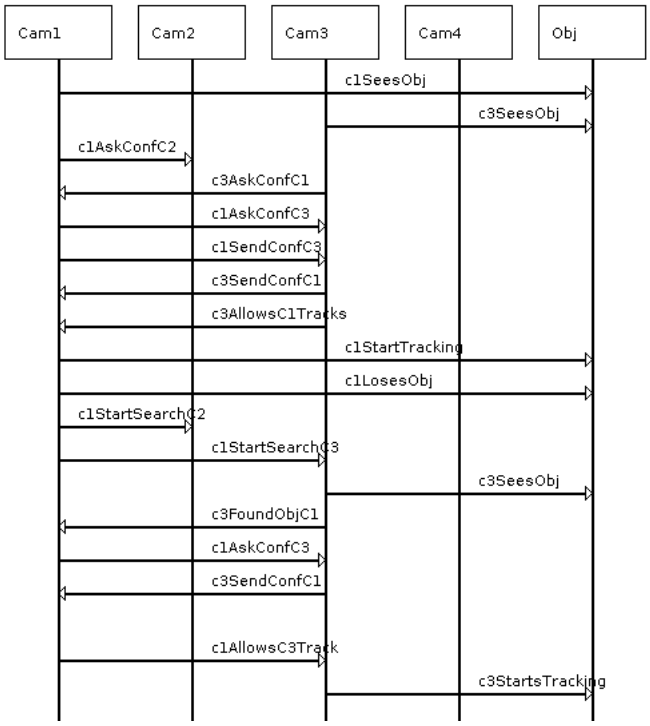


Figura A.3: Diagrama bMSC referente ao cenário 3 da Arquitetura 1

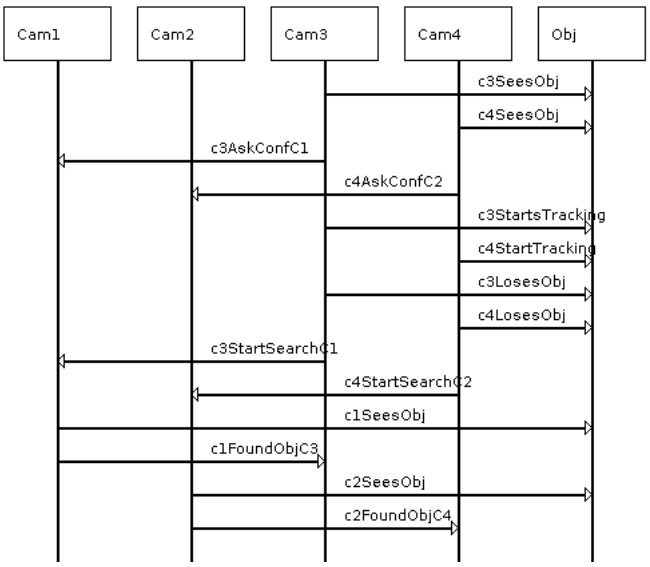


Figura A.4: Diagrama bMSC referente ao cenário 4 da Arquitetura 1

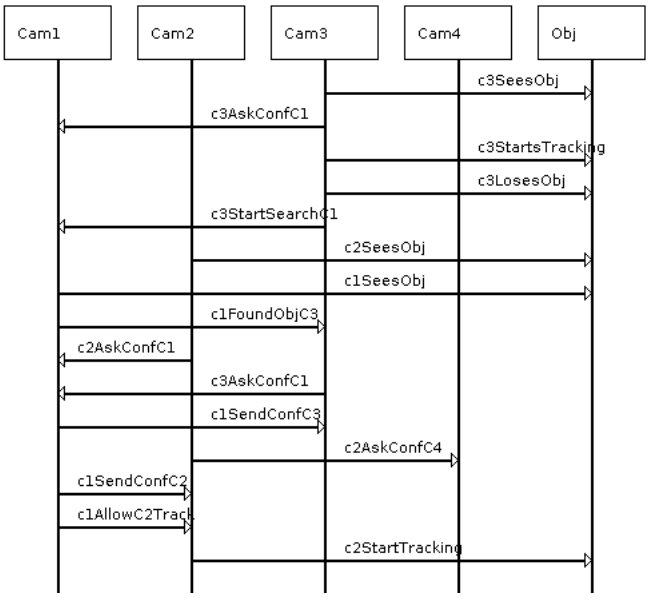


Figura A.5: Diagrama bMSC referente ao cenário 5 da Arquitetura 1

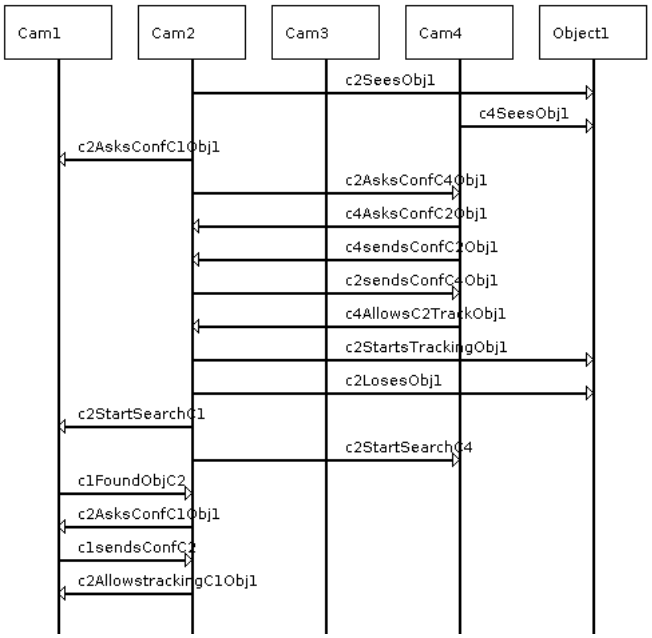


Figura A.6: Diagrama bMSC referente ao cenário 1 da Arquitetura 2

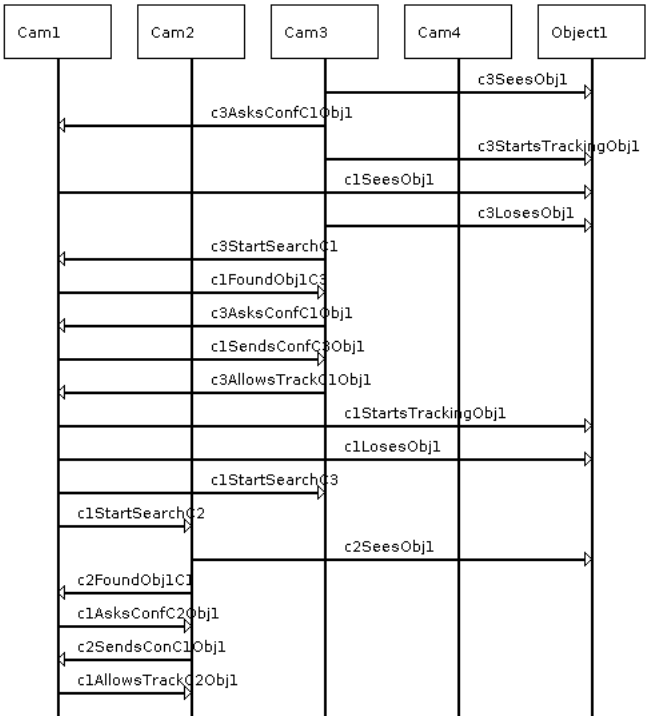


Figura A.7: Diagrama bMSC referente ao cenário 2 da Arquitetura 2

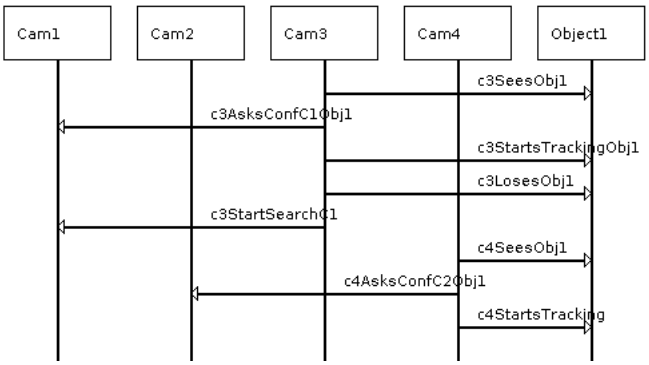


Figura A.8: Diagrama bMSC referente ao cenário 3 da Arquitetura 2

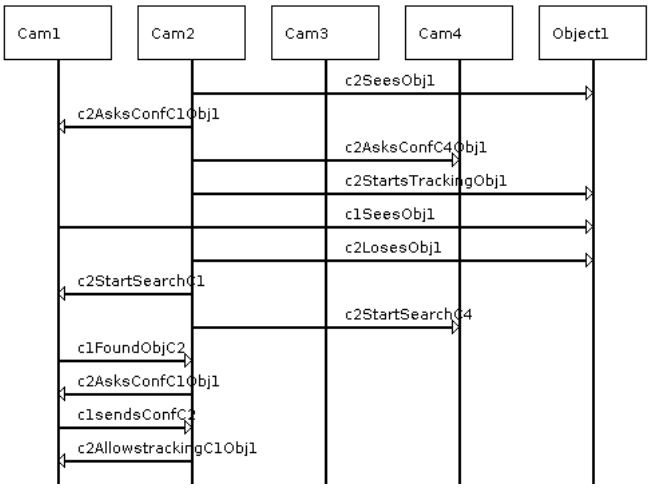


Figura A.9: Diagrama bMSC referente ao cenário 4 da Arquitetura 2

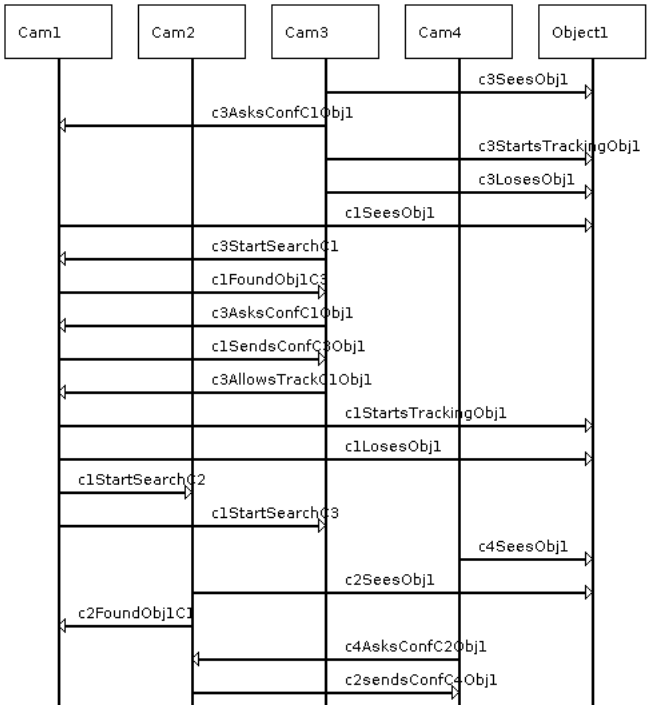


Figura A.10: Diagrama bMSC referente ao cenário 5 da Arquitetura 2

Anexo B

Modelos comportamentais expressos em FSP

B.1 Modelo comportamental da Arquitetura 1 expresso em FSP

```
1  ArchitectureModel = Q0,
2   Q0 = (c1SeesObj -> Q1 | c2SeesObj -> Q44 |c3SeesObj -> Q57),
3   Q1 = (c1AskConfC2 -> Q2 |c3SeesObj -> Q26),
4   Q2 = (c3SeesObj -> Q3),
5   Q3 = (c3AskConfC1 -> Q4),
6   Q4 = (c1AskConfC3 -> Q5 |c3StartsTracking -> Q20),
7   Q5 = (c1SendConfC3 -> Q6),
8   Q6 = (c3SendConfC1 -> Q7),
9   Q7 = (c3AllowsC1Tracks -> Q8),
10  Q8 = (c1StartTracking -> Q9),
11  Q9 = (c1LosesObj -> Q10),
12  Q10 = (c1StartSearchC2 -> Q11),
13  Q11 = (c1StartSearchC3 -> Q12),
14  Q12 = (c3SeesObj -> Q13),
15  Q13 = (c3FoundObjC1 -> Q14),
16  Q14 = (c1AskConfC3 -> Q15),
17  Q15 = (c3SendConfC1 -> Q16),
18  Q16 = (c1AllowsC3Track -> Q17),
19  Q17 = (c3StartsTracking -> Q18),
20  Q18 = (endAction -> Q19),
21  Q19 = END,
22  Q20 = (c3LosesObj -> Q21),
23  Q21 = (c4SeesObj -> Q22),
24  Q22 = (c2SeesObj -> Q23),
25  Q23 = (c2AskConfC4 -> Q24),
26  Q24 = (c4AskConfC2 -> Q25),
27  Q25 = STOP,
28  Q26 = (c1AskConfC2 -> Q3 |c1AskConfC3 -> Q27),
29  Q27 = (c1AskConfC2 -> Q28),
30  Q28 = (c3AskConfC1 -> Q29),
31  Q29 = (c3SendConfC1 -> Q30),
```

```
Q30 = (c1SendConfC3 -> Q31),
Q31 = (c1AllowsC3Track -> Q32),
Q32 = (c3StartsTracking -> Q33),
Q33 = (c3LosesObj -> Q34),
Q34 = (c3StartSearchC1 -> Q35 |c4SeesObj -> Q40),
Q35 = (c4SeesObj -> Q36),
Q36 = (c2SeesObj -> Q37),
Q37 = (c2AskConfC4 -> Q38),
Q38 = (c4AskConfC2 -> Q39),
Q39 = (c2AskConfC1 -> Q18),
Q40 = (c3StartSearchC1 -> Q36 |c2SeesObj -> Q41),
Q41 = (c3StartSearchC1 -> Q37 |c2AskConfC4 -> Q42),
Q42 = (c3StartSearchC1 -> Q38 |c4AskConfC2 -> Q43),
Q43 = (c3StartSearchC1 -> Q39),
Q44 = (c2AskConfC1 -> Q45),
Q45 = (c2AskConfC4 -> Q46),
Q46 = (c2StartTracking -> Q47),
Q47 = (c2LosesObj -> Q48),
Q48 = (c2StartSearchC1 -> Q49),
Q49 = (c2StartSearchC4 -> Q50 |c1SeesObj -> Q56),
Q50 = (c1SeesObj -> Q51),
Q51 = (c1FoundObjC2 -> Q52),
Q52 = (c2AskConfC1 -> Q53),
Q53 = (c1SendConfC2 -> Q54),
Q54 = (c2AllowsTrackC1Obj -> Q55),
Q55 = (c1StartTracking -> Q18),
Q56 = (c2StartSearchC4 -> Q51),
Q57 = (c3AskConfC1 -> Q58 |c4SeesObj -> Q94),
Q58 = (c4SeesObj -> Q59 |c3StartsTracking -> Q77),
Q59 = (c4AskConfC2 -> Q60 |c3StartsTracking -> Q76),
Q60 = (c3StartsTracking -> Q61),
Q61 = (c4StartTracking -> Q62),
Q62 = (c3LosesObj -> Q63),
Q63 = (c3StartSearchC1 -> Q64 |c4LosesObj -> Q74),
Q64 = (c4LosesObj -> Q65),
Q65 = (c4StartSearchC2 -> Q66 |c1SeesObj -> Q72),
Q66 = (c1SeesObj -> Q67),
Q67 = (c1FoundObjC3 -> Q68 |c2SeesObj -> Q70),
Q68 = (c2SeesObj -> Q69),
Q69 = (c2FoundObjC4 -> Q18),
Q70 = (c1FoundObjC3 -> Q69 |c2FoundObjC4 -> Q71),
Q71 = (c1FoundObjC3 -> Q18),
Q72 = (c4StartSearchC2 -> Q67 |c1FoundObjC3 -> Q73),
Q73 = (c4StartSearchC2 -> Q68),
Q74 = (c3StartSearchC1 -> Q65 |c4StartSearchC2 -> Q75),
Q75 = (c3StartSearchC1 -> Q66),
Q76 = (c4AskConfC2 -> Q61),
Q77 = (c3LosesObj -> Q78),
Q78 = (c3StartSearchC1 -> Q79 |c2SeesObj -> Q93),
Q79 = (c2SeesObj -> Q80),
Q80 = (c1SeesObj -> Q81),
Q81 = (c1FoundObjC3 -> Q82),
Q82 = (c2AskConfC1 -> Q83),
Q83 = (c3AskConfC1 -> Q84 |c2AskConfC4 -> Q91),
Q84 = (c1SendConfC3 -> Q85 |c2AskConfC4 -> Q89),
