Pós-Graduação em Ciência da Computação

“Modelagem e Análise de Mecanismos de Tratamento de Interrupções em Infraestruturas Computacionais dos Sistemas Distribuídos”

Por

Ermeson Carneiro de Andrade

Tese de Doutorado

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

RECIFE, 02/2014

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ERMESON CARNEIRO DE ANDRADE

"Modelagem e Análise de Mecanismos de Tratamento de Interrupções em Infraestruturas Computacionais dos Sistemas Distribuídos"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Prof. Dr. Paulo Romero Martins Maciel

RECIFE, FEVEREIRO/2014

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Andrade, Ermeson Carneiro de
Modelagem e análise de mecanismos de tratamento de interrupções em infraestruturas computacionais dos sistemas distribuídos / Ermeson Carneiro de Andrade. - Recife: O Autor, 2014.
138 p., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Tese (doutorado) - Universidade Federal de Pernambuco. CIn, Ciência da Computação, 2014.

Inclui referências.

1. Ciência da Computação. 2. Engenharia de software. 3. Avaliação de desempenho. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004 CDD (23. ed.) MEI2014 – 040

Tese de Doutorado apresentada por **Ermeson Carneiro de Andrade** à Pós Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Modelagem e Análise de Mecanismos de Tratamento de Interrupções em Infraestruturas Computacionais dos Sistemas Distribuídos**" orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Paulo Roberto Freire Cunha
Centro de Informática / UFPE

_____

Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

_____

Prof. Eduardo Antônio Guimarães Tavares
Centro de Informática / UFPE

_____

Prof. Rivalino Matias Junior
Faculdade de Computação / UFU

_____

Prof. Ricardo José Paiva de Britto Salgueiro
Departamento de Ciência da Computação e Estatística/UFS

Visto e permitida a impressão.
Recife, 24 de fevereiro de 2014.

_____

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

AGRADECIMENTOS

- A Deus por ter conseguido vencer mais esta etapa na minha vida.
- Ao meu orientador, Prof. Paulo Maciel, pela paciência, suporte e conselhos durante todos esses anos. Sou muito grato por tudo. Agradeço também ao Prof. Kishor Trivedi, pela oportunidade de fazer doutorado sanduíche no laboratório dele, nos Estados Unidos.
- A todos os meus colegas e amigos que fiz durante minha estadia nos EUA: Rahul Gosh, Dong-Seong Kim, Fumio Machida, Xiaoyan Yin, Panayiotis Antoniou, Marek Pruszynski, Artak Tovmasyan, Jing Zhao, Eduardo Jardim, Chongyang Du e Chris Luwaga.
- A todos os meus amigos que fizeram parte dessa longa jornada. Dentre eles: Bruno Nogueira, Bruno Silva, Marcelo Alves, José Guimarães, Gabriel Alves, Gustavo Callou, Ivaldir Júnior, Suzana Sampaio, Julian Menezes, Sidharth Bahuguna, Rubens Matos e Kalil Bispo. Meu agradecimento em especial a Julian e a José, pelos momentos de descontração e por todo o apoio e ajuda.
- À minha família por todo suporte e amor durante todos esses anos. Também gostaria de agradecer, em especial, à minha namorada, Milrian Mendes, pelo apoio, paciência e amor nesse último ano de doutorado.
- Ao SOFTEX Recife, em especial, a todos que fazem parte do NEXT.

RESUMO

Os *softwares* possuem defeitos. Os computadores falham. Os vírus se propagam. Os prédios pegam fogo. As pessoas cometem erros. Os desastres acontecem. Embora gostaríamos que tais eventos nunca acontecessem, é prudente prover mecanismos que tratem as interrupções dos serviços, pois o custo do *downtime* pode ser extremamente elevado. Dessa forma, as empresas têm investido cada vez mais em mecanismos de tratamento de interrupções, visto que as interrupções, sejam elas brandas ou severas, podem acontecer em qualquer lugar, a qualquer momento, com pouco ou nenhum aviso. Os sistemas distribuídos (ex.: computação em nuvem) têm sido largamente usados nos últimos anos como um meio de recuperação de interrupções de modo que o sistema (infraestrutura e dados) é distribuído sobre um conjunto de servidores dispersos geograficamente e na ocorrência de interrupções, outros servidores assumirão a operacionalidade do sistema.

A forma mais amplamente usada para modelar sistemas complexos (ex.: sistemas distribuídos) é através das linguagens semiformais, tais como SysML (*System Modelling Language*) ou UML (*Unified Modeling Language*), devido principalmente à sua notação amigável e intuitiva. Os modelos semiformais gerados por essas linguagens, por si sós não fornecem suporte para avaliação de dependabilidade/desempenho das especificações dos sistemas. Dessa forma, faz-se necessário o mapeamento desses modelos semiformais para modelos analíticos, pois modelos analíticos são apoiados por fundamentos matemáticos sólidos, que suportam sua semântica precisa, estimulam a avaliação de desempenho/dependabilidade e fornecem suporte para verificações das propriedades qualitativas e análises. Os modelos analíticos, no entanto, não são intuitivos e requerem um considerável esforço por parte dos projetistas para entenderem a notação usada. Portanto, é sensato adotar o uso colaborativo dos modelos semiformais e analíticos.

Este trabalho propõe um *framework* baseado em métricas, modelos e ferramentas que auxilia os projetistas, os quais não possuem (ou possuem pouca) expertise em modelagem estocástica, a modelar e estudar os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos, a partir de especificações de alto nível, descritas através da SysML e MARTE. Para alcançar esse objetivo, propõe-se o mapeamento dos diagramas anotados da SysML em modelos DSPN (Rede de Petri Determinística e Estocástica). Essa abordagem resulta num modelo cujas propriedades são garantidas e provê um conjunto de indicadores de dependabilidade dos mecanismos e do sistema. Ademais, a fim de mostrar a corretude e aplicabilidade do *framework* proposto, estudos de casos são apresentados.

Palavras-chave: Modelagem de Dependabilidade. Sistemas Distribuídos. DSPN. MARTE. SysML.

ABSTRACT

Software has defects. Computer breaks. Viruses spread. Buildings catch fire. People make mistakes. Disasters strike. Much as we might prefer that such events never happen, it is prudent to provide mechanisms to address service outages because the cost of downtime can be extremely high. Thus, companies have increasingly invested in mechanisms for handling outages, since such events, whether soft or severe, can strike anywhere at anytime with little or no warning. Distributed systems (e.g. cloud computing) have been widely used in recent years as a method of recovery from outages, so that the system (infrastructure and data) is distributed over a set of geographically dispersed servers and in the occurrence of outages, other servers take over the operation of the system.

The most widely used way to model complex systems (e.g. distributed systems) is through semiformal language, such as SysML (*System Modelling Language*) or UML (*Unified Modeling Language*), chiefly due to their friendly and intuitive notations. The semiformal models generated by these languages, in themselves do not provide support for performance/dependability evaluation of the system specifications. Thus, it is necessary the mapping of these semiformal models into formal models. Because formal models are supported by sound mathematic foundations that uphold their precise semantics, foster performance evaluation and provide support to qualitative property verification and analysis. These models, however, quite often are not intuitive and may require considerable designer's effort to figure out the respective representation. Therefore, it is wise adopting a collaborative use of both semi-formal and formal models.

This paper proposes a framework based on metrics, models and tools that helps designers, who do not have (or have limited) expertise in stochastic modeling, to model and study mechanisms to handle outages and distributed system infrastructure from high-level specifications, described through SysML and MARTE. To achieve this goal, we propose the translate process of annotated SysML diagrams into DSPN (Deterministic and Stochastic Petri Net) models. This approach results in a model whose properties are guaranteed and which is able to provides a set of dependability indicators for the mechanisms and system. Besides, in order to show the correctness and applicability of the proposed framework, case studies are shown.

Keywords: Dependability Modeling. Distributed Systems. DSPN. MARTE. Modeling. SysML.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ADS - *Avaliação de Desempenho de Sistemas.*

CPN - *Coloured Petri Net.*

DSPN - *Redes de Petri Determinísticas e Estocásticas.*

GSPN - *Generalized Stochastic Petri Net.*

MARTE - *Profile for Modeling and Analysis of Real-time and Embedded systems.*

MTTF - *Tempo Médio Até a Falha.*

MTTR - *Tempo Médio de Reparo.*

LB - Balanceador de Cargas.

OMG - *Object Management Group.*

RdP - Redes de Petri.

SD - *Sistemas Distribuídos.*

SLA - *Service Level Agreement.*

SPN - *Stochastic Petri Nets.*

SysML - *System Modelling Language.*

SyML-STM - Diagrama de Estados da SysML.

SyML-IBD - Diagrama de Bloco Interno da SysML.

SyML-AD - Diagrama de Atividades da SysML.

TimeNET - *Timed Net Evalution.*

TPN - Redes de Petri Temporizadas.

UML - *Unified Model Language.*

RTO - *Recovery Time Objective.*

RPO - *Recovery Point Objective.*

VM - Máquina Virtual.

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Never give in, never give in, never; never; never; never - in nothing, great or small, large or petty - never give in except to convictions of honor and good sense.

—WINSTON CHURCHILL

Os *softwares* possuem defeitos. Os *hardwares* falham. Os vírus se propagam. Os prédios pegam fogo. As pessoas cometem erros. Os desastres acontecem. Embora gostaríamos que tais eventos nunca acontecessem, é prudente prover mecanismos que tratem as interrupções dos serviços, pois o custo do *downtime* pode ser significativo. Uma pesquisa realizada pela Emerson Network Power em 2011 [Pre11] revelou que o custo do *downtime* não planejado em empresas de TI pode chegar a $5.000,00 por minuto. Vale ser ressaltado que esse custo pode ser ainda maior caso o *downtime* resulte em perda de dados. Dessa forma, um grande desafio atual é construir sistemas que sejam capazes de garantir alta disponibilidade dos serviços, mesmo diante de interrupções brandas (ex.: interrupção de energia, falha de disco, falha de software, entre outras) e severas, tais como *tsunami*, furacão, terremoto etc [LML+06].

Nos últimos anos, os Sistemas Distribuídos (SD) têm crescido de forma significativa em funcionalidade, escala e complexidade [GYJ+10]. Exemplos desses sistemas são computação em nuvem [AFG+10], computação em *clusters* [Buy99] e sistemas virtualizados [KMT09]. Embora esse crescimento acelerado tenha permitido os SD fornecerem um grande conjunto de serviços, bem como servido a um número cada vez maior de usuários, as interrupções nos serviços se tornaram inevitáveis devido à alta complexidade de desenvolvimento dos mesmos e a existência de componentes fortemente acoplados. Esses sistemas, no entanto, almejam fornecer um alto grau de disponibilidade dos serviços críticos, mesmo diante de interrupções. Isto é, caso o sistema ou módulo se torne indisponível devido a uma interrupção (branda ou severa), o sistema/módulo deve ser capaz de tratar essa interrupção e degradar de forma graciosa [Gär99]. Assim, a análise de mecanismos de tratamento de interrupções é extremamente importante, pois pode prover medidas preventivas, estratégias de recuperação e considerações técnicas a fim de garantir os *Service Level Agreements* (SLAs) [LKD+03] dos clientes, bem como evitar o *downtime* e melhorar a disponibilidade dos SD.

Modelos analíticos, tais como cadeias de Markov [Tri82], redes de Petri [Mur89] ou teoria das filas [BGdMT06], têm sido largamente usados na modelagem e análise dos mais variados tipos de sistemas complexos, tais como: sistemas embarcados de tempo-real [AMN+10], *data centers* [CMTA12], sistemas virtualizados [MKT10], entre outros. Esses

modelos analíticos, por sua vez, são apoiados por fundamentos matemáticos sólidos, que suportam sua semântica precisa, estimulam a avaliação de desempenho/dependabilidade e fornecem suporte para verificações das propriedades qualitativas e análises [And09]. Além disso, é importante salientar que os modelos analíticos podem ser usados para: comparar alternativas de projetos enquanto os sistemas estão sendo construídos, ajustar parâmetros dos sistemas existentes, realizar planejamento de capacidade dos sistemas e prover insumos para a tomada de decisão.