Q85 = (c2AskConfC4 -> Q86),
Q86 = (c2StartTracking -> Q25 |c1SendConfC2 -> Q87),
```

```
89  Q87 = (c1AllowC2Track -> Q88),
90  Q88 = (c2StartTracking -> Q18),
91  Q89 = (c1SendConfC3 -> Q86 |c2StartTracking -> Q90),
92  Q90 = (c1SendConfC3 -> Q25),
93  Q91 = (c3AskConfC1 -> Q89 |c2StartTracking -> Q92),
94  Q92 = (c3AskConfC1 -> Q90),
95  Q93 = (c3StartSearchC1 -> Q80),
96  Q94 = (c3AskConfC1 -> Q59 |c4AskConfC2 -> Q95),
97  Q95 = (c3AskConfC1 -> Q60).
```

Listagem B.1: Modelo comportamental da Arquitetura 1 expresso em FSP

B.2 Modelo comportamental da Arquitetura 2 expresso em FSP

```
1  ArchitectureModel = Q0,
2  Q0 = (c2SeesObj1 -> Q1 |c3SeesObj1 -> Q24),
3  Q1 = (c2AsksConfC1Obj1 -> Q2 |c4SeesObj1 -> Q23),
4  Q2 = (c2AsksConfC4Obj1 -> Q3 |c4SeesObj1 -> Q14),
5  Q3 = (c2StartsTrackingObj1 -> Q4),
6  Q4 = (c1seesObj1 -> Q5),
7  Q5 = (c2LosesObj1 -> Q6),
8  Q6 = (c2StartSearchC1 -> Q7),
9  Q7 = (c2StartSearchC4 -> Q8),
10 Q8 = (c1FoundObjC2 -> Q9),
11 Q9 = (c2AsksConfC1Obj1 -> Q10),
12 Q10 = (c1sendsConfC2 -> Q11),
13 Q11 = (c2AllowstrackingC1Obj1 -> Q12),
14 Q12 = (endAction -> Q13),
15 Q13 = END,
16 Q14 = (c2AsksConfC4Obj1 -> Q15),
17 Q15 = (c4AsksConfC2Obj1 -> Q16 |c2StartsTrackingObj1 -> Q20),
18 Q16 = (c4sendsConfC2Obj1 -> Q17),
19 Q17 = (c2sendsConfC4Obj1 -> Q18),
20 Q18 = (c4AllowsC2TrackObj1 -> Q19),
21 Q19 = (c2StartsTrackingObj1 -> Q5),
22 Q20 = (c2LosesObj1 -> Q21),
23 Q21 = (c2StartSearchC1 -> Q22),
24 Q22 = STOP,
25 Q23 = (c2AsksConfC1Obj1 -> Q14),
26 Q24 = (c3AsksConfC1Obj1 -> Q25),
27 Q25 = (c3StartsTrackingObj1 -> Q26),
28 Q26 = (c1SeesObj1 -> Q27 |c3LosesObj1 -> Q45),
29 Q27 = (c3LosesObj1 -> Q28),
30 Q28 = (c3StartSearchC1 -> Q29),
31 Q29 = (c1FoundObj1C3 -> Q30),
32 Q30 = (c3AsksConfC1Obj1 -> Q31),
33 Q31 = (c1SendsConfC3Obj1 -> Q32),
34 Q32 = (c3AllowsTrackC1Obj1 -> Q33),
35 Q33 = (c1StartsTrackingObj1 -> Q34),
36 Q34 = (c1LosesObj1 -> Q35),
37 Q35 = (endAction -> Q13 |c1StartSearchC3 -> Q36 |c1StartSearchC2 -> Q42
38  |c2SeesObj1 -> Q44),
39 Q36 = (c2SeesObj1 -> Q12 |endAction -> Q13 |c1StartSearchC2 -> Q37),
```

```
40 Q37 = (endAction -> Q13 |c2SeesObj1 -> Q38),
41 Q38 = (endAction -> Q13 |c2FoundObj1C1 -> Q39),
42 Q39 = (endAction -> Q13 |c1AsksConfC2Obj1 -> Q40),
43 Q40 = (endAction -> Q13 |c2SendsConC1Obj1 -> Q41),
44 Q41 = (c1AllowsTrackC2Obj1 -> Q12 |endAction -> Q13),
45 Q42 = (endAction -> Q13 |c1StartSearchC3 -> Q37 |c2SeesObj1 -> Q43),
46 Q43 = (endAction -> Q13 |c1StartSearchC3 -> Q38),
47 Q44 = (c1StartSearchC3 -> Q12 |endAction -> Q13),
48 Q45 = (c3startSearchC1 -> Q46 |c4SeesObj1 -> Q49 |c1SeesObj1 -> Q52),
49 Q46 = (c4SeesObj1 -> Q47),
50 Q47 = (c4AsksConfC2Obj1 -> Q48),
51 Q48 = (cam4.object1.c4StartsTracking -> Q12),
52 Q49 = (c3startSearchC1 -> Q47 |c4AsksConfC2Obj1 -> Q50),
53 Q50 = (c3startSearchC1 -> Q48 |cam4.object1.c4StartsTracking -> Q51),
54 Q51 = (c3startSearchC1 -> Q12),
55 Q52 = (c3StartSearchC1 -> Q53),
56 Q53 = (c1FoundObj1C3 -> Q54),
57 Q54 = (c3AsksConfC1Obj1 -> Q55),
58 Q55 = (c1SendsConfC3Obj1 -> Q56),
59 Q56 = (c3AllowsTrackC1Obj1 -> Q57),
60 Q57 = (c1StartsTrackingObj1 -> Q58),
61 Q58 = (c1LosesObj1 -> Q59),
62 Q59 = (endAction -> Q13 |c1StartSearchC3 -> Q60 |c1StartSearchC2 -> Q67
63 |c4SeesObj1 -> Q70),
64 Q60 = (endAction -> Q13 |c1StartSearchC2 -> Q61 |c4SeesObj1 -> Q66),
65 Q61 = (endAction -> Q13 |c4SeesObj1 -> Q62),
66 Q62 = (endAction -> Q13 |c2SeesObj1 -> Q63),
67 Q63 = (endAction -> Q13 |c2FoundObj1C1 -> Q64),
68 Q64 = (endAction -> Q13 |c1AsksConfC2Obj1 -> Q40 |c4AsksConfC2Obj1 -> Q65),
69 Q65 = (endAction -> Q13 |c2sendsConfC4Obj1 -> Q39),
70 Q66 = ({c2SeesObj1, c4AsksConfC2Obj1} -> Q12 |endAction -> Q13 |c1StartSearchC2 -> Q62),
71 Q67 = (endAction -> Q13 |c1StartSearchC3 -> Q61 |c4SeesObj1 -> Q68),
72 Q68 = (endAction -> Q13 |c1StartSearchC3 -> Q62 |c2SeesObj1 -> Q69),
73 Q69 = (endAction -> Q13 |c1StartSearchC3 -> Q63),
74 Q70 = (endAction -> Q13 |{c2SeesObj1, c4AsksConfC2Obj1} -> Q44 |c1StartSearchC3 -> Q66
75 |c1StartSearchC2 -> Q68).
```