As redes de Petri (RdP) têm se evidenciado como uma ferramenta gráfica e matemática eficiente, na modelagem e análise de sistemas complexos [Mur89, MLC96, Rei85, Pet77]. Elas têm a capacidade de modelar com facilidade sistemas onde há a ocorrência de paralelismo, sequenciamento, escolha não-determinística, sincronismo, compartilhamento de recursos, concorrência e exclusão mútua. Neste trabalho, propõe-se a utilização de uma extensão das redes de Petri, denominada de Redes de Petri Determinísticas e Estocásticas (DSPN) [MC87], como ferramenta de suporte a análise das infraestruturas distribuídas e dos mecanismos de tratamento de interrupções. Vários motivos levaram a adoção das DSPN neste trabalho. As DSPN podem ser adotadas para análise de desempenho, confiança e disponibilidade de sistemas dinâmicos a eventos discretos. Esse modelo analítico também permite que as métricas sejam aferidas tanto via simulação quanto pela análise do espaço de estados. Além disso, as DSPN proveem diversos recursos para modelagem, tais como funções de guarda, pesos, prioridade, taxas dependentes de marcação, entre outros.

Os modelos analíticos (como as RdP) não são tão facilmente obtidos por projetistas (ou administradores) de sistemas que não possuem expertise em modelagem estocástica. Um aspecto desafiador consiste justamente em permitir que tais profissionais sejam capazes de criar modelos analíticos que representem as infraestruturas dos sistemas distribuídos e os mecanismos de tratamento de interrupções a partir do conhecimento deles [AMKT11]. Dessa forma, se esses modelos puderem ser gerados através do conhecimento dos projetistas, os mesmos podem fornecer um conjunto de informações relevantes na tomada de decisão de implementar ou não uma determinada solução de recuperação/mitigação de interrupções que garanta alta disponibilidade e continuidade dos serviços para os SD mesmo em caso de falhas.

As linguagens semiformais, tais como UML [Gue08], SysML [H$^+$06] ou OMT [Der95] têm sido bastante adotadas no ciclo de desenvolvimento de sistemas complexos (ex.: sistemas distribuídos), devido, principalmente, à sua notação amigável e intuitiva [AMN$^+$10]. Essas linguagens consistem de um certo número de elementos gráficos que se combinam para formar diagramas, os quais são usadas para visualizar, especificar, construir e documentar artefatos de sistemas complexos. A modelagem dos sistemas usando as linguagens semiformais é importante por diversas razões, entre elas [And09]: (i) ajudam a visualizar o sistema como um todo, (ii) permitem modelar a estrutura ou o comportamento de um sistema e (iii) proporcionam um guia para a construção/entendimento de um sistema.

Dentre as linguagens semiformais mais utilizadas na indústria e na acadêmia destaca-se a SysML (*System Modelling Language*) [H$^+$06]. Essa linguagem suporta especificação,

análise, desenho e verificação de uma grande variedade de sistemas complexos. Esses sistemas podem incluir *hardware*, *software*, informações, métodos, pessoas e instrumentos. SysML estende UML (*Unified Model Language*) 2.0 [Uml04], sendo SysML usada para modelar sistemas que não são totalmente baseados em *software*. No entanto, SysML não possui suporte para notações quantitativas. Tais notações são extremamente importantes quando se está almejando realizar análises de sistemas complexos. Para isso, um *profile* da UML denominado de MARTE (*Modeling and Analysis of Real-time Embedded Systems*) [FBDSG07] foi especificado pela OMG. MARTE tem o objetivo de substituir o UML SPT (*Profile for Schedulability, Performance and Time*) [OMG03], sendo ele usado para auxiliar na construção de modelos que possam ser usados para fazer precisões quantitativas relativas às características dos sistemas, tais como desempenho, disponibilidade, confiabilidade etc.

Muito embora as linguagens semiformais sejam amigáveis e de fácil utilização, elas não fornecem suporte para a avaliação de dependabilidade/desempenho das especificações dos SD. Assim, é sensato adotar o uso colaborativo de modelos semiformais e analíticos, já que a informalidade das linguagens semiformais é superada pelo rigor do formalismo dos modelos analíticos [AMCN09]. Nesse sentido, este trabalho almeja desenvolver um *framework* que propõe o mapeamento dos diagramas da SysML, anotados de acordo com o *profile* MARTE, em modelos analíticos descritos em DSPN. Esses modelos, por sua vez, são usados para o cálculo de métricas relacionadas a disponibilidade, tais como disponibilidade estacionária, *downtime*, tarefas perdidas, entre outras.

1.1 MOTIVAÇÃO

Nos últimos anos, vários eventos catastróficos aconteceram ao redor do mundo, mostrando a necessidade de mecanismos de recuperação e mitigação de interrupções. Em setembro de 2001 [ST01], um ataque terrorista destruiu as torres gêmeas do *World Trade Center* nos USA. Em dezembro de 2004 [IAK+07], um terremoto de magnitude 9.15 no Oceano Índico, causou a morte de cerca de 226.000 pessoas na Indonésia, Sri Lanka, Índia, Tailândia e outros nove países. Em março de 2011 [Bla11], um terremoto de magnitude de 8.9 devastou várias cidades no Japão. Esse terremoto também resultou na explosão da usina nuclear de Fukushima. Em novembro de 2012 [KF02], um furação violento destruiu parte da cidade de Nova York, nos USA. Em junho de 2013 [Bha13], inundações devastaram o norte da Índia, resultando na morte de mais de 800 pessoas. Em abril de 2012 [War12], o Gmail ficou fora do ar, mundialmente, por mais de 45 minutos, afetando 350 milhões de usuários. Essa interrupção foi devido a uma atualização no serviço para uma tecnologia *open-source* chamada de OpenFlow. Em junho de 2012 [Kos12], o *Netflix*, o *Instagram* e o *Pinterest* ficaram todos fora do ar devido a uma tempestade que aconteceu no norte da Virgínia, nos USA. Em maio de 2013 [Dro13], o Dropbox sofreu um ataque de negação de serviço, resultando na indisponibilidade de seus serviços mundialmente. Note que os eventos citados acima são apenas alguns poucos exemplos de interrupções que aconteceram nos últimos anos, mostrando que as companhias, independente do porte, estão susceptíveis a eles.

Nossa sociedade tem se tornando tão dependente do bom funcionamento dos SD que o *downtime*, mesmo que por um curto período tempo, pode resultar em efeitos catastróficos para as empresas [WCR+10]. Exemplos incluem: (i) diminuir a credibilidade da empresa no mercado, (ii) arcar com penalidades devido às violações de SLAs, (iii) assumir custos relacionados à recuperação e reparos devido à perda de dados e (iv) perda de receita devido à incapacidade de fazer negócios com os clientes. Segundo [VMw12], 40% das empresas que passaram por um desastre nunca conseguiram reabrir novamente. Das empresas que conseguiram retomar os negócios, 25% fecharam em dois anos. Portanto, através da análise dos mecanismos de mitigação e recuperação de interrupções nos SD, os projetistas podem encontrar o ponto no qual o investimento financeiro em tais mecanismos não superam o custo das interrupções.

Apesar dos diagramas da SysML tenham sido utilizados para modelar sistemas complexos, eles não fornecem suporte para avaliação de dependabilidade/desempenho das especificações dos sistemas distribuídos devido a sua natureza semiformal [YAM+12]. Isto é, esses diagramas não possuem uma semântica totalmente definida, consequentemente, não é possível aplicar algoritmos de análise diretamente sobre os mesmos. Assim, faz-se necessário o mapeamento desses diagramas para modelos analíticos, já que os modelos analíticos possuem fundamentos matemáticos sólidos e uma semântica precisa, permitindo a aplicação de técnicas de análises sobre os modelos [AMF+10]. Entre os benefícios dos modelos analíticos, é possível destacar os seguintes [And09]:

- Permitir a redução da dependência da intuição e do julgamento humano.
- Possuir grande poder de abstração.
- Permitir a análise do modelo através de simulações.
- Possibilitar a detecção de inconsistências e ambiguidades na especificação decorrentes da linguagem natural.
- Permitir a prototipagem dos sistemas críticos.
- Constituir um mecanismo rigoroso e efetivo para modelagem, síntese e análise de sistemas.

No âmbito dos modelos analíticos, as redes de Petri costumam ser bastante utilizadas para a modelagem e a especificação de sistemas computacionais. As RdP foram propostas por Carl Adam Petri na sua tese de doutoramento em 1962 [Pet66]. Desde então, elas vêm sendo aplicadas nas mais diversas áreas, indo desde a Ciência da Computação, passando pela Administração de Empresas e indo até áreas da Biologia. Desde seu surgimento, diversas extensões foram propostas para esse formalismo. Entre as extensões, estão as DSPN [MC87], que são adotadas nesta pesquisa. Através dos modelos DSPN, pode-se obter um conjunto de métricas importante para o estudo das infraestruturas distribuídas e dos mecanismos de recuperação de interrupção, tais como disponibilidade e *downtime* do sistema, quantidade de tarefas perdidas durante as interrupções, custo da interrupção

etc. Esses tipos de métricas são muito importantes, pois podem ajudar os projetistas na escolha da melhor solução, levando em consideração um conjunto de atributos (ex.: custo, tarefas perdidas, *downtime*, entre outros). Ademais, respeitando algumas propriedades necessárias (ex.: conservação e repetitividade [Mur89]), os resultados das métricas podem ser obtidos de forma analítica, ou ainda pela simulação do modelo.

1.2 OBJETIVOS

Devido aos eventos catastróficos que aconteceram nos últimos anos, as empresas perceberam a importância de estudar os mecanismos de tratamento de interrupções nos SD, visto que as interrupções, sejam elas brandas ou severas, podem acontecer em qualquer lugar, a qualquer momento, com pouco ou nenhum aviso. Nesse contexto, este trabalho tem o objetivo de desenvolver um *framework*, baseado em métricas, modelos e ferramentas, que auxilie os projetistas, os quais não possuem (ou possuem pouca) expertise em modelagem estocástica, a estudarem os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos, a partir de modelos de alto nível descritos em SysML, a fim de garantir a alta disponibilidade dos SD.

A integração dos modelos formais e semiformais não é uma tarefa trivial. Ela requer um conjunto de passos, indo desde o projeto do problema de interesse, usando os digramas da SysML e as anotação de MARTE, passando pelo mapeamento de tais diagramas em modelos DSPN, até a análise dos modelos obtidos. O *framework* proposto suporta tal integração, de modo que os diagramas anotados da SysML são mapeamento em componentes do modelo de disponibilidade descritos em DSPN. Esses componentes, por sua vez, são sincronizados e combinados para formar um modelo de disponibilidade completo que representa todo o sistema modelado. Contudo, os modelos DSPN gerados pelo processo de mapeamento deverão possuir algumas propriedades estruturais das redes de Petri, como conservação e repetividade, a fim de permitir a computação de métricas (ex.: disponibilidade ou *downtime*) a partir do regime transiente ou do regime estacionário.

Apesar dos modelos DSPN serem obtidos através de um processo de mapeamento pré-definido, esses modelos gerados podem ser grandes e complexos. Dessa forma, uma abordagem hierárquica é proposta para aliviar tais problemas tanto com relação aos diagramas da SysML (ex.: estruturas compostas) quanto com relação as redes de Petri (ex.: modelos hierárquicos). Além disso, com o intuito de comprovar a abordagem proposta, foi criado um protótipo de uma ferramenta que suporta o *framework* proposto. Essa ferramenta permite que os usuários possam aferir as métricas de dependabilidade a partir de um conjunto bem definido de parâmetros de entrada dos modelos. Adicionalmente, os resultados dos modelos DSPN gerados através do processo de mapeamento dos diagramas anotados da SysML serão comparados com os resultados de experimentos montados e executados em infraestruturas reais dos sistemas distribuídos. Tal comparação almeja demonstrar que os modelos obtidos pelo *framework* representam de fato o ambiente real.

Com o *framework* proposto, espera-se otimizar o tempo e o esforço da análise dos mecanismos de recuperação e mitigação das interrupções implantados nos SD. Adicional-

mente, os projetistas podem utilizar o *framework* para realizar análises de cenários de consequência de acordo com alguma ação que eles venham a realizar. Isto é, são cenários do tipo "e se" (*what if*), que permitem os projetistas entendam as consequências de possíveis atos, e decidam ou não por eles. Os resultados obtidos dessas análises também podem ser utilizados no processo de tomada de decisão da alta gerência, a fim de maximizar a utilização dos recursos ao menor custo possível.

Mais especificamente, os objetivos deste trabalho são:

- Desenvolver um *framework* que axilie os projetistas a estudarem os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos.
- Desenvolver um método de mapeamento dos diagramas da SysML em modelos DSPN para análise de disponibilidade.
- Definir uma estratégia de modelagem e avaliação hierárquica que promova escalabilidade.
- Desenvolver uma ferramenta que suporte o *framework* proposto.