Listagem B.2: Modelo comportamental da Arquitetura 2 expresso em FSP

Anexo C

Modelagem na linguagem do PRISM

C.1 Modelagem na linguagem do PRISM para a Arquitetura 1

```
1
2 dtmc
3
4 const double R_C;
5 const int FINAL_STATE = 19;
6 const int ERROR_STATE = 96;
7
8 module ArchitectureModel
9
10 sys_st : [0..96] init 0;
11 [c1SeesObj] sys_st = 0 -> (sys_st'=1);
12 [c2SeesObj] sys_st = 0 -> (sys_st'=44);
13 [c3SeesObj] sys_st = 0 -> (sys_st'=57);
14 [c1AskConfC2] sys_st = 1 -> R_C:(sys_st'=2) + (1-R_C):(sys_st'=ERROR_STATE);
15 [c3SeesObj] sys_st = 1 -> (sys_st'=26);
16 [c3SeesObj] sys_st = 2 -> (sys_st'=3);
17 [c3AskConfC1] sys_st = 3 -> R_C:(sys_st'=4) + (1-R_C):(sys_st'=ERROR_STATE);
18 [c1AskConfC3] sys_st = 4 -> R_C:(sys_st'=5) + (1-R_C):(sys_st'=ERROR_STATE);
19 [c3StartTracking] sys_st = 4 -> (sys_st'=20); //Trace com cenario implicito
20 [c1SendConfC3] sys_st = 5 -> R_C:(sys_st'=6) + (1-R_C):(sys_st'=ERROR_STATE);
21 [c3SendConfC1] sys_st = 6 -> R_C:(sys_st'=7) + (1-R_C):(sys_st'=ERROR_STATE);
22 [c3AllowC1Track] sys_st = 7 -> R_C:(sys_st'=8) + (1-R_C):(sys_st'=ERROR_STATE);
23 [c1StartTracking] sys_st = 8 -> (sys_st'=9);
24 [c1LosesObj] sys_st = 9 -> (sys_st'=10);
25 [c1StartSearchC2] sys_st = 10 -> R_C:(sys_st'=11) + (1-R_C):(sys_st'=ERROR_STATE);
26 [c1StartSearchC3] sys_st = 11 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
27 [c3SeesObj] sys_st = 12 -> (sys_st'=13);
28 [c3FoundObjC1] sys_st = 13 -> R_C:(sys_st'=14) + (1-R_C):(sys_st'=ERROR_STATE);
29 [c1AskConfC3] sys_st = 14 -> R_C:(sys_st'=15) + (1-R_C):(sys_st'=ERROR_STATE);
30 [c3SendConfC1] sys_st = 15 -> R_C:(sys_st'=16) + (1-R_C):(sys_st'=ERROR_STATE);
31 [c1AllowC3Track] sys_st = 16 -> R_C:(sys_st'=17) + (1-R_C):(sys_st'=ERROR_STATE);
32 [c3StartTracking] sys_st = 17 -> (sys_st'=11);
33 [end] sys_st = 18 -> (sys_st'=FINAL_STATE);
34 [c3LosesObj] sys_st = 20 -> (sys_st'=21);
```

```
35 [c4SeesObj] sys_st = 21 -> (sys_st'=22);
36 [c2SeesObj] sys_st = 22 -> (sys_st'=23);
37 [c2AskConfC4] sys_st = 23 -> R_C:(sys_st'=24) + (1-R_C):(sys_st'=ERROR_STATE);
38 [c4AskConfC2] sys_st = 24 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
39 [c1AskConfC3] sys_st = 26 -> R_C:(sys_st'=27) + (1-R_C):(sys_st'=ERROR_STATE);
40 [c1AskConfC2] sys_st = 26 -> R_C:(sys_st'=3) + (1-R_C):(sys_st'=ERROR_STATE);
41 [c1AskConfC2] sys_st = 27 -> R_C:(sys_st'=28) + (1-R_C):(sys_st'=ERROR_STATE);
42 [c3AskConfC1] sys_st = 28 -> R_C:(sys_st'=29) + (1-R_C):(sys_st'=ERROR_STATE);
43 [c3SendConfC1] sys_st = 29 -> R_C:(sys_st'=30) + (1-R_C):(sys_st'=ERROR_STATE);
44 [c1SendConfC3] sys_st = 30 -> R_C:(sys_st'=31) + (1-R_C):(sys_st'=ERROR_STATE);
45 [c1AllowC3Track] sys_st = 31 -> R_C:(sys_st'=32) + (1-R_C):(sys_st'=ERROR_STATE);
46 [c3StartTracking] sys_st = 32 -> (sys_st'=33);
47 [c3LosesObj] sys_st = 33 -> (sys_st'=34);
48 [c3StartSearchC1] sys_st = 34 -> R_C:(sys_st'=35) + (1-R_C):(sys_st'=ERROR_STATE);
49 [c4SeesObj] sys_st = 34 -> (sys_st'=40);
50 [c4SeesObj] sys_st = 35 -> (sys_st'=36);
51 [c2SeesObj] sys_st = 36 -> (sys_st'=37);
52 [c2AskConfC4] sys_st = 37 -> R_C:(sys_st'=38) + (1-R_C):(sys_st'=ERROR_STATE);
53 [c4AskConfC2] sys_st = 38 -> R_C:(sys_st'=39) + (1-R_C):(sys_st'=ERROR_STATE);
54 [c2AskConfC1] sys_st = 39 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
55 [c2SeesObj] sys_st = 40 -> (sys_st'=41);
56 [c3StartSearchC1] sys_st = 40 -> R_C:(sys_st'=36) + (1-R_C):(sys_st'=ERROR_STATE);
57 [c2AskConfC4] sys_st = 41 -> R_C:(sys_st'=42) + (1-R_C):(sys_st'=ERROR_STATE);
58 [c3StartSearchC1] sys_st = 41 -> R_C:(sys_st'=37) + (1-R_C):(sys_st'=ERROR_STATE);
59 [c3StartSearchC1] sys_st = 42 -> R_C:(sys_st'=38) + (1-R_C):(sys_st'=ERROR_STATE);
60 [c4AskConfC2] sys_st = 42 -> R_C:(sys_st'=43) + (1-R_C):(sys_st'=ERROR_STATE);
61 [c3StartSearchC1] sys_st = 43 -> R_C:(sys_st'=39) + (1-R_C):(sys_st'=ERROR_STATE);
62 [c2AskConfC1] sys_st = 44 -> R_C:(sys_st'=45) + (1-R_C):(sys_st'=ERROR_STATE);
63 [c2AskConfC4] sys_st = 45 -> R_C:(sys_st'=46) + (1-R_C):(sys_st'=ERROR_STATE);
64 [c2StartTracking] sys_st = 46 -> (sys_st'=47);
65 [c2LosesObj] sys_st = 47 -> (sys_st'=48);
66 [c2StartSearchC1] sys_st = 48 -> R_C:(sys_st'=49) + (1-R_C):(sys_st'=ERROR_STATE);
67 [c2StartSearchC4] sys_st = 49 -> R_C:(sys_st'=50) + (1-R_C):(sys_st'=ERROR_STATE);
68 [c1SeesObj] sys_st = 49 -> (sys_st'=56);
69 [c1SeesObj] sys_st = 50 -> (sys_st'=51);
70 [c1FoundObjC2] sys_st = 51 -> R_C:(sys_st'=52) + (1-R_C):(sys_st'=ERROR_STATE);
71 [c2AskConfC1] sys_st = 52 -> R_C:(sys_st'=38) + (1-R_C):(sys_st'=ERROR_STATE);
72 [c1SendConfC2] sys_st = 53 -> R_C:(sys_st'=39) + (1-R_C):(sys_st'=ERROR_STATE);
73 [c2AllowC1Track] sys_st = 54 -> R_C:(sys_st'=55) + (1-R_C):(sys_st'=ERROR_STATE);
74 [c1StartTracking] sys_st = 55 -> (sys_st'=18);
75 [c2StartSearchC4] sys_st = 56 -> R_C:(sys_st'=51) + (1-R_C):(sys_st'=ERROR_STATE);
76 [c3AskConfC1] sys_st = 57 -> R_C:(sys_st'=58) + (1-R_C):(sys_st'=ERROR_STATE);
77 [c4SeesObj] sys_st = 57 -> (sys_st'=94);
78 [c3StartTracking] sys_st = 58 -> (sys_st'=77);
79 [c4SeesObj] sys_st = 58 -> (sys_st'=59);
80 [c3StartTracking] sys_st = 59 -> (sys_st'=76);
81 [c4AskConfC2] sys_st = 59 -> R_C:(sys_st'=60) + (1-R_C):(sys_st'=ERROR_STATE);
82 [c3StartTracking] sys_st = 60 -> (sys_st'=61);
83 [c4StartTracking] sys_st = 61 -> (sys_st'=62);
84 [c3LosesObj] sys_st = 62 -> (sys_st'=63);
85 [c3StartSearchC1] sys_st = 63 -> R_C:(sys_st'=64) + (1-R_C):(sys_st'=ERROR_STATE);
86 [c4LosesObj] sys_st = 63 -> (sys_st'=74);
87 [c4LosesObj] sys_st = 64 -> (sys_st'=65);
88 [c4StartSearchC2] sys_st = 65 -> R_C:(sys_st'=66) + (1-R_C):(sys_st'=ERROR_STATE);
89 [c1SeesObj] sys_st = 65 -> (sys_st'=72);
90 [c1SeesObj] sys_st = 66 -> (sys_st'=67);
91 [c1FoundObjC3] sys_st = 67 -> R_C:(sys_st'=68) + (1-R_C):(sys_st'=ERROR_STATE);
```

```
92 [c2SeesObj] sys_st = 67 -> (sys_st'=70);
93 [c2SeesObj] sys_st = 68 -> (sys_st'=76);
94 [c2FoundObjC4] sys_st = 69 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
95 [c1FoundObjC3] sys_st = 70 -> R_C:(sys_st'=69) + (1-R_C):(sys_st'=ERROR_STATE);
96 [c2FoundObjC4] sys_st = 70 -> R_C:(sys_st'=71) + (1-R_C):(sys_st'=ERROR_STATE);
97 [c1FoundObjC3] sys_st = 71 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
98 [c4StartSearchC2] sys_st = 72 -> R_C:(sys_st'=67) + (1-R_C):(sys_st'=ERROR_STATE);
99 [c1FoundObjC3] sys_st = 72 -> R_C:(sys_st'=73) + (1-R_C):(sys_st'=ERROR_STATE);
100 [c4StartSearchC2] sys_st = 73 -> R_C:(sys_st'=68) + (1-R_C):(sys_st'=ERROR_STATE);
101 [c3StartSearchC1] sys_st = 74 -> R_C:(sys_st'=65) + (1-R_C):(sys_st'=ERROR_STATE);
102 [c4StartSearchC2] sys_st = 74 -> R_C:(sys_st'=75) + (1-R_C):(sys_st'=ERROR_STATE);
103 [c3StartSearchC1] sys_st = 75 -> R_C:(sys_st'=66) + (1-R_C):(sys_st'=ERROR_STATE);
104 [c4AskConfC2] sys_st = 76 -> R_C:(sys_st'=61) + (1-R_C):(sys_st'=ERROR_STATE);
105 [c3LosesObj] sys_st = 77 -> (sys_st'=78);
106 [c3StartSearchC1] sys_st = 78 -> R_C:(sys_st'=79) + (1-R_C):(sys_st'=ERROR_STATE);
107 [c2SeesObj] sys_st = 78 -> (sys_st'=93);
108 [c2SeesObj] sys_st = 79 -> (sys_st'=80);
109 [c1SeesObj] sys_st = 80 -> (sys_st'=81);
110 [c1FoundObjC3] sys_st = 81 -> R_C:(sys_st'=82) + (1-R_C):(sys_st'=ERROR_STATE);
111 [c2AskConfC1] sys_st = 82 -> R_C:(sys_st'=83) + (1-R_C):(sys_st'=ERROR_STATE);
112 [c3AskConfC1] sys_st = 83 -> R_C:(sys_st'=84) + (1-R_C):(sys_st'=ERROR_STATE);
113 [c2AskConfC4] sys_st = 83 -> R_C:(sys_st'=91) + (1-R_C):(sys_st'=ERROR_STATE);
114 [c1SendConfC3] sys_st = 84 -> R_C:(sys_st'=85) + (1-R_C):(sys_st'=ERROR_STATE);
115 [c2AskConfC4] sys_st = 84 -> R_C:(sys_st'=89) + (1-R_C):(sys_st'=ERROR_STATE);
116 [c2AskConfC4] sys_st = 85 -> R_C:(sys_st'=86) + (1-R_C):(sys_st'=ERROR_STATE);
117 [c2StartTracking] sys_st = 86 -> (sys_st'=25); //Trace com cenario implicito
118 [c1SendConfC2] sys_st = 86 -> R_C:(sys_st'=87) + (1-R_C):(sys_st'=ERROR_STATE);
119 [c1AllowC2Track] sys_st = 87 -> R_C:(sys_st'=88) + (1-R_C):(sys_st'=ERROR_STATE);
120 [c2StartTracking] sys_st = 88 -> (sys_st'=18);
121 [c1SendConfC3] sys_st = 89 -> R_C:(sys_st'=86) + (1-R_C):(sys_st'=ERROR_STATE);
122 [c2StartTracking] sys_st = 89 -> (sys_st'=90); //Trace com cenario implicito
123 [c1SendConfC3] sys_st = 90 -> R_C:(sys_st'=25) + (1-R_C):(sys_st'=ERROR_STATE);
124 [c3AskConfC1] sys_st = 91 -> R_C:(sys_st'=89) + (1-R_C):(sys_st'=ERROR_STATE);
125 [c2StartTracking] sys_st = 91 -> (sys_st'=92); //Trace com cenario implicito
126 [c3AskConfC1] sys_st = 92 -> R_C:(sys_st'=90) + (1-R_C):(sys_st'=ERROR_STATE);
127 [c3StartSearchC1] sys_st = 93 -> R_C:(sys_st'=80) + (1-R_C):(sys_st'=ERROR_STATE);
128 [c3AskConfC1] sys_st = 94 -> R_C:(sys_st'=59) + (1-R_C):(sys_st'=ERROR_STATE);
129 [c4AskConfC2] sys_st = 94 -> R_C:(sys_st'=95) + (1-R_C):(sys_st'=ERROR_STATE);
130 [c3AskConfC1] sys_st = 95 -> R_C:(sys_st'=60) + (1-R_C):(sys_st'=ERROR_STATE);
131
132 endmodule
```

Listagem C.1: Modelagem na linguagem do PRISM para a Arquitetura 1

C.2 Modelagem na linguagem do PRISM para a Arquitetura 2

```
1 dtmc
2
3 const double R_C;
4 const int FINAL_STATE = 13;
5 const int ERROR_STATE = 73;
6
7 module ArchitectureModel
```

```
8
9  sys_st : [0..73] init 0;
10 [c2SeesObj] sys_st = 0 -> (sys_st'=1);
11 [c3SeesObj] sys_st = 0 -> (sys_st'=24);
12 [c2AskConfC1] sys_st = 1 -> R_C:(sys_st'=2) + (1-R_C):(sys_st'=ERROR_STATE);
13 [c4SeesObj] sys_st = 1 -> (sys_st'=23);
14 [c2AskConfC4] sys_st = 2 -> R_C:(sys_st'=3) + (1-R_C):(sys_st'=ERROR_STATE);
15 [c4SeesObj] sys_st = 2 -> (sys_st'=14);
16 [c2StartTracking] sys_st = 3 -> (sys_st'=4);
17 [c1SeesObj] sys_st = 4 -> (sys_st'=5);
18 [c2LosesObj] sys_st = 5 -> (sys_st'=6);
19 [c2StartSearchC1] sys_st = 6 -> R_C:(sys_st'=7) + (1-R_C):(sys_st'=ERROR_STATE);
20 [c2StartSearchC4] sys_st = 7 -> R_C:(sys_st'=8) + (1-R_C):(sys_st'=ERROR_STATE);
21 [c1FoundObjC2] sys_st = 8 -> R_C:(sys_st'=9) + (1-R_C):(sys_st'=ERROR_STATE);
22 [c2AskConfC1] sys_st = 9 -> R_C:(sys_st'=10) + (1-R_C):(sys_st'=ERROR_STATE);
23 [c1SendConfC2] sys_st = 10 -> R_C:(sys_st'=11) + (1-R_C):(sys_st'=ERROR_STATE);
24 [c2AllowstrackingC1Obj1] sys_st = 11 -> (sys_st'=12);
25 [end] sys_st = 12 -> (sys_st'=FINAL_STATE);
26 [c2AskConfC4] sys_st = 14 -> R_C:(sys_st'=15) + (1-R_C):(sys_st'=ERROR_STATE);
27 [c4AskConfC2] sys_st = 15 -> R_C:(sys_st'=16) + (1-R_C):(sys_st'=ERROR_STATE);
28 [c2StartTracking] sys_st = 15 -> (sys_st'=20); //Trace com o cenario implicito
29 [c4SendConfC2] sys_st = 16 -> R_C:(sys_st'=17) + (1-R_C):(sys_st'=ERROR_STATE);
30 [c2SendConfC4] sys_st = 17 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
31 [c4AllowC2Track] sys_st = 18 -> R_C:(sys_st'=19) + (1-R_C):(sys_st'=ERROR_STATE);
32 [c2StartTracking] sys_st = 19 -> (sys_st'=5);
33 [c2LosesObj] sys_st = 20 -> (sys_st'=21);
34 [c2StartSearchC1] sys_st = 21 -> R_C:(sys_st'=22) + (1-R_C):(sys_st'=ERROR_STATE);
35 [STOP] sys_st = 22 -> (sys_st'=FINAL_STATE);
36 [c2AskConfC1] sys_st = 23 -> R_C:(sys_st'=14) + (1-R_C):(sys_st'=ERROR_STATE);
37 [c3AskConfC1] sys_st = 24 -> R_C:(sys_st'=25) + (1-R_C):(sys_st'=ERROR_STATE);
38 [c3StartTracking] sys_st = 25 -> (sys_st'=26);
39 [c1SeesObj] sys_st = 26 -> (sys_st'=59);
40 [c3LosesObj] sys_st = 26 -> (sys_st'=27);
41 [c3StartSearchC1] sys_st = 27 -> R_C:(sys_st'=28) + (1-R_C):(sys_st'=ERROR_STATE);
42 [c4SeesObj1] sys_st = 27 -> (sys_st'=31);
43 [c1SeesObj1] sys_st = 27 -> (sys_st'=34);
44 [c4SeesObj] sys_st = 28 -> (sys_st'=29);
45 [c4AskConfC2] sys_st = 29 -> R_C:(sys_st'=30) + (1-R_C):(sys_st'=ERROR_STATE);
46 [c4StartTracking] sys_st = 30 -> (sys_st'=12);
47 [c3StartSearchC1] sys_st = 31 -> R_C:(sys_st'=29) + (1-R_C):(sys_st'=ERROR_STATE);
48 [c4AskConfC2] sys_st = 31 -> R_C:(sys_st'=32) + (1-R_C):(sys_st'=ERROR_STATE);
49 [c3StartSearchC1] sys_st = 32 -> R_C:(sys_st'=30) + (1-R_C):(sys_st'=ERROR_STATE);
50 [c4StartTracking] sys_st = 32 -> (sys_st'=33);
51 [c3StartSearchC1] sys_st = 33 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
52 [c3StartSearchC1] sys_st = 34 -> R_C:(sys_st'=35) + (1-R_C):(sys_st'=ERROR_STATE);
53 [c1FoundObjC3] sys_st = 35 -> R_C:(sys_st'=36) + (1-R_C):(sys_st'=ERROR_STATE);
54 [c3AskConfC1] sys_st = 36 -> R_C:(sys_st'=37) + (1-R_C):(sys_st'=ERROR_STATE);
55 [c1SendConfC3] sys_st = 37 -> R_C:(sys_st'=38) + (1-R_C):(sys_st'=ERROR_STATE);
56 [c3AllowC1Track] sys_st = 38 -> R_C:(sys_st'=39) + (1-R_C):(sys_st'=ERROR_STATE);
57 [c1StartTracking] sys_st = 39 -> (sys_st'=40);
58 [c1LosesObj] sys_st = 40 -> (sys_st'=41);
59 [c1StartSearchC3] sys_st = 41 -> R_C:(sys_st'=42) + (1-R_C):(sys_st'=ERROR_STATE);
60 [c1StartSearchC2] sys_st = 41 -> R_C:(sys_st'=54) + (1-R_C):(sys_st'=ERROR_STATE);
61 [c4SeesObj] sys_st = 41 -> (sys_st'=57);
62 [c4SeesObj] sys_st = 42 -> (sys_st'=53);
63 [c1StartSearchC2] sys_st = 42 -> R_C:(sys_st'=43) + (1-R_C):(sys_st'=ERROR_STATE);
64 [c4SeesObj] sys_st = 43 -> (sys_st'=44);
```

```
65 [c2SeesObj] sys_st = 44 -> (sys_st'=45);
66 [c2FoundObjC1] sys_st = 45 -> R_C:(sys_st'=46) + (1-R_C):(sys_st'=ERROR_STATE);
67 [c1AskConfC2] sys_st = 46 -> R_C:(sys_st'=47) + (1-R_C):(sys_st'=ERROR_STATE);
68 [c4AskConfC2] sys_st = 46 -> R_C:(sys_st'=49) + (1-R_C):(sys_st'=ERROR_STATE);
69 [c2SendConfC1] sys_st = 47 -> R_C:(sys_st'=48) + (1-R_C):(sys_st'=ERROR_STATE);
70 [c1AllowC2Track] sys_st = 48 -> R_C:(sys_st'=22) + (1-R_C):(sys_st'=ERROR_STATE);
71 [c2SendConfC4] sys_st = 49 -> R_C:(sys_st'=50) + (1-R_C):(sys_st'=ERROR_STATE);
72 [c1AskConfC2] sys_st = 50 -> R_C:(sys_st'=51) + (1-R_C):(sys_st'=ERROR_STATE);
73 [c2SendConfC1] sys_st = 51 -> R_C:(sys_st'=52) + (1-R_C):(sys_st'=ERROR_STATE);
74 [c1AllowC2Track] sys_st = 52 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
75 [c2SeesObj_c4AsksConfC2] sys_st = 53 -> R_C:(sys_st'=22)  + (1-R_C):(sys_st'=ERROR_STATE)
       ;
76 [c1StartSearchC2] sys_st = 53 -> R_C:(sys_st'=44) + (1-R_C):(sys_st'=ERROR_STATE);
77 [c4SeesObj] sys_st = 54 -> (sys_st'=55);
78 [c1StartSearchC3] sys_st = 54 -> R_C:(sys_st'=43) + (1-R_C):(sys_st'=ERROR_STATE);
79 [c2SeesObj] sys_st = 55 -> (sys_st'=56);
80 [c1StartSearchC3] sys_st = 55 -> R_C:(sys_st'=44) + (1-R_C):(sys_st'=ERROR_STATE);
81 [c1StartSearchC3] sys_st = 56 -> R_C:(sys_st'=45) + (1-R_C):(sys_st'=ERROR_STATE);
82 [c1StartSearchC3] sys_st = 57 -> R_C:(sys_st'=53) + (1-R_C):(sys_st'=ERROR_STATE);
83 [c1StartSearchC2] sys_st = 57 -> R_C:(sys_st'=55) + (1-R_C):(sys_st'=ERROR_STATE);
84 [c2SeesObj_c4AsksConfC2] sys_st = 57 -> R_C:(sys_st'=58) + (1-R_C):(sys_st'=ERROR_STATE);
85 [c1StartSearchC3] sys_st = 58 -> R_C:(sys_st'=22) + (1-R_C):(sys_st'=ERROR_STATE);
86 [c3LosesObj] sys_st = 59 -> (sys_st'=60);
87 [c3StartSearchC1] sys_st = 60 -> R_C:(sys_st'=61) + (1-R_C):(sys_st'=ERROR_STATE);
88 [c1FoundObjC3] sys_st = 61 -> R_C:(sys_st'=62) + (1-R_C):(sys_st'=ERROR_STATE);
89 [c3AskConfC1] sys_st = 62 -> R_C:(sys_st'=63) + (1-R_C):(sys_st'=ERROR_STATE);
90 [c1SendConfC3] sys_st = 63 -> R_C:(sys_st'=64) + (1-R_C):(sys_st'=ERROR_STATE);
91 [c3AllowC1Track] sys_st = 64 -> R_C:(sys_st'=65) + (1-R_C):(sys_st'=ERROR_STATE);
92 [c1StartTracking] sys_st = 65 -> (sys_st'=66);
93 [c1LosesObj] sys_st = 66 -> (sys_st'=67);
94 [c2SeesObj] sys_st = 67 -> R_C:(sys_st'=58) + (1-R_C):(sys_st'=ERROR_STATE);
95 [c1StartSearchC3] sys_st = 67 -> R_C:(sys_st'=68) + (1-R_C):(sys_st'=ERROR_STATE);
96 [c1StartSearchC2] sys_st = 67 -> R_C:(sys_st'=71) + (1-R_C):(sys_st'=ERROR_STATE);
97 [c2SeesObj] sys_st = 68 -> (sys_st'=22);
98 [c1StartSearchC2] sys_st = 68 -> R_C:(sys_st'=69) + (1-R_C):(sys_st'=ERROR_STATE);
99 [c2SeesObj] sys_st = 69 -> (sys_st'=70);
100 [c2FoundObjC1] sys_st = 70 -> R_C:(sys_st'=50) + (1-R_C):(sys_st'=ERROR_STATE);
101 [c2SeesObj] sys_st = 71 -> (sys_st'=72);
102 [c1StartSearchC3] sys_st = 71 -> R_C:(sys_st'=69) + (1-R_C):(sys_st'=ERROR_STATE);
103 [c1StartSearchC3] sys_st = 72 -> R_C:(sys_st'=70) + (1-R_C):(sys_st'=ERROR_STATE);
104
105 endmodule
```