Partes dos resultados desta pesquisa foram apresentadas nas seguintes publicações [AMF+10, AMN+10, TAM12, TGA10, AMKT11, MAKT11, AAM+13].

1.3 ESTRUTURA DO TRABALHO

O Capítulo 2 introduz os conceitos fundamentais a serem utilizados nesta pesquisa, tais como sistemas distribuídos, mecanismos de tratamento de interrupções, SysML, MARTE, RBD e Redes de Petri. O Capítulo 3 detalha os trabalhos relacionados. O Capítulo 4 descreve o *framework* proposto. O Capítulo 5 detalha o protótipo da ferramenta desenvolvida. O Capítulo 6 apresenta estudos de casos nos quais foi aplicado o *framework* proposto. Finalmente, o Capítulo 7 conclui o trabalho, lista as contribuições, detalha as limitações e apresenta os trabalhos futuros.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

The greater danger for most of us lies not in setting our aim too high and falling short, but in setting our aim too low and achieving our mark.

—MICHELANGELO

O trabalho desenvolvido nesta pesquisa envolve várias áreas do conhecimento, indo desde a especificação de alto nível dos sistemas distribuídos e seus mecanismos de tratamento de interrupções, passando pelo mapeamento dessas especificações, até a análise do modelo analítico obtido. Dessa forma, este capítulo apresenta os principais conceitos utilizados nesta tese para um melhor entendimento do trabalho proposto. Primeiramente, os sistemas distribuídos são introduzidos, destacando os principais mecanismos de recuperação/mitigação de interrupções. Em seguida, os diagramas comportamentais e estruturais da SysML usados nesta pesquisa são apresentados. Após isso, o *profile* da UML MARTE também é introduzido. Por fim, são abordados os principais conceitos que envolvem os modelos analíticos, destacando as redes de Petri.

2.1 SISTEMAS DISTRIBUÍDOS

Atualmente, os sistemas aplicativos estão necessitando cada vez mais de poder computacional que vão além do que a computação sequencial de Von Neumann pode proporcionar. Uma maneira de endereçar essa questão é melhorar tanto a velocidade das operações dos processadores, quanto os outros componentes dos computadores de modo que possam oferecer um poder computacional necessário para aplicações que exijam computação intensiva, i.e., meteorologia, oceanografia e genoma. Muito embora tal computação seja possível atualmente, até certo ponto, avanços futuros são limitados pela velocidade da luz, leis da termodinâmica, e altos custos financeiros para a fabricação dos processadores. Uma solução viável consiste em combinar o poder de processamento de vários computadores e coordenar seus esforços computacionais. O sistema resultante dessa combinação é popularmente conhecido como sistema distribuído [BB99].

Segundo Andrew Tanenbaum [TVS02] , um sistema distribuído é uma "*coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente*". Os computadores que fazem parte desse sistema podem estar fisicamente próximos, conectados por uma LAN (*Local Area Network*), ou eles podem estar geograficamente dispersos, conectados através de uma MAN (*Metropolitan Area Network*) ou WAN (*Wide Area Network*). Vale ser ressaltado que tais sistemas podem ser compostos por configurações variadas, incluindo *mainframes*, computadores pessoais, estações de trabalho, minicomputadores, entre outros.

Os sistemas distribuídos podem ser classificados de acordo com as suas características e funcionalidades, tais como: computação em *cluster*, computação em grade, computação em nuvem e sistemas virtualizados. Um *cluster* consiste de estações de trabalho de alto desempenho ou PCs interconectados por uma rede de alta velocidade [BB99]. Ele funciona como um conjunto integrado de recursos, criando a impressão de recurso único. Já as grades, segundo Ian Foster [FKT03], podem ser definidas como sendo "*sistemas que dão suporte à criação de aplicações paralelas que agregam e fornecem recursos heterogêneos distribuídos geograficamente de forma consistente e barata*". A computação em grade surge como sendo uma alternativa para a computação em *cluster*, visto que ela permite casar tecnologias heterogêneas e, muitas vezes, geograficamente dispersas.

Segundo o Instituto Nacional de Padrões e Tecnologia dos Estados Unidos da America (NIST) [MG11], a computação em nuvem "*é um modelo para acesso conveniente, sob demanda e de qualquer localização, a uma rede compartilhada de recursos de computação (ex.: redes, servidores, armazenamento, aplicativos e serviços) que possam ser prontamente disponibilizados e liberados com um esforço mínimo de gestão ou de interação com o provedor de serviços*". A definição de computação em nuvem como um modelo computacional de utilidade para serviços computacionais é semelhante ao modelo de utilidade implantado para a eletricidade, a água e os serviços de telecomunicação. Isto é, os usuários podem consumir a quantidade de serviços que eles desejarem, sempre que quiserem, e serão cobrados apenas pelos recursos utilizados [BA12].

O NIST define três modelos de serviços para a computação em nuvem: *software* como serviço (SaaS), plataforma como serviço (PaaS) e infraestrutura como serviço (IaaS).

- ***Software* como Serviço.** "A capacidade fornecida ao consumidor destina-se à utilização dos aplicativos do provedor rodando em uma infraestrutura de nuvem. As aplicações são acessíveis a partir de diversos dispositivos clientes, quer através de uma interface "leve"(*thin*), como um navegador *Web* (por exemplo, *webmail*), ou uma interface de programa. O consumidor não administra e nem controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento, ou mesmo capacidades de aplicativos individuais, com a possível exceção de configurações limitadas do aplicativo, específicas do usuário."

- **Plataforma como Serviço.** "A capacidade fornecida ao consumidor destina-se à infraestrutura criada ou comprada pelo consumidor para a nuvem, criada usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor não administra e nem controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre os aplicativos implementados e possivelmente sobre as configurações para o ambiente de hospedagem de aplicativos."

- **Infraestrutura como serviço.** "A capacidade fornecida ao consumidor destina-se ao provisionamento de processamento, armazenamento, redes e outros recursos de computação fundamentais em que o consumidor é capaz de implementar e executar

softwares arbitrários, que podem incluir sistemas operacionais e aplicativos. O consumidor não administra e nem controla a infraestrutura de nuvem subjacente, mas tem controle sobre sistemas operacionais, armazenamento e aplicativos implementados e, possivelmente, um controle limitado de componentes de rede selecionados (por exemplo, *firewalls* do *host*)."

O NIST reconhece quatro modelos de implantação de computação em nuvem:

- **Nuvem privada.** "A infraestrutura de nuvem é provisionada para uso exclusivo por uma única organização, compreendendo múltiplos consumidores (por exemplo, unidades de negócio). Ela pode ser controlada, gerenciada e operada pela organização, um terceiro, ou alguma combinação deles, e pode existir com ou sem premissas."
- **Nuvem comunitária.** "A infraestrutura de nuvem é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que têm preocupações comuns (por exemplo, considerações referentes à missão, requisitos de segurança, política e compliance). Ela pode ser controlada, gerenciada e operada por uma ou mais das organizações na comunidade, um terceiro, ou alguma combinação deles, e pode existir com ou sem premissas."
- **Nuvem pública.** "A infraestrutura de nuvem é provisionada para uso aberto ao público em geral. Ela pode ser controlada, gerenciada e operada por organização empresarial, acadêmica ou governamental, ou alguma combinação delas. Ela existe sob as premissas do fornecedor da nuvem."
- **Nuvem híbrida.** "A infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem como entidades únicas, mas são unidas por tecnologia padronizada ou proprietária que permita a portabilidade de dados e aplicativos (por exemplo, balanceamento de carga entre nuvens)."

Um dos componentes chave da computação nas nuvens é a virtualização. A virtualização consiste no processo de criar uma ou várias versões virtuais de dispositivos ou recursos de infraestrutura, i.e., ambientes de computação, sistemas operacionais, dispositivos de armazenamento ou componentes de rede, ao invés de criar versões físicas dos mesmos recursos [CA11]. Em outras palavras, a virtualização é uma técnica que permite a execução de mais de um servidor (ou outros componentes de infraestrutura) no mesmo *hardware*. Ela foi inicialmente desenvolvida para melhorar a utilização de recursos dos *mainframes* e evoluiu para se tornar uma das características mais comuns de computação em nuvem [BA12]. Note que a virtualização é uma das tecnologias de computação em nuvem. No entanto, a virtualização não é sinônimo de computação em nuvem. A computação em nuvem permite às empresas armazenar e acessar aplicações e dados em servidores virtuais, ao invés de usar *hardware* físico como exige a virtualização.

Os serviços oferecidos pela computação em nuvem (ex.: *pay-as-you-go* e auto *scaling*) são também um meio econômico de gerenciamento de dados/sistemas, já que não requerem a compra ou manutenção de *hardware* ou *software*. Além disso, vale ser ressaltado que a virtualização ocorre em um ambiente local, enquanto os serviços de computação em nuvem podem estar dispersos geograficamente pela *Web*.

2.2 MECANISMOS DE TRATAMENTO DE INTERRUPÇÕES

Os sistemas distribuídos são vulneráveis a um conjunto de interrupções brandas (ex.: interrupção de energia, falha de discos etc.) e severas (ex.: incêndio, terremoto, entre outros) [S+10]. Algumas dessas vulnerabilidades podem ser eliminadas, ou pelo menos minimizadas, através das estratégias de garantia de qualidade, i.e., testes, revisões etc. Porém, é impossível eliminar todos os riscos. Os mecanismos de tratamento de interrupções são projetados justamente para mitigar/contigenciar esses problemas, provendo soluções apropriadas para evitar perda de dados e/ou diminuir o tempo para a recuperação dos serviços depois de uma interrupção. Note que os mecanismos de contingência consistem em ações para lidar com os eventos de interrupções, caso eles se concretizem, enquanto os mecanismos de mitigação significam as ações utilizadas para reduzir a probabilidade de ocorrência dos mesmos. Dentre as causas mais comuns de interrupções brandas, as falhas de *hardware* e *software* se destacam. Enquanto as falhas de *hardware* têm sido extensivamente estudadas na literatura e seus mecanismos de mitigação/contigência são conhecidos, as falhas de *software* e seus mecanismos de tratamento de interrupções têm sido pouco endereçadas [TGA10]. A seguir, são discutidos as falhas de *software* e seus respectivos mecanismos de tratamento de interrupções, bem como as soluções de recuperação de desastres.

2.2.1 Falhas de Software

As falhas de *software* é uma das principais causas de interrupções nos serviços dos sistemas, sendo elas ativadas por duas classes de defeitos (do inglês *bugs*), conhecidos como *Bohrbugs* e *Mandelbugs* [GT05, GT07]. Os *Bohrbugs* são, em princípio, fáceis de isolar e se manifestam consistentemente sob condições bem definidas, enquanto os *Mandelbugs* possuem um comportamento caótico ou até mesmo não determinístico, pois a ativação do defeito e/ou propagação dos erros são complexos. Um subtipo dos *Mandelbugs* possui a característica de aumentar a taxa de falha e/ou degradar a performance do sistema ao longo do tempo. Esse tipo de *bug* tem sido observado em vários sistemas e são chamados de *bugs* de envelhecimento (*aging bugs*) [AW97, GT07]. Este trabalho, mais especificamente, tem o objetivo de estudar os mecanismos de mitigação dos *bugs* de envelhecimento.

O mecanismo de tratamento de interrupção utilizado, devido ao envelhecimento de *software*, é denominados de rejuvenescimento [HKKF95, GT07]. O rejuvenescimento de *software* é uma técnica proativa (mecanismo de mitigação) de sistema de *software* usada justamente para lidar com esse fenômeno de envelhecimento. Essa técnica evita

a ocorrência de falhas futuras, já que, periodicamente, finaliza o sistema para imediatamente reiniciá-lo. Isto é, o rejuvenescimento visando impedir a ocorrência de uma falha em detrimento do acúmulo de defeitos no sistema. Dentre as técnicas de rejuvenescimento, as mais utilizadas são [GMT08, GT07, TGA10]: o rejuvenescimento baseado em tempo e o rejuvenescimento baseado em condição. No rejuvenescimento baseado em tempo, o sistema é rejuvenescido em intervalos de tempo fixos, enquanto no rejuvenescimento baseado em condição, o sistema é rejuvenescido quando o mesmo satisfaz uma condição específica.