Listagem C.2: Modelagem na linguagem do PRISM para a Arquitetura 2

Anexo D

Constrained Model expresso em FSP

D.1 *Constrained Model* referente à Arquitetura 1 expresso em FSP

```
1  ConstrainedArchitectureModel = Q0,
2   Q0 = (c2SeesObj -> Q1 | c3SeesObj -> Q16 | c1SeesObj -> Q53),
3   Q1 = (c2AskConfC1 -> Q2),
4   Q2 = (c2AskConfC4 -> Q3),
5   Q3 = (c2StartTracking -> Q4),
6   Q4 = (c2LosesObj -> Q5),
7   Q5 = (c2StartSearchC1 -> Q6),
8   Q6 = (c2StartSearchC4 -> Q7 | c1SeesObj -> Q15),
9   Q7 = (c1SeesObj -> Q8),
10  Q8 = (c1FoundObjC2 -> Q9),
11  Q9 = (c2AskConfC1 -> Q10),
12  Q10 = (c1SendConfC2 -> Q11),
13  Q11 = (c2AllowsTrackC1Obj -> Q12),
14  Q12 = (c1StartTracking -> Q13),
15  Q13 = (endAction -> Q14),
16  Q14 = END,
17  Q15 = (c2StartSearchC4 -> Q8),
18  Q16 = (c3AskConfC1 -> Q17 | c4SeesObj -> Q51),
19  Q17 = (c4SeesObj -> Q18 | c3StartsTracking -> Q36),
20  Q18 = (c4AskConfC2 -> Q19 | c3StartsTracking -> Q35),
21  Q19 = (c3StartsTracking -> Q20),
22  Q20 = (c4StartTracking -> Q21),
23  Q21 = (c3LosesObj -> Q22),
24  Q22 = (c3StartSearchC1 -> Q23 | c4LosesObj -> Q33),
25  Q23 = (c4LosesObj -> Q24),
26  Q24 = (c4StartSearchC2 -> Q25 | c1SeesObj -> Q31),
27  Q25 = (c1SeesObj -> Q26),
28  Q26 = (c1FoundObjC3 -> Q27 | c2SeesObj -> Q29),
29  Q27 = (c2SeesObj -> Q28),
30  Q28 = (c2FoundObjC4 -> Q13),
31  Q29 = (c1FoundObjC3 -> Q28 | c2FoundObjC4 -> Q30),
32  Q30 = (c1FoundObjC3 -> Q13),
33  Q31 = (c4StartSearchC2 -> Q26 | c1FoundObjC3 -> Q32),
34  Q32 = (c4StartSearchC2 -> Q27),
```

```
35  Q33 = (c3StartSearchC1 -> Q24 | c4StartSearchC2 -> Q34),
36  Q34 = (c3StartSearchC1 -> Q25),
37  Q35 = (c4AskConfC2 -> Q20),
38  Q36 = (c3LosesObj -> Q37),
39  Q37 = (c3StartSearchC1 -> Q38 | c2SeesObj -> Q50),
40  Q38 = (c2SeesObj -> Q39),
41  Q39 = (c1SeesObj -> Q40),
42  Q40 = (c1FoundObjC3 -> Q41),
43  Q41 = (c2AskConfC1 -> Q42),
44  Q42 = (c3AskConfC1 -> Q43 | c2AskConfC4 -> Q49),
45  Q43 = (c1SendConfC3 -> Q44 | c2AskConfC4 -> Q48),
46  Q44 = (c2AskConfC4 -> Q45),
47  Q45 = (c1SendConfC2 -> Q46),
48  Q46 = (c1AllowC2Track -> Q47),
49  Q47 = (c2StartTracking -> Q13),
50  Q48 = (c1SendConfC3 -> Q45),
51  Q49 = (c3AskConfC1 -> Q48),
52  Q50 = (c3StartSearchC1 -> Q39),
53  Q51 = (c3AskConfC1 -> Q18 | c4AskConfC2 -> Q52),
54  Q52 = (c3AskConfC1 -> Q19),
55  Q53 = (c1AskConfC2 -> Q54 | c3SeesObj -> Q70),
56  Q54 = (c3SeesObj -> Q55),
57  Q55 = (c3AskConfC1 -> Q56),
58  Q56 = (c1AskConfC3 -> Q57),
59  Q57 = (c1SendConfC3 -> Q58),
60  Q58 = (c3SendConfC1 -> Q59),
61  Q59 = (c3AllowsC1Tracks -> Q60),
62  Q60 = (c1StartTracking -> Q61),
63  Q61 = (c1LosesObj -> Q62),
64  Q62 = (c1StartSearchC2 -> Q63),
65  Q63 = (c1StartSearchC3 -> Q64),
66  Q64 = (c3SeesObj -> Q65),
67  Q65 = (c3FoundObjC1 -> Q66),
68  Q66 = (c1AskConfC3 -> Q67),
69  Q67 = (c3SendConfC1 -> Q68),
70  Q68 = (c1AllowsC3Track -> Q69),
71  Q69 = (c3StartsTracking -> Q13),
72  Q70 = (c1AskConfC2 -> Q55 | c1AskConfC3 -> Q71),
73  Q71 = (c1AskConfC2 -> Q72),
74  Q72 = (c3AskConfC1 -> Q73),
75  Q73 = (c3SendConfC1 -> Q74),
76  Q74 = (c1SendConfC3 -> Q75),
77  Q75 = (c1AllowsC3Track -> Q76),
78  Q76 = (c3StartsTracking -> Q77),
79  Q77 = (c3LosesObj -> Q78),
80  Q78 = (c3StartSearchC1 -> Q79 | c4SeesObj -> Q84),
81  Q79 = (c4SeesObj -> Q80),
82  Q80 = (c2SeesObj -> Q81),
83  Q81 = (c2AskConfC4 -> Q82),
84  Q82 = (c4AskConfC2 -> Q83),
85  Q83 = (c2AskConfC1 -> Q13),
86  Q84 = (c3StartSearchC1 -> Q80 | c2SeesObj -> Q85),
87  Q85 = (c3StartSearchC1 -> Q81 | c2AskConfC4 -> Q86),
88  Q86 = (c3StartSearchC1 -> Q82 | c4AskConfC2 -> Q87),
```

```
89  Q87 = (c3StartSearchC1 -> Q83).
```