A disponibilidade do sistema pode ser influenciada pelo o rejuvenescimento, já que na ocorrência do mesmo o sistema pode ficar indisponível. No entanto, o custo de tal indisponibilidade planejada tende a ser menor do que se uma interrupção não planejada ocorresse e causasse um maior período de indisponibilidade. Note que o custo de interrupções não planejadas (ex.: corrompimento de banco de dados e falha humana) é muito maior do que o custo de interrupções planejadas (ex.: rejuvenescimento e *upgrade* de *software*). As interrupções não planejadas podem durar mais tempo, resultando em perda de receita e insatisfação dos clientes. Por outro lado, as interrupções planejadas podem ser realizadas em horários de menor demanda de modo que, muitas vezes, se torna imperceptível para os usuários. Todavia, em caso de desastres, os mecanismos apresentados anteriormente podem não ser suficientes para garantir alta disponibilidade e continuidade dos serviços oferecidos pelos SD, já que o desastre pode resultar na destruição de toda a infraestrutura, incluíndo as instâncias redundantes.

2.2.2 Recuperação de Desastres

Sistemas de alta disponibilidade são projetados e implementados para resistir às mais diversas interrupções. No entanto, em caso de interrupções severas (ex.: atentados terroristas, incêndio, tsunami e vandalismo), que impactam o *data center* primário, bem como as instâncias redundantes, essas infraestruturas podem não resistir e resultar na indisponibilidade das mesmas. Visto que esses eventos podem prejudicar drasticamente as soluções de alta disponibilidade, os mecanismos de recuperação de desastres são implantados para proteger os serviços críticos. Esses mecanismos, geralmente, apoiam-se em instalações remotas de *data centers* redundantes e em mecanismos de *failover* que prontamente restabelecem os serviços críticos em outra localidade (site remotos/secundários) após o desastre [BA12].

Cada dia mais as empresas adotam a computação em nuvem como um mecanismo de recuperação de desastres de modo que o sistema (infraestrutura e dados) é distribuído sobre um conjunto de servidores dispersos geograficamente e na ocorrência de interrupções, outros servidores assumirão a operacionalidade do sistema. Além disso, a computação em nuvem para sistemas críticos requer uma disponibilidade de 24 horas por dia, sete dias por semana e não tolera interrupções que durem mais do que poucas horas, visto que as mesmas podem resultar em um enorme prejuízo financeiro ou colocar a empresa para fora do negócio. Dessa forma, com o intuito de garantir alta disponibilidade de seus

serviços, a computação em nuvem provê um conjunto de mecanismos que são utilizados para tolerar e tratar as interrupções (severas e brandas). Esses mecanismos são discutidos a seguir [Tea13a].

- **Zonas de Isolamento.** Os usuários de serviços de computação em nuvem podem facilmente especificar a localização onde as instâncias de uma aplicação serão executadas. Cada localização, chamada de zona de isolamento, é isolada de falhas de outras zonas. Essas zonas permitem aos usuários montar configurações tolerantes a interrupções através da distribuição das instâncias por diversas zonas.
- **Função de Auto *Scaling*.** Essa função permite criar ou destruir instâncias de uma aplicação, de acordo com as regras definidas pelos os usuários. Isto é, esses usuários podem determinar a quantidade mínima de instâncias rodando, mesmo diante de interrupções severas.
- **Balanceador de Carga.** O balanceador de carga distribui as requisições entre as instâncias de uma aplicação. Essas instâncias, por sua vez, podem estar sendo executadas a partir de diferentes zonas de isolamento. A configuração do balanceador de carga pode mudar drasticamente, dependendo do comportamento da função de auto *scaling*.

Desde o surgimento dos provedores de serviços em nuvens, tais como *Amazon Web Services*, *Google* e *Rackspace*, a utilização de serviços em nuvens têm crescido de forma significativa nos últimos anos. No entanto, o problema é que alguns provedores de tais serviços não oferecem a segurança e a privacidade de informações que muitas corporações exigem. Por isso, apesar de todas as suas vantagens, a computação em nuvem fornecida por terceiros ainda é vista com cautela, ou até desconfiança, por diversas empresas. Dessa forma, algumas empresas têm adotado *softwares open-source* que provêem a criação de nuvens para o tratamento de interrupções (ex.: Eucalyptus [NWG+09], Hadoop [SKRC10], OpenCirrus [CGH+09], Reservoir [RBL+09] etc.), já que elas podem ser vistas como instâncias modernas de *hot sites* para a recuperação de interrupções severas. Além disso, essas nuvens *open-source* possuem um conjunto de benefícios, tais como custo, personalização, transparência e desenvolvimento colaborativo. Por outro lado, da mesma forma que as nuvens *open-sources* possuem um conjunto de vantagens, elas possuem desvantagens. Muito embora a ideia de ter uma opção livre de licença ou de custo baixo possa ser atraente para os departamentos de Tecnologia da Informação (TI) das empresas, a natureza "faça você mesmo"(*do-it-yourself*) das opções de código aberto pode ser uma grande desvantagem para algumas empresas, visto que essas opções, geralmente, não possuem (ou possuem pouco) suporte para os usuários. Adicionalmente, as plataformas *open-source* não garantem integração e portabilidade entre os ambientes em nuvem.

A virtualização também tem sido bastante utilizada como um mecanismo *ad-hoc* de recuperação de interrupções severas, pois, geralmente, envolve a replicação do site primário

(infraestrutura e dados) para um site remoto/secundário. Isso permite que as funcionalidades do sistema, assim como os dados estejam sempre acessíveis, mesmo se o site primário estiver indisponível. Vale ser ressaltado que numa infraestrutura virtual, os servidores virtuais podem ser facilmente iniciados de modo reproduzível e confiável, uma vez que intervenções manuais podem não ser necessárias. Ademais, a virtualização abstrai as dependências de *hardware* e *software* [VMw12].

Geralmente, o planejamento de recuperação de desastre foca em dois objetivos principais: RTO (*Recovery Time Objective*) e RPO (*Recovery Point Objective*). O RTO compreende o tempo máximo para trazer um sistema ou aplicativo ao seu estado operacional, enquanto o RPO compreende a quantidade máxima de dados que podem ser perdidos com o processo de recuperação. A Figura 2.1 ilustra esses objetivos no contexto do acontecimento de uma interrupção severa. O sistema está operando normalmente, quando um desastre ocorre (furacão, terremoto etc.), resultando na indisponibilidade do serviço. Se o dano causado pelo desastre for pequeno, então, a empresa poderá optar por recuperar o serviço no próprio *data center* impactado. Por outro lado, se o dano causado for catastrófico, então, a empresa irá declarar um desastre e ativará o plano de recuperação de desastres. O plano de recuperação de desastres, tipicamente, envolve a recuperação dos serviços em um *data center* remoto que não foi impactado pelo evento.

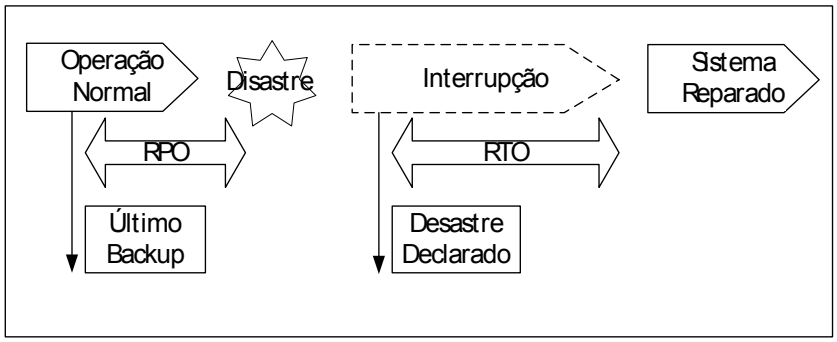


Figura 2.1: Objetivos da Recuperação de Desastres.

Para avaliar esses mecanismos quantitativamente, modelos analíticos que capturam as configurações dos sistemas (ex.: componentes redundantes e zonas de isolamento), como também os mecanismos de tratamento de interrupções (ex.: função de auto *scaling* e *failover*) são requeridos. Um aspecto desafiador consiste justamente em permitir que os projetistas (ou administradores de sistemas), os quais não possuem (ou possuem pouca) expertise em modelagem estocástica, sejam capazes de estudar essas infraestruturas e mecanismos a partir do conhecimento deles. Nesse sentido, este trabalho propõe um *framework* para auxiliar os projetistas nessa atividade, em que as infraestruturas distribuídas e os mecanismos de tratamento de interrupção são modelados, a partir de uma linguagem de alto nível, descritas em SysML e MARTE.

2.3 SYSML

A SysML é uma linguagem de modelagem de propósito geral para aplicações em engenharia de sistemas. Ela suporta a especificação, análise, desenho e verificação de sistemas complexos o que pode incluir *hardware*, *software*, dados, pessoal, processos e infraestrutura [H$^+$06, AMC$^+$10, AMF$^+$10]. SysML é uma linguagem gráfica de modelagem que fornece semântica (significado) e notação (representação do significado) para modelagem de sistemas complexos. Porém, SysML não é uma metodologia ou uma ferramenta de desenvolvimento de *software*. Essa linguagem de modelagem é definida como uma extensão de um subconjunto da Linguagem de Modelagem Unificada (UML) [Uml04] usando mecanismos de *profile* da UML.

A UML é linguagem de modelagem largamente usada na indústria para o desenvolvimento de sistemas. Essa linguagem foi desenvolvida fortemente caracterizada pela orientação a objetos, que é específico para o desenvolvimento de *software*. No entanto, para o desenvolvimento de aplicações da engenharia de sistemas, é necessário uma abordagem interdisciplinar que envolve várias áreas (ex.: engenharia de *software*, mecânica, elétrica e eletrônica) e que não se baseie somente em *software*. Dessa forma, é adotada a SysML como linguagem de modelagem dos sistemas distribuídos e dos mecanismos de tratamento de interrupções. A Figura 2.2 apresenta toda a estrutura dos diagramas da SysML. Alguns diagramas foram importados dos diagramas da UML 2.0, sem modificações. Outros diagramas, todavia, ou foram adicionados, ou modificados para atender às necessidades da engenharia de sistema.

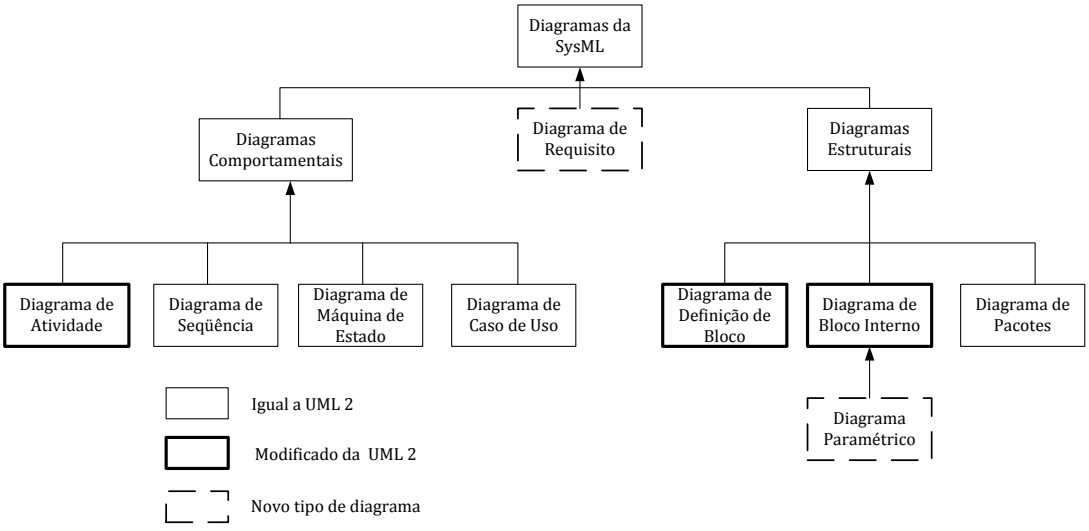


Figura 2.2: Estrutura da SysML.

A SysML é composta por 9 tipos de diagramas, que podem ser divididos em três partes: estrutural, comportamental e genérico, como descritos a seguir [S$^+$06]:

- **Estrutural.** As construções estruturais da SysML representam os elementos conceituais ou físicos. Essa parte é formada pelos seguintes diagramas: pacote, de-

finição de bloco, bloco interno e paramétrico. O diagrama de pacote é utilizado para organizar o modelo. O diagrama de definição de bloco descreve o relacionamento entre os blocos. O diagrama de bloco interno representa a estrutura interna de um bloco em termos das suas propriedades e conexões. O diagrama paramétrico é usado para expressar as restrições impostas pelos sistemas.

- **Comportamental.** As construções comportamentais da SysML representam o comportamento do modelo no tempo e no espaço. Essas construções são formadas pelos seguintes diagramas: atividades, sequência, estados e caso de uso. O diagrama de atividades descreve o fluxo de controle e o fluxo de entradas e saídas entre as ações. O diagrama de sequência mostra a colaboração dinâmica entre as várias entidades do sistema. O diagrama de estados representa o comportamento de um elemento do sistema em termos de transições entre estados provocados por eventos. Tipicamente, esse diagrama é utilizado para representar o ciclo de vida de um bloco. O diagrama de caso de uso mostra como o sistema a ser desenvolvido vai interagir com seu ambiente (usuários, sistemas e entre outros).

- **Genérico (*Cross-cutting constructs*).** As construções genéricas da SysML representam as construções que podem ser aplicadas nas partes estruturais e comportamentais. A parte genérica da SysML é formada pelo diagrama de requisito e relacionamentos de alocação. O diagrama de requisitos representa um requisito baseado em texto (definido textualmente). O relacionamento de alocação representa as relações que mapeiam um elemento do modelo em outro. A SysML suporta várias formas de alocações entre os diferentes elementos do modelo, incluindo a alocação comportamental, a estrutural e a de propriedades.

A SysML é uma linguagem destinada a apoiar uma ampla variedade de aplicações de domínio específico (*domain-specific applications*), tais como a modelagem de sistemas distribuídos [MAKT11], embarcados [AMN+10] ou aeroespaciais [GB11]. A SysML foi projetada para permitir extensões que suportam esses domínios especializados. Um exemplo pode ser a customização da SysML para o domínio dos sistemas distribuídos que inclui conceitos de dependências entre os elementos do modelo (ex.: data center e máquinas virtuais). Assim, a SysML inclui os seus próprios mecanismos de extensão, chamados de *estereótipos*, que permitem classificar os elementos com algo em comum [FMS11]. Tanto as alocações quanto os estereótipos usados neste trabalho são descritos no decorrer desta tese. Nas seções seguintes, os diagramas da SysML adotados nesta pesquisa (bloco interno, estados e atividades) são descritos.

2.3.1 Diagrama de Bloco Interno (SysML-IBD)

O bloco é a unidade básica da estrutura da SysML, sendo ele usado para representar um conjunto de funcionalidades dos sistemas ou de outros elementos de interesse. O bloco pode ser utilizado para modelar tanto a decomposição física quanto a lógica de um sistema, bem como, representar *hardware*, *software*, instalações, recursos ou pessoal [Wei11].

Múltiplos compartimentos podem ser utilizados para descrever as características do bloco, tais como propriedades, operações, restrições, alocações, e requerimento que o bloco satisfaz [Tea12]. O SysML-IBD é usado para representar as configurações estáticas dos sistemas, e.g., componentes de uma infraestrutura distribuída com ou sem redundância. Esse diagrama é baseado no diagrama de estruturas compostas da UML, o qual suporta blocos e portas de fluxo.

Os SysML-IBDs são compostos basicamente pelos seguintes elementos [Tea12]:

- **Blocos.** É usado para representar qualquer elemento do sistema (ver Figura 2.3).
- **Conectores.** Um conector é um *link* entre os blocos que permite a comunicação entre eles (ver Figura 2.3).
- **Portas.** Especificam os tipos de serviços que um bloco requer ou proporciona (ver Figura 2.3).
- **Fluxos.** Especificam o que pode fluir para dentro e para fora dos blocos (ver Figura 2.3).

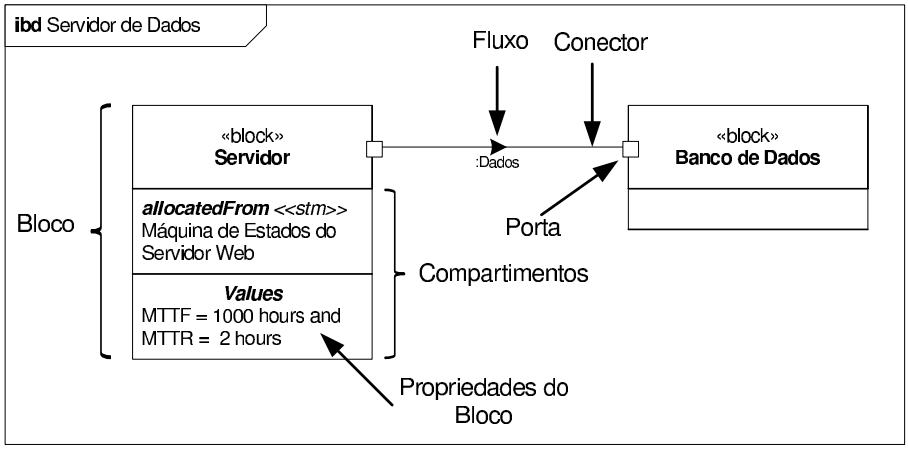


Figura 2.3: Exemplo do Diagrama Interno de Blocos.

É importante ser ressaltado que todos os diagramas da SysML possuem um quadro (*frame*) que compreende tanto o cabeçalho quanto o conteúdo do diagrama (ver Figura 2.4). O cabeçalho do diagrama, por sua vez, é composto do tipo de diagrama, do nome do diagrama, e de algumas informações adicionais que fornecem o contexto para o conteúdo do diagrama.

2.3.2 Diagrama de Estados (SysML-STM)

O diagrama de estados da SysML é importado do diagrama de estados da UML 2.0 sem modificações ou extensões. Ele é usado para representar o comportamento dinâmico de

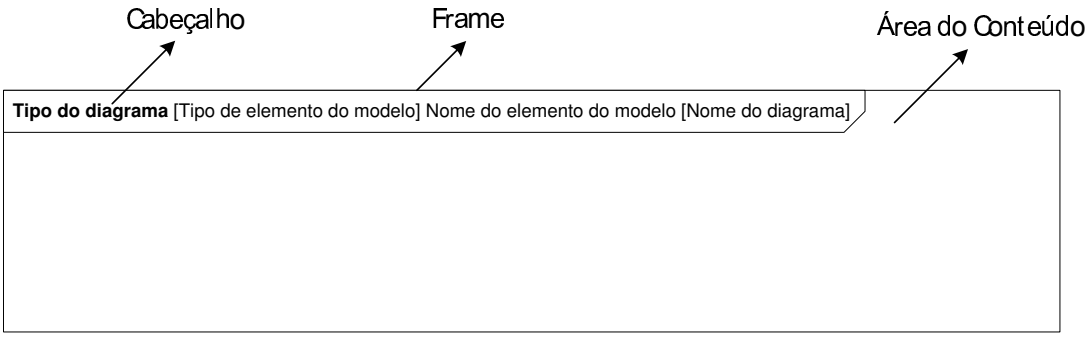


Figura 2.4: *Frame* dos Diagramas da SysML.

um elemento do sistema (ex.: o comportamento de falha e reparo de um servidor). Os SysML-STMs são baseados nos *statecharts* [Har87], que são um formalismo visual concebido por David Harel para especificar sistemas de tempo-real do tipo reativo. Sistemas do tipo reativo são caracterizados pelo fato de estarem continuamente reagindo a estímulos externos e internos. Exemplos: sistemas embutidos, aeronaves e máquinas industriais, sistemas de controle de tráfego aéreo e urbano, entre outros. Os *statecharts*, por sua vez, são uma evolução dos clássicos Diagramas de Transição de Estados (DTE). No entanto, os DTE não permitem uma visão de profundidade, hierarquia ou modularidade dos sistemas modelados. Em outras palavras, em à medida que a complexidade do sistema cresce linearmente, o número de estados e transições cresce exponencialmente, gerando diagramas grandes e confusos. Além disso, os DTEs não suportam concorrência, sendo sequenciais por natureza.

Os SysML-STMs são compostos basicamente pelos seguintes elementos [Tea12]:

- **Estado.** Representa a situação em que um elemento do sistema se encontra em um determinado momento durante sua participação em um processo. A SysML define três tipos de estados: estado simples, estado composto e sub-máquina. Estados simples são os mais simples de todos os estados, os quais não possuem subestados, conforme descrito na Figura 2.5. Estados compostos são estados que contêm dois ou mais estados. As sub-máquinas são estados compostos, cujos subestados são descritos em outro diagrama. Além disso, os estados podem conter atividades internas (*entry*, *do* e *exit*) e transições internas. Mais informações podem ser encontradas em [Gue08].

- **Transição.** Representa um evento que causa uma mudança no estado do elemento do sistema, gerando um novo estado. Uma transição é representada por uma seta ligando dois estados (ver Figura 2.5).

- **Decisão.** Utilizada para controlar desvios no fluxo de controle, o qual é composto de condições booleanas e cada condição, quando satisfeita, dispara uma transição correspondente (ver Figura 2.5).

- **Estado Inicial.** É um pseudoestado cuja função é determinar o início de um SysML-STM ou de um estado composto (ver Figura 2.5).
- **Estado Final.** O estado final é um estado cuja função é determinar o fim de um SysML-STM ou de um estado composto (ver Figura 2.5).

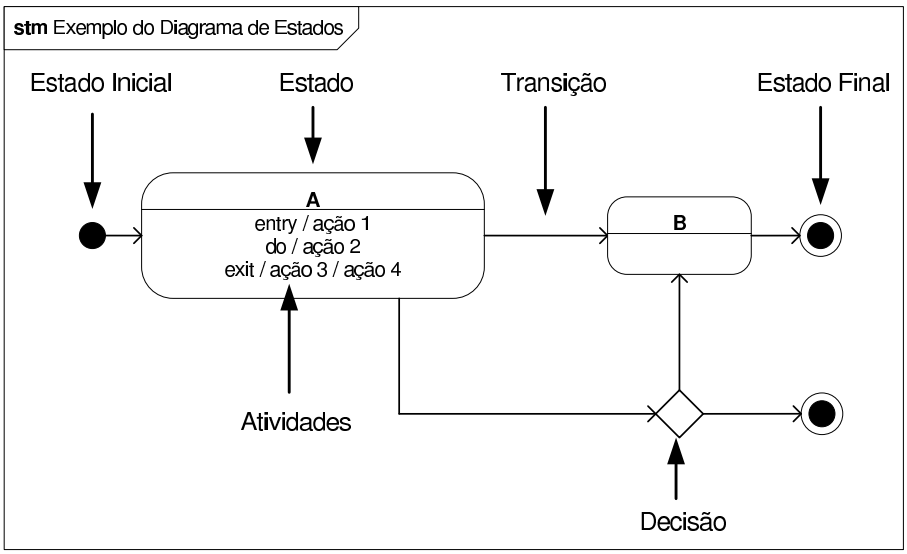


Figura 2.5: Exemplo do Diagrama de Estados.

2.3.3 Diagrama de Atividades (SysML-AD)

De forma semelhante aos SysML-STMs, os SysML-ADs são utilizados para a modelagem dos aspectos dinâmicos de um sistema. Neste trabalho, no entanto, o SysML-AD é utilizado para descrever o fluxo de atividades dos mecanismos de tratamento de interrupções. É importante ser destacado que o SysML-AD possui o enfoque no fluxo entre atividades, enquanto o SysML-STM no fluxo de estados. As modificações no diagrama de atividades da SysML, em relação ao diagrama de atividades da UML 2.0, incluem os seguintes aspectos: controle com dados, sistemas contínuos, probabilidades, atividades como blocos e linha do tempo. Mais informações podem ser encontradas em [Tea12].

Os SysML-ADs são compostos basicamente pelos seguintes elementos [Tea12]:

- **Atividades e Ações.** Atividade refere-se à execução de um processamento não atômico, envolvendo uma ou mais ações. Uma ação consiste em um processamento que resulta em uma mudança de estado no sistema. Ações e atividades têm a mesma representação gráfica (ver Figura 2.6).
- **Transição.** Representa o fluxo de uma atividade/ação para outra (ver Figura 2.6).