Listagem D.1: *Constrained Model* referente à Arquitetura 1 expresso em FSP

D.2 *Constrained Model* referente à Arquitetura 2 expresso em FSP

```
1  ConstrainedArchitectureModel = Q0,
2   Q0 = (c2SeesObj1 -> Q1 | c3SeesObj1 -> Q21),
3   Q1 = (c2AsksConfC1Obj1 -> Q2 | c4SeesObj1 -> Q20),
4   Q2 = (c2AsksConfC4Obj1 -> Q3 | c4SeesObj1 -> Q14),
5   Q3 = (c2StartsTrackingObj1 -> Q4),
6   Q4 = (c1seesObj1 -> Q5),
7   Q5 = (c2LosesObj1 -> Q6),
8   Q6 = (c2StartSearchC1 -> Q7),
9   Q7 = (c2StartSearchC4 -> Q8),
10  Q8 = (c1FoundObjC2 -> Q9),
11  Q9 = (c2AsksConfC1Obj1 -> Q10),
12  Q10 = (c1sendsConfC2 -> Q11),
13  Q11 = (c2AllowstrackingC1Obj1 -> Q12),
14  Q12 = (endAction -> Q13),
15  Q13 = END,
16  Q14 = (c2AsksConfC4Obj1 -> Q15),
17  Q15 = (c4AsksConfC2Obj1 -> Q16),
18  Q16 = (c4sendsConfC2Obj1 -> Q17),
19  Q17 = (c2sendsConfC4Obj1 -> Q18),
20  Q18 = (c4AllowsC2TrackObj1 -> Q19),
21  Q19 = (c2StartsTrackingObj1 -> Q5),
22  Q20 = (c2AsksConfC1Obj1 -> Q14),
23  Q21 = (c3AsksConfC1Obj1 -> Q22),
24  Q22 = (c3StartsTrackingObj1 -> Q23),
25  Q23 = (c1SeesObj1 -> Q24 | c3LosesObj1 -> Q42),
26  Q24 = (c3LosesObj1 -> Q25),
27  Q25 = (c3StartSearchC1 -> Q26),
28  Q26 = (c1FoundObj1C3 -> Q27),
29  Q27 = (c3AsksConfC1Obj1 -> Q28),
30  Q28 = (c1SendsConfC3Obj1 -> Q29),
31  Q29 = (c3AllowsTrackC1Obj1 -> Q30),
32  Q30 = (c1StartsTrackingObj1 -> Q31),
33  Q31 = (c1LosesObj1 -> Q32),
34  Q32 = (endAction -> Q13 | c1StartSearchC3 -> Q33 | c1StartSearchC2 -> Q39 | c2SeesObj1
        -> Q41),
35  Q33 = (c2SeesObj1 -> Q12 | endAction -> Q13 | c1StartSearchC2 -> Q34),
36  Q34 = (endAction -> Q13 | c2SeesObj1 -> Q35),
37  Q35 = (endAction -> Q13 | c2FoundObj1C1 -> Q36),
38  Q36 = (endAction -> Q13 | c1AsksConfC2Obj1 -> Q37),
39  Q37 = (endAction -> Q13 | c2SendsConC1Obj1 -> Q38),
40  Q38 = (c1AllowsTrackC2Obj1 -> Q12 | endAction -> Q13),
41  Q39 = (endAction -> Q13 | c1StartSearchC3 -> Q34 | c2SeesObj1 -> Q40),
42  Q40 = (endAction -> Q13 | c1StartSearchC3 -> Q35),
43  Q41 = (c1StartSearchC3 -> Q12 | endAction -> Q13),
44  Q42 = (c3startSearchC1 -> Q43 | c4SeesObj1 -> Q46 | c1SeesObj1 -> Q49),
45  Q43 = (c4SeesObj1 -> Q44),
46  Q44 = (c4AsksConfC2Obj1 -> Q45),
```

```
47 Q45 = (cam4.object1.c4StartsTracking -> Q12),
48 Q46 = (c3startSearchC1 -> Q44 | c4AsksConfC2Obj1 -> Q47),
49 Q47 = (c3startSearchC1 -> Q45 | cam4.object1.c4StartsTracking -> Q48),
50 Q48 = (c3startSearchC1 -> Q12),
51 Q49 = (c3StartSearchC1 -> Q50),
52 Q50 = (c1FoundObj1C3 -> Q51),
53 Q51 = (c3AsksConfC1Obj1 -> Q52),
54 Q52 = (c1SendsConfC3Obj1 -> Q53),
55 Q53 = (c3AllowsTrackC1Obj1 -> Q54),
56 Q54 = (c1StartsTrackingObj1 -> Q55),
57 Q55 = (c1LosesObj1 -> Q56),
58 Q56 = (endAction -> Q13 | c1StartSearchC3 -> Q57 | c1StartSearchC2 -> Q64 | c4SeesObj1
       -> Q67),
59 Q57 = (endAction -> Q13 | c1StartSearchC2 -> Q58 | c4SeesObj1 -> Q63),
60 Q58 = (endAction -> Q13 | c4SeesObj1 -> Q59),
61 Q59 = (endAction -> Q13 | c2SeesObj1 -> Q60),
62 Q60 = (endAction -> Q13 | c2FoundObj1C1 -> Q61),
63 Q61 = (endAction -> Q13 | c1AsksConfC2Obj1 -> Q37 | c4AsksConfC2Obj1 -> Q62),
64 Q62 = (endAction -> Q13 | c2sendsConfC4Obj1 -> Q36),
65 Q63 = ({c2SeesObj1, c4AsksConfC2Obj1} -> Q12 | endAction -> Q13 | c1StartSearchC2 -> Q59
       ),
66 Q64 = (endAction -> Q13 | c1StartSearchC3 -> Q58 | c4SeesObj1 -> Q65),
67 Q65 = (endAction -> Q13 | c1StartSearchC3 -> Q59 | c2SeesObj1 -> Q66),
68 Q66 = (endAction -> Q13 |c1StartSearchC3 -> Q60),
69 Q67 = (endAction -> Q13 | {c2SeesObj1, c4AsksConfC2Obj1} -> Q41 | c1StartSearchC3 -> Q63
        | c1StartSearchC2 -> Q65).
```