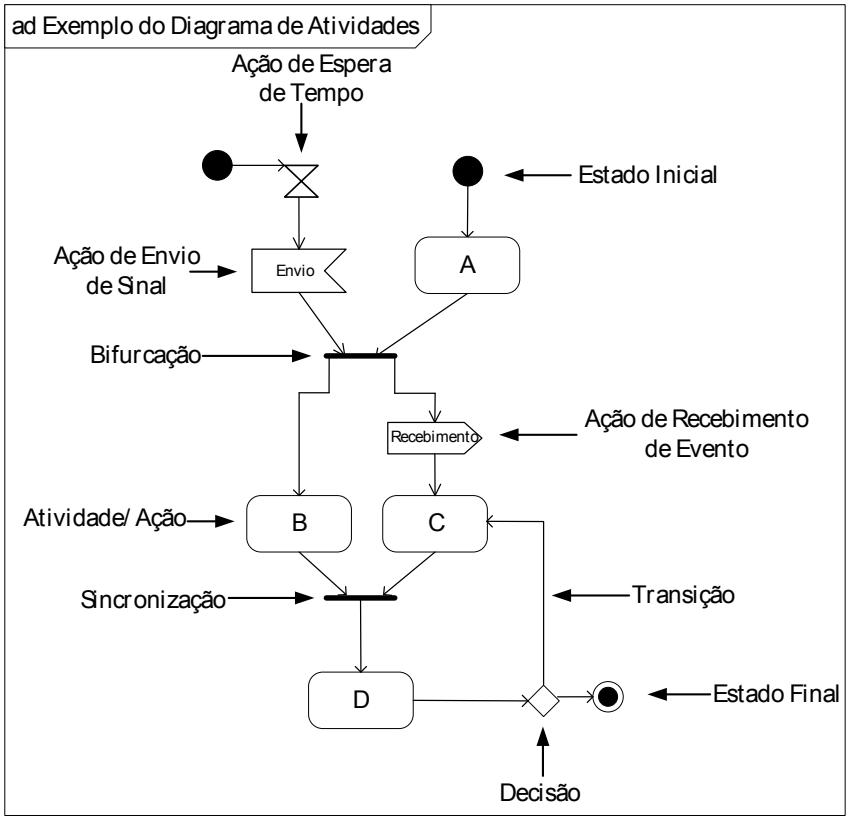


Figura 2.6: Exemplo do Diagrama de Atividades.

- **Bifurcação.** Representa a divisão de um fluxo de controle em dois ou mais fluxos de controle concorrentes e independentes (ver Figura 2.6).
- **Sincronização.** Representa a sincronização de dois ou mais fluxos concorrentes (ver Figura 2.6).
- **Decisão.** Utilizada para controlar desvios no fluxo de controle, o qual é composto de condições booleanas e cada condição, quando satisfeita, dispara uma transição correspondente (ver Figura 2.6).
- **Ação de Envio de Sinal.** São sinais[1] enviados para participantes externos que podem causar a execução de uma atividade (ver Figura 2.6).
- **Ação de Recebimento de Evento.** Representa o recebimento de sinais (ver Figura 2.6).
- **Ação de Evento de Tempo.** Representa o tempo que transcorre até que ocorra a próxima ação/atividade (ver Figura 2.6).
- **Estado Inicial.** É um pseudoestado cuja função é determinar o início de um SysML-AD (ver Figura 2.6).

[1]Sinais são mensagens que podem ser enviadas e recebidas.

- **Estado Final.** O estado final é um estado cuja função é determinar o fim de um SysML-AD (ver Figura 2.6).

2.4 MARTE

A linguagem de modelagem UML possui um mecanismo que permite a extensão da linguagem, denominado *profile*. Através da criação desse *profile*, os projetistas podem criar elementos que expressam conceitos específicos de um determinado domínio, sem que haja a necessidade da criação de uma nova linguagem de modelagem [OMG11]. Nesse sentido, em junho de 2007, a OMG criou um novo *profile* da UML 2.0, cuja principal função é auxiliar a construção de modelos que possam ser usados para fazer precisões quantitativas relativas às características de sistemas de tempo-real e embarcados, levando em consideração tanto as características de *hardware* quanto de *software*. Esse novo *profile* é chamado de MARTE (*Modeling and Analysis of Real-time and Embedded systems*). É importante destacar que MARTE é compatível com os demais *profiles* ou padrões existentes, tais como: SysML [Tea12], SAE AADL [FLV03], EAST-ADL v2 [DSLT05], OSEK [Moo02], ARINC 653 [Pri08], e POSIX [Gal95].

As notações do *profile* MARTE usam estereótipos que permitem aos projetistas mapear elementos do modelo para uma semântica de domínio de análise (ex.: análise de dependabilidade) e atribuir valores para as propriedades que são necessárias, a fim de obter as métricas de interesse. A especificação do *profile* MARTE é dividida em um conjunto de unidades de extensão que contêm anotações para especificar as características dos sistemas. Este trabalho usa a unidade de extensão para Modelagem de Análise de Desempenho (PAM), que tem o objetivo de oferecer conceitos gerais, os quais são necessários para modelar as propriedades temporais dos sistemas. Essa unidade é composta por um conjunto de estereótipos e valores marcados. Neste trabalho, o estereótipo *PAStep* e o valor marcado *hostDemand* são usados.

A Figura 2.7 apresenta um exemplo do diagrama de estados com anotações de disponibilidade especificadas por MARTE. O estereótipo *PAStep* e o valor marcado *HostDemand* são usados. O estereótipo descreve uma ação, enquanto o valor marcado representa o nome da propriedade, bem como o valor atribuído àquela propriedade. A transição *Falha* é anotada com o estereótipo *PAStep*, significando que o evento de falha tem um tempo exponential de 8766 horas. O valor marcado é dado por: $\{HostDemand = (exp(8766), h)\}$. De forma semelhante, a transição *Rec* é anotada com *PAStep*, significando que o evento de recuperação tem um tempo exponential de 2 horas. O valor marcado é dado por $\{HostDemand = (exp(2), h)\}$. Mais informações sobre todos os pacotes, estereótipos e valores marcados suportados por MARTE podem ser encontradas em [OMG11].

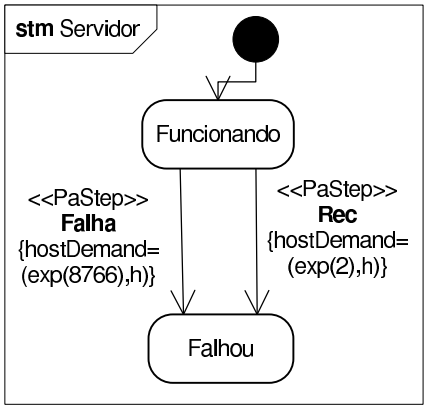


Figura 2.7: Diagrama de Estados com Anotações de MARTE.

2.5 DEPENDABILIDADE

Devido à crescente expansão dos sistemas distribuídos, a dependabilidade tem se tornado um atributo de grande interesse no desenvolvimento, operação, implantação e manutenção de tais sistemas, visto que os serviços oferecidos por eles precisam ser cada vez mais confiáveis [MTMK11]. Os primeiros conceitos de dependabilidade foram inicialmente definidos no artigo *Dependability Basic Concepts and Terminology* [Lap92], e, desde então, têm sido largamente utilizados tanto na academia como na indústria. Os sistemas que são analisados sob as métricas de dependabilidade são denominados de sistemas de funcionamento confiável.

De acordo com [ALRL04], "*a dependabilidade de um sistema é a habilidade de evitar falhas de serviços que são mais frequentes ou mais severas do que o aceitável*". A Figura 2.8 apresenta a taxonomia dos sistemas de funcionamento confiável. A dependabilidade compreende os seguintes atributos: disponibilidade (prontidão do sistema para ser usado), confiabilidade (continuidade da prestação do serviço), segurança (ausência de consequências catastróficas), integridade (ausência de alterações no sistema) e manutenção (capacidade de manutenção). Os meios são agrupados em remoção de defeitos (reduzir o número ou a severidade das falhas), tolerância a defeitos (entregar o serviço correto mesmo na presença de falhas), prevenção de defeitos (prevenir a ocorrência ou introdução de falhas) e previsão de defeitos (estimar o número presente, a incidência futura e as consequências das falhas). As ameças são agrupadas em defeito (do inglês *fault*), error (do inglês *erro*) e falha (do inglês *failure*). A falha do sistema consiste no desvio do comportamento do sistema em relação à sua especificação. A causa de uma falha é um erro. Um erro é a porção do estado do sistema responsável por conduzi-lo a uma falha. Cada erro, por sua vez, é causado pela ativação de um defeito.

Nossa sociedade está cada vez mais dependente dos serviços oferecidos pelos sistemas distribuídos de modo que interrupções mínimas em tais serviços podem causar tanto prejuízos financeiros quanto danos à imagem institucional. Dessa forma, os sistemas distribuídos precisam cada vez mais prover serviços com alta disponibilidade. Ou seja, esses

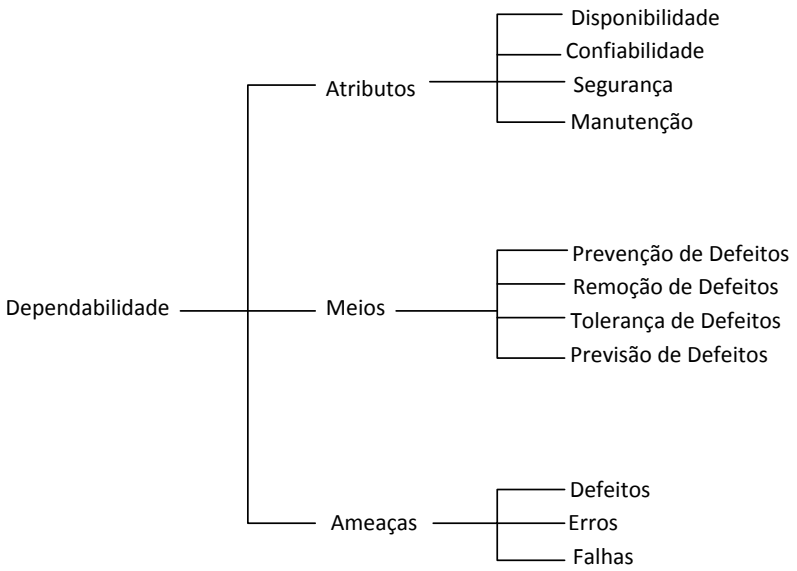


Figura 2.8: Taxonomia dos Sistemas de Funcionamento Confiável.

sistemas precisam manter seus serviços disponíveis o máximo de tempo possível, mesmo na presença de interrupções. Este trabalho tem o foco de estudar aspectos de dependabilidade desses sistemas, mais especificamente o atributo de disponibilidade, em que modelos analíticos são criados para calcular métricas relacionadas à disponibilidade (ex.: disponibilidade estacionária, indisponibilidade, *downtime* etc.) dos sistemas distribuídos, considerando mecanismos de tratamento de interrupções. Os resultados obtidos a partir das métricas aferidas são utilizados para encontrar possíveis pontos gargalos nas infraestruturas analisadas, bem como, prover informações para os projetistas na tomada de decisão.

2.6 MODELAGEM ANALÍTICA

Segundo [JC11], "*a modelagem de um sistema é a representação, geralmente simplificada, de um sistema (real) através de relações lógicas e matemáticas com o objetivo de estudar e entender o comportamento desse sistema*". A Figura 2.9 apresenta o esquema de modelagem. Um modelo, na maioria das vezes, aproxima o comportamento do sistema real, em que a qualidade do mesmo é dada justamente pela aptidão demonstrada para a previsão de resposta do sistema (variáveis de saída) a um estímulo (variáveis de entrada). Isto é, o nível de abstração do modelo deve estar condizente com os objetivos da análise. A inclusão de muitos detalhes poderá impactar na complexidade do modelo, bem como na sua avaliação. Por outro lado, a omissão de características importantes podem resultar em dados imprecisos. Nas seções seguintes, são introduzidos os principais modelos analíticos usados para avaliação de dependabilidade dos sistemas computacionais. Esses modelos são baseados em dois tipos: modelos de espaço de estados e modelos sem espaço de estados. No entanto, primeiramente, são apresentadas as abordagens para modelar a disponibilidade dos sistemas computacionais.

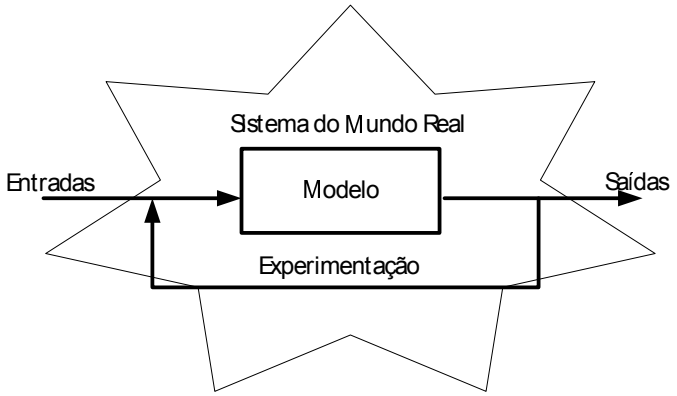


Figura 2.9: Esquema de Modelagem.