Listagem D.2: *Constrained Model* referente à Arquitetura 2 expresso em FSP

Anexo E

Constrained Model expresso na linguagem da PRISM

E.1 *Constrained Model* referente à Arquitetura 1 expresso na linguagem da PRISM

```
1  dtmc
2
3  const double R_C;
4  const int FINAL_STATE = 14;
5  const int ERROR_STATE = ERROR_STATE = 88;
6
7  module ArchitectureModel
8
9  sys_st : [0..88] init 0;
10
11 [c2SeesObj] sys_st = 0 -> (sys_st'=1);
12 [c3SeesObj] sys_st = 0 -> (sys_st'=16);
13 [c1SeesObj] sys_st = 0 -> (sys_st'=53);
14 [c2AskConfC1] sys_st = 1 -> R_C:(sys_st'=2) + (1-R_C):(sys_st'=ERROR_STATE);
15 [c2AskConfC4] sys_st = 2 -> R_C:(sys_st'=3) + (1-R_C):(sys_st'=ERROR_STATE);
16 [c2StartTracking] sys_st = 3 -> R_C:(sys_st'=4) + (1-R_C):(sys_st'=ERROR_STATE);
17 [c2LosesObj] sys_st = 4 -> (sys_st'=5);
18 [c2StartSearchC1] sys_st = 5 -> R_C:(sys_st'=6) + (1-R_C):(sys_st'=ERROR_STATE);
19 [c2StartSearchC4] sys_st = 6 -> R_C:(sys_st'=7) + (1-R_C):(sys_st'=ERROR_STATE);
20 [c1SeesObj] sys_st = 6 -> R_C:(sys_st'=15) + (1-R_C):(sys_st'=ERROR_STATE);
21 [c1SeesObj]  sys_st = 7 -> R_C:(sys_st'=8) + (1-R_C):(sys_st'=ERROR_STATE);
22 [c1FoundObjC2]  sys_st = 8 -> R_C:(sys_st'=9) + (1-R_C):(sys_st'=ERROR_STATE);
23 [c2AskConfC1] sys_st = 9 -> R_C:(sys_st'=10) + (1-R_C):(sys_st'=ERROR_STATE);
24 [c1SendConfC2] sys_st = 10 -> R_C:(sys_st'=11) + (1-R_C):(sys_st'=ERROR_STATE);
25 [c2AllowsTrackC1Obj] sys_st = 11 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
26 [c1StartTracking] sys_st = 12 -> (sys_st'=13);
27 [endAction] sys_st = 13 -> (sys_st'=14);
28 [c2StartSearchC4] sys_st = 15 -> R_C:(sys_st'=8) + (1-R_C):(sys_st'=ERROR_STATE);
29 [c3AskConfC1] sys_st = 16 -> R_C:(sys_st'=17) + (1-R_C):(sys_st'=ERROR_STATE);
30 [c4SeesObj] sys_st = 16 -> (sys_st'=51);
31 [c4SeesObj] sys_st = 17 -> (sys_st'=18);
```

```
[c3StartsTracking] sys_st = 17 -> (sys_st'=36);
[c4AskConfC2] sys_st = 18 -> R_C:(sys_st'=19) + (1-R_C):(sys_st'=ERROR_STATE);
[c3StartsTracking] sys_st = 18 -> (sys_st'=35);
[c3StartsTracking] sys_st = 19 -> (sys_st'=20);
[c4StartsTracking] sys_st = 20 -> (sys_st'=21);
[c3LosesObj] sys_st = 21 -> (sys_st'=22);
[c3StartSearchC1] sys_st = 22 -> R_C:(sys_st'=23) + (1-R_C):(sys_st'=ERROR_STATE);
[c4LosesObj] sys_st = 22 -> (sys_st'=33);
[c4LosesObj] sys_st = 23 -> (sys_st'=24);
[c4StartSearchC2] sys_st = 24 -> R_C:(sys_st'=25) + (1-R_C):(sys_st'=ERROR_STATE);
[c1SeesObj] sys_st = 24 -> (sys_st'=31);
[c1SeesObj] sys_st = 25 -> (sys_st'=26);
[c1FoundObjC3] sys_st = 26 -> R_C:(sys_st'=27) + (1-R_C):(sys_st'=ERROR_STATE);
[c2SeesObj] sys_st = 26 -> (sys_st'=29);
[c2SeesObj] sys_st = 27 -> (sys_st'=28);
[c2FoundObjC4] sys_st = 28 -> R_C:(sys_st'=13) + (1-R_C):(sys_st'=ERROR_STATE);
[c1FoundObjC3] sys_st = 29 -> R_C:(sys_st'=28) + (1-R_C):(sys_st'=ERROR_STATE);
[c2FoundObjC4] sys_st = 29 -> R_C:(sys_st'=30) + (1-R_C):(sys_st'=ERROR_STATE);
[c1FoundObjC3] sys_st = 30 -> R_C:(sys_st'=13) + (1-R_C):(sys_st'=ERROR_STATE);
[c4StartSearchC2] sys_st = 31 -> R_C:(sys_st'=26) + (1-R_C):(sys_st'=ERROR_STATE);
[c1FoundObjC3] sys_st = 31 -> R_C:(sys_st'=32) + (1-R_C):(sys_st'=ERROR_STATE);
[c4StartSearchC2] sys_st = 32 -> R_C:(sys_st'=27) + (1-R_C):(sys_st'=ERROR_STATE);
[c3StartSearchC1] sys_st = 33 -> R_C:(sys_st'=24) + (1-R_C):(sys_st'=ERROR_STATE);
[c4StartSearchC2] sys_st = 33 -> R_C:(sys_st'=34) + (1-R_C):(sys_st'=ERROR_STATE);
[c3StartSearchC1] sys_st = 34 -> R_C:(sys_st'=25) + (1-R_C):(sys_st'=ERROR_STATE);
[c4AskConfC2] sys_st = 35 -> R_C:(sys_st'=20) + (1-R_C):(sys_st'=ERROR_STATE);
[c3LosesObj] sys_st = 36 -> (sys_st'=37);
[c3StartSearchC1] sys_st = 37 -> R_C:(sys_st'=38) + (1-R_C):(sys_st'=ERROR_STATE);
[c2SeesObj] sys_st = 37 -> (sys_st'=50);
[c2SeesObj] sys_st = 38 -> (sys_st'=39);
[c1SeesObj] sys_st = 39 -> (sys_st'=40);
[c1FoundObjC3] sys_st = 40 -> R_C:(sys_st'=41) + (1-R_C):(sys_st'=ERROR_STATE);
[c2AskConfC1] sys_st = 41 -> R_C:(sys_st'=42) + (1-R_C):(sys_st'=ERROR_STATE);
[c3AskConfC1] sys_st = 42 -> R_C:(sys_st'=43) + (1-R_C):(sys_st'=ERROR_STATE);
[c2AskConfC4] sys_st = 42 -> R_C:(sys_st'=49) + (1-R_C):(sys_st'=ERROR_STATE);
[c1SendConfC3] sys_st = 43 -> R_C:(sys_st'=44) + (1-R_C):(sys_st'=ERROR_STATE);
[c2AskConfC4] sys_st = 43 -> R_C:(sys_st'=48) + (1-R_C):(sys_st'=ERROR_STATE);
[c2AskConfC4] sys_st = 44 -> R_C:(sys_st'=45) + (1-R_C):(sys_st'=ERROR_STATE);
[c1SendConfC2] sys_st = 45 -> R_C:(sys_st'=46) + (1-R_C):(sys_st'=ERROR_STATE);
[c1AllowC2Track] sys_st = 46 -> R_C:(sys_st'=47) + (1-R_C):(sys_st'=ERROR_STATE);
[c2StartTracking] sys_st = 47 -> (sys_st'=13);
[c1SendConfC3] sys_st = 48 -> R_C:(sys_st'=45) + (1-R_C):(sys_st'=ERROR_STATE);
[c3AskConfC1] sys_st = 49 -> R_C:(sys_st'=48) + (1-R_C):(sys_st'=ERROR_STATE);
[c3StartSearchC1] sys_st = 50 -> R_C:(sys_st'=39) + (1-R_C):(sys_st'=ERROR_STATE);
[c3AskConfC1] sys_st = 51 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
[c4AskConfC2] sys_st = 51 -> R_C:(sys_st'=52) + (1-R_C):(sys_st'=ERROR_STATE);
[c3AskConfC1] sys_st = 52 -> R_C:(sys_st'=19) + (1-R_C):(sys_st'=ERROR_STATE);
[c1AskConfC2] sys_st = 53 -> R_C:(sys_st'=54) + (1-R_C):(sys_st'=ERROR_STATE);
[c3SeesObj] sys_st = 53 -> (sys_st'=70);
[c3SeesObj] sys_st = 54 -> (sys_st'=55);
[c3AskConfC1] sys_st = 55 -> R_C:(sys_st'=56) + (1-R_C):(sys_st'=ERROR_STATE);
[c1AskConfC3] sys_st = 56 -> R_C:(sys_st'=57) + (1-R_C):(sys_st'=ERROR_STATE);
[c1SendConfC3] sys_st = 57 -> R_C:(sys_st'=58) + (1-R_C):(sys_st'=ERROR_STATE);
[c3SendConfC1] sys_st = 58 -> R_C:(sys_st'=59) + (1-R_C):(sys_st'=ERROR_STATE);
[c3AllowsC1Tracks] sys_st = 59 -> R_C:(sys_st'=60) + (1-R_C):(sys_st'=ERROR_STATE);
[c1StartTracking] sys_st = 60 -> (sys_st'=61);
[c1LosesObj] sys_st = 61 -> (sys_st'=62);
```

```
89 [c1StartSearchC2] sys_st = 62 -> R_C:(sys_st'=63) + (1-R_C):(sys_st'=ERROR_STATE);
90 [c1StartSearchC3] sys_st = 63 -> R_C:(sys_st'=64) + (1-R_C):(sys_st'=ERROR_STATE);
91 [c3SeesObj] sys_st = 64 -> (sys_st'=65);
92 [c3FoundObjC1] sys_st = 65 -> R_C:(sys_st'=66) + (1-R_C):(sys_st'=ERROR_STATE);
93 [c1AskConfC3] sys_st = 66 -> R_C:(sys_st'=67) + (1-R_C):(sys_st'=ERROR_STATE);
94 [c3SendConfC1] sys_st = 67 -> R_C:(sys_st'=68) + (1-R_C):(sys_st'=ERROR_STATE);
95 [c1AllowsC3Track] sys_st = 68 -> R_C:(sys_st'=69) + (1-R_C):(sys_st'=ERROR_STATE);
96 [c3StartsTracking] sys_st = 69 -> (sys_st'=13);
97 [c1AskConfC2] sys_st = 70 -> R_C:(sys_st'=55) + (1-R_C):(sys_st'=ERROR_STATE);
98 [c1AskConfC3] sys_st = 70 -> R_C:(sys_st'=71) + (1-R_C):(sys_st'=ERROR_STATE);
99 [c1AskConfC2] sys_st = 71 -> R_C:(sys_st'=72) + (1-R_C):(sys_st'=ERROR_STATE);
100 [c3AskConfC1] sys_st = 72 -> R_C:(sys_st'=73) + (1-R_C):(sys_st'=ERROR_STATE);
101 [c3SendConfC1] sys_st = 73 -> R_C:(sys_st'=74) + (1-R_C):(sys_st'=ERROR_STATE);
102 [c1SendConfC3] sys_st = 74 -> R_C:(sys_st'=75) + (1-R_C):(sys_st'=ERROR_STATE);
103 [c1AllowsC3Track] sys_st = 75 -> R_C:(sys_st'=76) + (1-R_C):(sys_st'=ERROR_STATE);
104 [c3StartsTracking] sys_st = 76 -> (sys_st'=77);
105 [c3LosesObj] sys_st = 77 -> (sys_st'=78);
106 [c3StartSearchC1] sys_st = 78 -> R_C:(sys_st'=79) + (1-R_C):(sys_st'=ERROR_STATE);
107 [c4SeesObj] sys_st = 78 -> (sys_st'=84);
108 [c4SeesObj] sys_st = 79 -> (sys_st'=80);
109 [c2SeesObj] sys_st = 80 -> (sys_st'=81);
110 [c2AskConfC4] sys_st = 81 -> R_C:(sys_st'=82) + (1-R_C):(sys_st'=ERROR_STATE);
111 [c4AskConfC2] sys_st = 82 -> R_C:(sys_st'=83) + (1-R_C):(sys_st'=ERROR_STATE);
112 [c2AskConfC1] sys_st = 83 -> R_C:(sys_st'=13) + (1-R_C):(sys_st'=ERROR_STATE);
113 [c3StartSearchC1] sys_st = 84 -> R_C:(sys_st'=80) + (1-R_C):(sys_st'=ERROR_STATE);
114 [c2SeesObj] sys_st = 84 -> (sys_st'=85);
115 [c3StartSearchC1] sys_st = 85 -> R_C:(sys_st'=81) + (1-R_C):(sys_st'=ERROR_STATE);
116 [c2AskConfC4] sys_st = 85 -> R_C:(sys_st'=86) + (1-R_C):(sys_st'=ERROR_STATE);
117 [c3StartSearchC1] sys_st = 86 -> R_C:(sys_st'=82) + (1-R_C):(sys_st'=ERROR_STATE);
118 [c4AskConfC2] sys_st = 86 -> R_C:(sys_st'=87) + (1-R_C):(sys_st'=ERROR_STATE);
119 [c3StartSearchC1] sys_st = 87 -> R_C:(sys_st'=83) + (1-R_C):(sys_st'=ERROR_STATE);
120
121 endmodule
```

Listagem E.1: *Constrained Model* referente à Arquitetura 1 expresso na linguagem da PRISM