2.6.1 Abordagens de Modelagem

Segundo [Rec94], "*disponibilidade é a habilidade de um sistema/componente estar em um estado para executar uma função requerida em um instante qualquer de tempo ou em um instante qualquer dentro de um intervalo de tempo, assumindo que os recursos externos, se necessários, são fornecidos*". Para sistemas críticos (ex.: sistemas hospitalares, sistemas de usinas nucleares e sistemas de controle de aeronaves), alta disponibilidade é extremamente importante, visto que falhas podem ser catastróficas, resultando em perda de vidas ou altos prejuízos financeiros.

Modelos estocásticos têm sido largamente usados para prever a disponibilidade dos sistemas. Esses modelos podem prover importantes informações para os projetistas antes de o produto ser colocado em operação considerando diferente cenários. Os aspectos de disponibilidade de um sistema/componente são geralmente descritos através de modelos sem espaço de estados (ex.: diagrama de bloco de confiança (RBD) [XDP04], árvore de falhas (FT) [STP96] e gráfico de confiança (Relgraph) [Tri82]), modelos de espaço de estados (e.g.: cadeia de Markov [Tri82], redes de Petri [MF76], entre outros), hierarquia [LYT03] e iteração de ponto fixa (*fixed-point iterative*) [GTNK10]. *Downtime*, disponibilidade estacionária, disponibilidade instantânea e disponibilidade intervalar são algumas métricas de disponibilidade dos sistemas. Assumindo tempos de falha e reparo exponencialmente distribuídos com as respectivas taxas λ e μ, a disponibilidade no tempo t e a disponibilidade intervalar podem ser calculadas pelas seguintes expressões [Tri82].

$$A(t) = \frac{\mu}{\lambda+\mu} + \frac{\mu}{\lambda+\mu}e^{-(\lambda+\mu)t} \tag{2.1}$$

$$A_I(t) = \frac{\int_0^t A(x)dx}{t} = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{(\lambda+\mu)^2 t}(1 - e^{-(\lambda+\mu)t}) \tag{2.2}$$

Calculando o limite até o infinito da disponibilidade instantânea, a disponibilidade em estado estacionário (A_{sys}) pode ser calculada como segue.

$$A_{sys} = \lim_{t \to \infty} A(t) = \frac{\mu}{\lambda + \mu} \tag{2.3}$$

$$\tag{2.4}$$

A indisponibilidade em estado estacionário e o *downtime* (em minutos por ano) são obtidos a partir da disponibilidade em estado estacionário (A_{sys}), conforme as equações abaixo.

$$U_{sys} = (1 - A_{sys}) \tag{2.5}$$

$$Downtime = (1 - A_{sys}) \times 8760 \times 60 \tag{2.6}$$

2.6.2 Modelos sem Espaços de Estados

Modelos de disponibilidade podem ser construídos usando modelos sem espaço de estados, tais como diagrama de blocos de confiança, gráfico de confiança e árvore de falhas com e sem eventos repetidos. Tais modelos são fáceis de serem usados e possuem uma solução geralmente rápida, pois eles não requerem a geração de espaço de estados durante a sua solução [SRT00]. Porém, para uma rápida solução, os modelos assumem que os componentes do sistema são independentes. A disponibilidade, a indisponibilidade, a confiança e o tempo médio de falha dos sistemas podem ser computados usando esses modelos. As três principais técnicas de solução para os modelos sem espaço de estados são: fatoração [Tri82], soma dos produtos disjuntos [Tri82] e árvore de decisão binária [ZWST03].

O RBD é um modelo sem espaço de estados que permite calcular métricas de confiança e disponibilidade de sistemas complexos. No modelo do RBD, os componentes são combinados em blocos, os quais podem estar em série, paralelo e *k-out-of-n*. Uma estrutura em série representa uma dependência direta entre os componente, em que o sistema todo falha se um de seus componentes falhar. Uma estrutura paralela é usada para mostrar redundância e significa que todo o sistema pode continuar suas operações desde que ao menos um componente esteja operational. A estrutura de *k-out-of-n* representa que todo o subsistema está funcionando corretamente desde que k componentes estejam funcionando corretamente de n componentes. As estruturas em série e paralelo são casos especiais do *k-out-of-n* [STP96]. Uma estrutura em série é um *n-out-of-n*, enquanto uma estrutura paralela é um *1-out-of-n*. A Figura 2.10 apresenta um exemplo de um sistema de armazenamento de dados modelado através do RBD. Esse modelo de disponibilidade é composto por um servidor, um *hub* e n dispositivos de armazenamento (DA). O sistema está funcionado corretamente, se pelo menos um dispositivo de cada tipo (*hub*, servidor e dispositivos de armazenamento) estiver funcionando corretamente.

A confiabilidade do sistema para o arranjo em série, considerando n componentes é dada por:

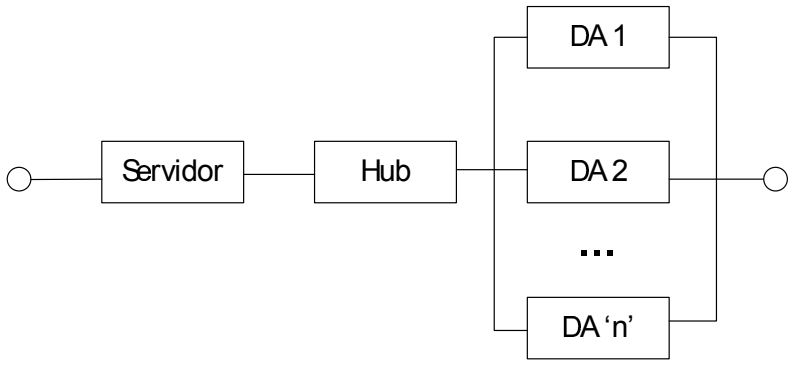


Figura 2.10: Diagrama de Bloco para o Sistema de Armazenamento de Dados.

$$R_s = \prod_{i=1}^{n} R_i \tag{2.7}$$

em que R_i é a confiabilidade do bloco i.

A confiabilidade do modelo em paralelo, considerando n componentes, pode ser obtida pela equação abaixo.

$$R_p = 1 - \prod_{i=1}^{n} (1 - R_i) \tag{2.8}$$

onde R_i é a confiabilidade do bloco i.

Segundo Xie *et al.* [XDP04], as equações básicas para calcular a confiabilidade do sistema em série e em paralelo (ver Equações 2.7 e 2.8) podem ser combinadas para calcular a confiabilidade do sistema contendo estruturas tanto em paralelo quanto em série (sistema série-paralelo). Dessa forma, qualquer sistema série-paralelo pode ser eventualmente transformado em um bloco e sua confiabilidade pode ser facilmente calculada usando, repetidamente, as equações acima.

2.6.3 Modelos de Espaço de Estados (ou Modelos Combinatórios)

Modelos sem espaço de estados (ex.: RBD e FT) não conseguem facilmente levar em consideração detalhes comportamentais, e.g., fator de cobertura, falhas correlatas, dependências de reparo, etc. Por outro lado, os modelos de espaço de estados conseguem capturar tais comportamentos em detalhes. As cadeias de Markov de tempo contínuo (CTMCs) são modelos de espaço de estados amplamente conhecidos no meio acadêmico, sendo elas usadas no estudo de desempenho e disponibilidade dos sistemas computacionais. As CTMCs homogêneas são representadas por estados e transições entre os estados, em que o tempo médio de permanência (*sojourn time*) segue uma distribuição exponential. Se relaxarmos a suposição da distribuição exponential, então o modelo pode se tornar

um processo semi-Markoviano [CST00], um processo de Markov regenerativo [LTP95] ou uma cadeia de Makov não homogênea [GLT04]. A Figura 2.11 apresenta um exemplo simples de um modelo CTMC para um sistema redundante de dois componentes com a mesma taxa de reparo μ. A taxa de falha dos dois componentes é λ. Quando ambos os componentes tiverem falhado, então o sistema é considerado *down*.

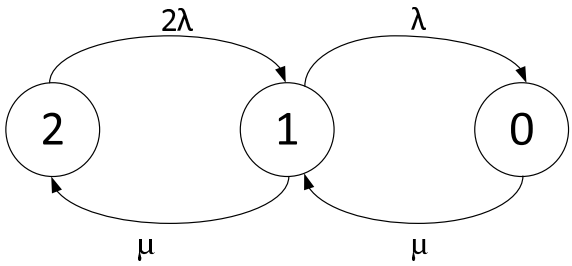


Figura 2.11: Exemplo de uma CTMC.

Um dos principais problemas das cadeias de Markov é a explosão do espaço de estados, visto que a cadeia de Markov de um sistema complexo pode chegar a milhares ou milhões de estados. As redes de Petri estocásticas (SPNs) podem ser usadas para a especificação e geração automática das cadeias de Markov, a fim de tolerar o problema de explosão do espaço de estados através de um modelo conciso e menor. Modelos baseados em redes de Petri (RdP) têm sido largamente usados para modelar vários projetos indo desde os sistemas embarcados de tempo-real [AMCN09] até os sistemas de computação em nuvem [Gho12]. As redes de Petri são detalhadas na seção seguinte, visto que elas são usadas ao longo de toda a tese.

2.7 REDES DE PETRI

O conceito das RdP foi criado por Carl Adam Petri na sua tese de doutorado, a qual foi intitulada de *Kommunikation mit Automaten* (Comunicação com Autômatos) e apresentada na Universidade de Bonn em 1962 [Pet66]. Deste então, esse formalismo tem sido amplamente utilizado em diferentes áreas, tais como Ciência da Computação, Engenharia Elétrica, Administração, Química, entre outras. Diversas variantes do modelo de RdP clássico têm sido desenvolvidas ao longo do tempo, tais como redes temporizadas [MF76], estocásticas [Mar90], alto-nível [Jen91] e orientadas a objetos [Jan98]. Isso é devido à necessidade de suprir as diferentes áreas de aplicação, além de prover facilidades de comunicação e transferência de métodos e ferramentas de uma área para outra.

As vantagens do uso das RdPs são diversas, entre elas, é possível destacar as seguintes [GV02]:

- RdPs fornecem um formalismo de modelagem que permite sua representação gráfica e são fundamentadas matematicamente.
- RdPs fornecem mecanismos de refinamento e abstração que são de grande importância para o projeto de sistemas complexos.

- Existe uma grande variedade de ferramentas disponíveis para as RdPs, tanto comerciais quanto acadêmicas, para modelagem, análise e verificação.
- RdPs têm sido utilizadas nas mais diversas áreas. Portanto, vários resultados são encontrados na literatura para os diversos domínios da sua aplicação.
- Existem várias extensões do modelo básico das RdPs.

As RdPs são compostas pelos seguintes elementos [Mur89]:

- **Lugares.** Representam os elementos passivos da rede, tendo como principal função o armazenamento de *tokens*, os quais são removidos e adicionados à medida que as transições são disparadas. Graficamente, os lugares são representados por círculos (ver Figura 2.12 (a)).
- **Transições.** Representam os elementos ativos da rede, ou seja, as ações realizadas pelo sistema. Para que uma transição esteja habilitada, é necessário que todas as suas pré-condições sejam satisfeitas, caso uma pré-condição não seja satisfeita, a transição estará desabilitada. Uma vez que a transição satisfaz todas as pré-condições, ela poderá disparar, removendo uma determinada quantidade de *tokens* dos lugares e colocando em outros, gerando assim as pós-condições. Graficamente, são representadas por traços ou barras (ver Figura 2.12 (b)).
- **Arcos.** Representam o fluxo dos *tokens* pela rede (ver Figura 2.12 (c)). Na representação gráfica, as transições e os lugares podem ser conectados por múltiplos arcos (arcos multivalorados) que podem ser compactados em um único arco rotulado (Figura 2.13).
- **Tokens.** Representam o estado em que o sistema se encontra em determinado momento (ver Figura 2.12 (d)).

Figura 2.12: Elementos de uma Rede de Petri.

Formalmente, as redes de Petri podem ser definidas conforme descreve a Definição 2.1.

Definição 2.1. (Redes de Petri) Uma rede de Petri é uma 5-tupla, $PN = (P, T, F, W, M_0)$, onde:

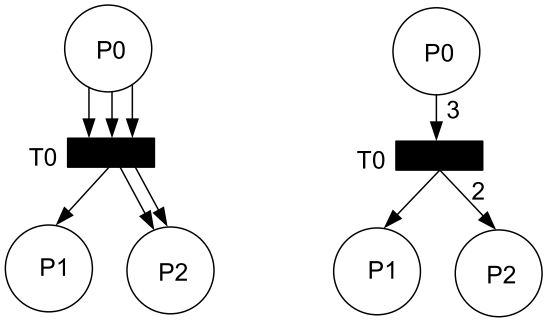


Figura 2.13: Arco Multivalorado.