E.2 *Constrained Model* referente à Arquitetura 2 expresso na linguagem da PRISM

```
1 dtmc
2
3 const double R_C;
4
5 module ArchitectureModel
6
7 sys_st : [0..68] init 0;
8 const int FINAL_STATE = 13;
9 const int ERROR_STATE = ERROR_STATE = 68;
10
11 [c2SeesObj1] sys_st = 0 -> (sys_st'=1);
12 [c3SeesObj1] sys_st = 0 -> (sys_st'=21);
13 [c2AsksConfC1Obj1] sys_st = 1 -> R_C:(sys_st'=2) + (1-R_C):(sys_st'=ERROR_STATE);
14 [c4SeesObj1] sys_st = 1 -> (sys_st'=20);
```

```
15 [c2AsksConfC4Obj1] sys_st = 2 -> R_C:(sys_st'=3) + (1-R_C):(sys_st'=ERROR_STATE);
16 [c4SeesObj1] sys_st = 2 -> (sys_st'=14);
17 [c2StartsTrackingObj1] sys_st = 3 -> (sys_st'=4);
18 [c1seesObj1] sys_st = 4 -> (sys_st'=5);
19 [c2LosesObj1] sys_st = 5 -> (sys_st'=6);
20 [c2StartSearchC1] sys_st = 6 -> R_C:(sys_st'=7) + (1-R_C):(sys_st'=ERROR_STATE);
21 [c2StartSearchC4] sys_st = 7 -> R_C:(sys_st'=8) + (1-R_C):(sys_st'=ERROR_STATE);
22 [c1FoundObjC2] sys_st = 8 -> (sys_st'=9);
23 [c2AsksConfC1Obj1] sys_st = 9 -> R_C:(sys_st'=10) + (1-R_C):(sys_st'=ERROR_STATE);
24 [c1sendsConfC2] sys_st = 10 -> R_C:(sys_st'=11) + (1-R_C):(sys_st'=ERROR_STATE);
25 [c2AllowstrackingC1Obj1] sys_st = 11 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
26 [endAction] sys_st = 12 -> (sys_st'=FINAL_STATE);
27 [c2AsksConfC4Obj1] sys_st = 14 -> R_C:(sys_st'=15) + (1-R_C):(sys_st'=ERROR_STATE);
28 [c4AsksConfC2Obj1] sys_st = 15 -> R_C:(sys_st'=16) + (1-R_C):(sys_st'=ERROR_STATE);
29 [c4sendsConfC2Obj1] sys_st = 16 -> R_C:(sys_st'=17) + (1-R_C):(sys_st'=ERROR_STATE);
30 [c2sendsConfC4Obj1] sys_st = 17 -> R_C:(sys_st'=18) + (1-R_C):(sys_st'=ERROR_STATE);
31 [c4AllowsC2TrackObj1] sys_st = 18 -> R_C:(sys_st'=19) + (1-R_C):(sys_st'=ERROR_STATE);
32 [c2StartsTrackingObj1] sys_st = 19 -> (sys_st'=5);
33 [c2AsksConfC1Obj1] sys_st = 20 -> R_C:(sys_st'=14) + (1-R_C):(sys_st'=ERROR_STATE);
34 [c3AsksConfC1Obj1] sys_st = 21 -> R_C:(sys_st'=22) + (1-R_C):(sys_st'=ERROR_STATE);
35 [c3StartsTrackingObj1] sys_st = 22 -> (sys_st'=23);
36 [c1SeesObj1] sys_st = 23 -> (sys_st'=24);
37 [c3LosesObj1] sys_st = 23 -> (sys_st'=42);
38 [c3LosesObj1] sys_st = 24 -> (sys_st'=25);
39 [c3StartSearchC1] sys_st = 25 -> R_C:(sys_st'=26) + (1-R_C):(sys_st'=ERROR_STATE);
40 [c1FoundObj1C3] sys_st = 26 -> R_C:(sys_st'=27) + (1-R_C):(sys_st'=ERROR_STATE);
41 [c3AsksConfC1Obj1] sys_st = 27 -> R_C:(sys_st'=28) + (1-R_C):(sys_st'=ERROR_STATE);
42 [c1SendsConfC3Obj1] sys_st = 28 -> R_C:(sys_st'=29) + (1-R_C):(sys_st'=ERROR_STATE);
43 [c3AllowsTrackC1Obj1] sys_st = 29 -> R_C:(sys_st'=30) + (1-R_C):(sys_st'=ERROR_STATE);
44 [c1StartsTrackingObj1] sys_st = 30 -> (sys_st'=31);
45 [c1LosesObj1] sys_st = 31 -> (sys_st'=32);
46 [endAction] sys_st = 32 -> (sys_st'= FINAL_STATE);
47 [c1StartSearchC3] sys_st = 32 -> R_C:(sys_st'=33) + (1-R_C):(sys_st'=ERROR_STATE);
48 [c1StartSearchC2] sys_st = 32 -> R_C:(sys_st'=39) + (1-R_C):(sys_st'=ERROR_STATE);
49 [c2SeesObj1] sys_st = 32 -> (sys_st'=41);
50 [c2SeesObj1] sys_st = 33 -> (sys_st'=12);
51 [endAction] sys_st = 33 -> (sys_st'= FINAL_STATE);
52 [c1StartSearchC2] sys_st = 33 -> R_C:(sys_st'=34) + (1-R_C):(sys_st'=ERROR_STATE);
53 [endAction] sys_st = 34 -> (sys_st'= FINAL_STATE);
54 [c2SeesObj1] sys_st = 34 -> (sys_st'=35);
55 [endAction] sys_st = 35 -> (sys_st'= FINAL_STATE);
56 [c2FoundObj1C1] sys_st = 35 -> (sys_st'=36);
57 [endAction] sys_st = 36 -> (sys_st'= FINAL_STATE);
58 [c1AsksConfC2Obj1] sys_st = 36 -> R_C:(sys_st'=37) + (1-R_C):(sys_st'=ERROR_STATE);
59 [endAction] sys_st = 37 -> (sys_st'= FINAL_STATE);
60 [c2SendsConC1Obj1]  sys_st = 37 -> R_C:(sys_st'=38) + (1-R_C):(sys_st'=ERROR_STATE);
61 [c1AllowsTrackC2Obj1] sys_st = 38 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
62 [endAction] sys_st = 38 -> (sys_st'= FINAL_STATE);
63 [endAction] sys_st = 39 -> (sys_st'= FINAL_STATE);
64 [c1StartSearchC3] sys_st = 39 -> R_C:(sys_st'=34) + (1-R_C):(sys_st'=ERROR_STATE);
65 [c2SeesObj1] sys_st = 39 -> (sys_st'= 40);
66 [endAction] sys_st = 40 -> (sys_st'= FINAL_STATE);
67 [c1StartSearchC3] sys_st = 40 -> R_C:(sys_st'=35) + (1-R_C):(sys_st'=ERROR_STATE);
68 [c1StartSearchC3] sys_st = 41 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
69 [endAction] sys_st = 41 -> (sys_st'= FINAL_STATE);
70 [c3startSearchC1] sys_st = 42 -> R_C:(sys_st'=43) + (1-R_C):(sys_st'=ERROR_STATE);
71 [c4SeesObj1] sys_st = 42 -> (sys_st'= 46);
```

```
72 [c1SeesObj1] sys_st = 42 -> (sys_st'= 49);
73 [c4SeesObj1] sys_st = 43 -> (sys_st'= 44);
74 [c4AsksConfC2Obj1]  sys_st = 44 -> R_C:(sys_st'=45) + (1-R_C):(sys_st'=ERROR_STATE);
75 [c4StartsTracking] sys_st = 45 -> (sys_st'= 12);
76 [c3startSearchC1] sys_st = 46 -> R_C:(sys_st'=44) + (1-R_C):(sys_st'=ERROR_STATE);
77 [c4AsksConfC2Obj1] sys_st = 46 -> R_C:(sys_st'=47) + (1-R_C):(sys_st'=ERROR_STATE);
78 [c3startSearchC1] sys_st = 47 -> R_C:(sys_st'=45) + (1-R_C):(sys_st'=ERROR_STATE);
79 [c4StartsTracking] sys_st = 47 -> (sys_st'= 48);
80 [c3startSearchC1]  sys_st = 48 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=ERROR_STATE);
81 [c3StartSearchC1]  sys_st = 49 -> R_C:(sys_st'=50) + (1-R_C):(sys_st'=ERROR_STATE);
82 [c1FoundObj1C3]  sys_st = 50 -> R_C:(sys_st'=51) + (1-R_C):(sys_st'=ERROR_STATE);
83 [c3AsksConfC1Obj1]  sys_st = 51 -> R_C:(sys_st'=52) + (1-R_C):(sys_st'=ERROR_STATE);
84 [c1SendsConfC3Obj1]  sys_st = 52 -> R_C:(sys_st'=53) + (1-R_C):(sys_st'=ERROR_STATE);
85 [c3AllowsTrackC1Obj1]  sys_st = 53 -> R_C:(sys_st'=54) + (1-R_C):(sys_st'=ERROR_STATE);
86 [c1StartsTrackingObj1] sys_st = 54 -> (sys_st'= 55);
87 [c1LosesObj1] sys_st = 55 -> (sys_st'= 56);
88 [endAction] sys_st = 56 -> (sys_st'= FINAL_STATE);
89 [c1StartSearchC3] sys_st = 56 -> R_C:(sys_st'=57) + (1-R_C):(sys_st'=ERROR_STATE);
90 [c1StartSearchC2] sys_st = 56 -> R_C:(sys_st'=64) + (1-R_C):(sys_st'=ERROR_STATE);
91 [c4SeesObj1] sys_st = 56 -> (sys_st'= 67);
92 [endAction] sys_st = 57 -> (sys_st'= FINAL_STATE);
93 [c1StartSearchC2] sys_st = 57 -> R_C:(sys_st'=58) + (1-R_C):(sys_st'=ERROR_STATE);
94 [c4SeesObj1] sys_st = 57 -> (sys_st'= 63);
95 [endAction] sys_st = 58 -> (sys_st'= FINAL_STATE);
96 [c4SeesObj1] sys_st = 58 -> (sys_st'= 59);
97 [endAction] sys_st = 59 -> (sys_st'= FINAL_STATE);
98 [c2SeesObj1] sys_st = 59 -> (sys_st'= 60);
99 [endAction] sys_st = 60 -> (sys_st'= FINAL_STATE);
100 [c2FoundObj1C1] sys_st = 60 -> R_C:(sys_st'=61) + (1-R_C):(sys_st'=ERROR_STATE);
101 [endAction] sys_st = 61 -> (sys_st'= FINAL_STATE);
102 [c1AsksConfC2Obj1] sys_st = 61 -> R_C:(sys_st'=37) + (1-R_C):(sys_st'=ERROR_STATE);
103 [c4AsksConfC2Obj1] sys_st = 61 -> R_C:(sys_st'=62) + (1-R_C):(sys_st'=ERROR_STATE);
104 [endAction] sys_st = 62 -> (sys_st'= FINAL_STATE);
105 [c2sendsConfC4Obj1] sys_st = 62 -> R_C:(sys_st'=36) + (1-R_C):(sys_st'=ERROR_STATE);
106 [c2SeesObj1_c4AsksConfC2Obj1] sys_st = 63 -> R_C:(sys_st'=12) + (1-R_C):(sys_st'=
        ERROR_STATE);
107 [endAction] sys_st = 63 -> (sys_st'= FINAL_STATE);
108 [c1StartSearchC2] sys_st = 63 -> R_C:(sys_st'=59) + (1-R_C):(sys_st'=ERROR_STATE);
109 [endAction] sys_st = 64 -> (sys_st'= FINAL_STATE);
110 [c1StartSearchC3] sys_st = 64 -> R_C:(sys_st'=58) + (1-R_C):(sys_st'=ERROR_STATE);
111 [c4SeesObj1] sys_st = 64 -> (sys_st'= 65);
112 [endAction] sys_st = 65 -> (sys_st'= FINAL_STATE);
113 [c1StartSearchC3] sys_st = 65 -> R_C:(sys_st'=59) + (1-R_C):(sys_st'=ERROR_STATE);
114 [c2SeesObj1] sys_st = 65 -> (sys_st'= 66);
115 [endAction] sys_st = 66 -> (sys_st'= FINAL_STATE);
116 [c1StartSearchC3] sys_st = 66 -> R_C:(sys_st'=60) + (1-R_C):(sys_st'=ERROR_STATE);
117 [endAction] sys_st = 67 -> (sys_st'= FINAL_STATE);
118 [c2SeesObj1_c4AsksConfC2Obj1] sys_st = 67 -> R_C:(sys_st'=41) + (1-R_C):(sys_st'=
        ERROR_STATE);
119 [c1StartSearchC3] sys_st = 67 -> R_C:(sys_st'=63) + (1-R_C):(sys_st'=ERROR_STATE);
120 [c1StartSearchC2] sys_st = 67 -> R_C:(sys_st'=65) + (1-R_C):(sys_st'=ERROR_STATE);
121
122 endmodule
```

Listagem E.2: *Constrained Model* referente à Arquitetura 2 expresso na linguagem da PRISM