- $P = \{p_1, p_2, ..., p_n\}$, é um conjunto finito de lugares;
- $T = \{t_1, t_2, ..., t_n\}$, é um conjunto finito de transições;
- $F \subseteq (P \times T) \cup (T \times P)$, é um conjunto de arcos;
- $W : F \rightarrow \{1, 2, 3, 4...n\}$, é a função de atribuição de peso aos arcos.
- $M_0 : P \rightarrow \{0, 1, 2, 3...\}$, é a marcação inicial.

A seguir, na Figura 2.14, é apresentado um exemplo de uma RdP, no qual o funcionamento de uma lâmpada é modelado. Nesse modelo, os lugares e as transições representam, respectivamente, os estados da lâmpada (*aceso* ou *apagado*) e as ações que alteram o estado da lâmpada (*ligar* ou *desligar*). O estado atual do modelo é representado por uma marca (*token*) no lugar correspondente à situação atual do modelo, como descrito na Figura 2.14 (a). Visto que o estado atual do modelo é *aceso*, a única transição que poderá ser disparada é *desligar*. Uma vez que essa transição seja disparada, então, o modelo passará do estado *aceso* (ver Figura 2.14 (a)) para o estado *apagado* (ver Figura 2.14 (b)). Mais informações relacionadas a estrutura básica das redes de Petri podem ser encontradas em [Mur89].

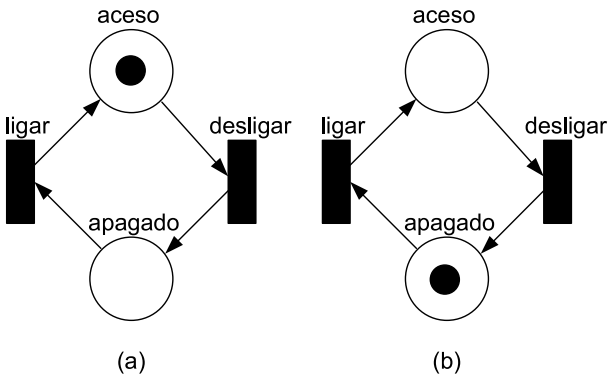


Figura 2.14: Exemplo de uma Rede de Petri.

2.7.1 Propriedades das Redes de Petri

Diversas propriedades podem ser obtidas a partir dos sistemas modelos usando as RdP, permitindo assim revelar as mais diversas características do sistema. Essas propriedades podem ser subdivididas em comportamentais e estruturais, as quais são descritas nas subseções seguintes.

2.7.1.1 Propriedades Comportamentais Essa subseção descreverá as principais propriedades comportamentais baseadas em [Mur89]. As propriedades comportamentais são aquelas que dependem da marcação.

Alcançabilidade

A propriedade de alcançabilidade indica a possibilidade de atingir uma determinada marcação pelo disparo de um número finito de transições a partir de uma dada marcação inicial. Uma marcação M_0 é dita alcançável a partir de M_i, se existir uma sequência de disparos que transforme M_0 em M_i. A sequência de disparo é denotada pelo conjunto $\sigma = \{t1, t2, ..., tn\}$. Nesse caso, M_i é alcançável a partir de M_0 por σ. Onde σ é formalmente descrito por $M_0[\sigma > M_i$.

Limitação e Safeness

Uma rede é dita *k-limitada* se todos os seus lugares forem limitados, ou seja, o número de *tokens* em cada lugar não deve ultrapassar um número finito k, para qualquer marcação alcançável a partir de M_0. Uma rede de Petri é dita safeness se $k = 1$.

Liveness

Uma rede é dita *live* se, não importa quais marcações sejam alcançáveis a partir de um marcação inicial m_0, for possível disparar qualquer transição através do disparo de alguma sequência de transições $L(M_0)$. O conceito de *deadlock* está fortemente conectado ao conceito de *liveness*. No entanto, o fato de um sistema ser livre de *deadlocks* não resulta que este seja *liveness*, contudo um sistema *liveness* implica em um sistema livre de *deadlocks*. A análise de *liveness* de uma rede permite verificar se os eventos modelados efetivamente ocorrem durante o funcionamento do sistema, ou se foram definidos eventos mortos no modelo. Essa propriedade *liveness* pode ser definida em níveis, conforme apresentados abaixo:

- morta (L0-*live*). Se t nunca pode ser disparada em qualquer sequência $L(M_0)$;
- L1-*live* (potencialmente disparável). Se t pode ser disparável em pelo menos alguma sequência $L(M_0)$;
- L2-*live*. Se dado um inteiro positivo k, t puder ser disparado pelo menos k vezes em alguma sequência $L(M_0)$;
- L3-*live*. Se t aparece um número infinito de vezes em alguma sequencia de disparo $L(M_0)$;

- L4-*live* ou simplesmente *live*. Se t é potencialmente disparável para todas as marcações da rede, ela é dita *liveness*.

Cobertura

A propriedade de cobertura está fortemente conectada ao conceito de alcançabilidade e *liveness*. Quando se deseja saber se alguma marcação M_i, pode ser obtida a partir de uma marcação M_j, temos o problema denominado cobertura de uma marcação. Uma marcação M_i é dita coberta se existe uma marcação M_j tal que $M_j > M_i$. Fora isso, em alguns sistemas, deseja-se apenas observar o comportamento de determinados lugares. Para isso, restringe-se a pesquisa a apenas um conjunto de lugares de particular interesse (cobertura de submarcações).

Reversibilidade e Home State

Uma rede é dita reversível se, para cada marcação M em $R(M_0)$, M_0 é alcançável a partir de M. Assim, a rede possui a capacidade de retornar à marcação inicial. Além disso, em algumas aplicações não é necessário voltar à marcação inicial, mas sim a uma marcação específica. Essa marcação específica é denominada *Home State*.

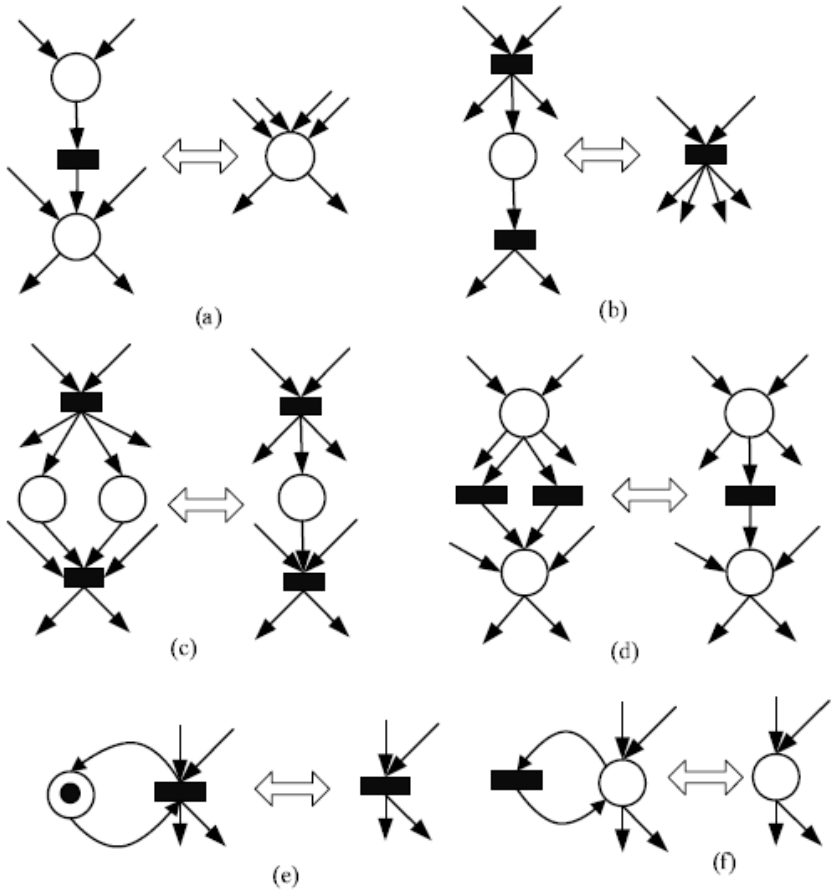
2.7.1.2 Propriedades Estruturais As propriedades estruturais são aquelas que não dependem da marcação, ou seja, possuem dependência exclusivamente da topologia da rede. Abaixo serão descritas as principais propriedades estruturais baseadas em [Mur89].

- **Limitação Estrutural.** Uma rede é dita limitada estrutural se o número de *tokens* é limitado para qualquer marcação inicial.
- **Conservação.** A conservação é uma importante propriedade das RdP, pois permite a verificação da não destruição de recursos através da conservação de *tokens*.
- **Repetitividade.** Uma rede é considerada repetitiva se para uma marcação e uma sequencia de transições disparáveis, todas as transições dessa rede são disparadas ilimitadamente.
- **Consistência.** Uma rede é dita consistente se dada uma sequencia de transições disparáveis a partir de uma marcação inicial M_0 retornar a M_0, porém todas as transições da rede são disparadas pelo menos uma vez.

2.7.2 Redução

Reduções das RdP são transformações aplicadas ao modelo de um sistema com o objetivo de simplificá-lo, preservando as propriedades do sistema a ser analisado. Normalmente essa técnica é utilizada para facilitar a análise de sistemas complexos. Por outro lado, é possível transformar um modelo abstrato em um modelo refinado, mantendo-se suas propriedades [Mur89]. Existem várias técnicas de transformação para as RdPs. A Figura 2.15 mostra algumas delas, dentre elas:

- Lugares em série, como apresentado na Figura 2.15 (a).
- Transições em série, como apresentado na Figura 2.15 (b).
- Lugares paralelos, como apresentado na Figura 2.15 (c).
- Transições paralelas, como apresentado na Figura 2.15 (d).
- Lugares de *self-loops*, como apresentado na Figura 2.15 (e).
- Transições de *self-loops*, como apresentado na Figura 2.15 (f).



Fonte: [Mur89, p. 553]

Figura 2.15: Redução e Refinamento para as Redes de Petri.

2.7.3 Extensões Temporizadas das Redes de Petri

Originalmente, as redes de Petri propostas por Adam Petri não possuíam notação de tempo ou probabilidade, devido às dificuldades que a temporização adicionaria à análise da rede. Os primeiros trabalhos utilizando as redes de Petri com tempo são os de P.M Merlin e D.J Faber [MF76] e J.D Noe e G.J Nutt [NN73]. Existem diferentes formas de incorporar tempo às redes de Petri, divergindo em relação à localização e ao tipo de tempo associado.

Quanto à localização, o tempo pode ser associado aos lugares, às transições e aos *tokens* [VDAVHR00].

- **Lugares.** O lugar permite que seus *tokens* sejam consumidos após um dado tempo que é associado ao lugar.
- **Transições.** O disparo da transição ocorre depois de um tempo de atraso correspondente a partir do momento em que a transição se torna habilitada.
- **Tokens.** Os *tokens* possuem uma informação de tempo disponível que indica quando o *token* estará disponível para ser consumido.

Na maioria dos modelos das redes de Petri temporizadas, o tempo é associado às transições, visto que em poucos modelos o tempo está associado aos lugares ou arcos [VDAVHR00]. Quanto ao tipo de tempo, pode ser determinístico [Ram74], intervalar [MF76] ou estocástico [AMCB84].

- **Determinísticos.** Os tempos determinísticos indicam tempos absolutos relativos à execução dos eventos correspondentes.
- **Intervalares.** Os tempos intervalares usam intervalos para descrever os tempos máximos e mínimos para a duração das atividades.
- **Estocásticos.** Neste modelo, cada *delay* é descrito por uma distribuição de probabilidade.

Com adição de tempo as transições das RdP, surgem novos problemas com relação às regras de disparo das transições. Embora a transição que tem o menor tempo dispare primeiro, o problema está em decidir o que será feito com as transições que forem desabilitadas e com as não envolvidas no conflito. Para a resolução desse tipo de problema, três tipos de abordagens podem ser adotadas [VDAVHR00]:

- **Resampling.** Após cada disparo, os *timers* de todas as transições são reiniciados. Nesse caso, não há a necessidade de memória.
- **Enabling Memory.** Após cada disparo, os *timers* das transições que ficaram desabilitadas são reiniciados.

- **Age Memory.** Após cada disparo, os *timers* de todas as transições mantêm seus valores presentes.

Um novo conceito associado às redes de Petri com tempo é o de grau de habilitação. Esse conceito determina o número de vezes que uma determinada transição pode ser disparada numa determinada marcação, antes de se tornar desabilitada. As semânticas de temporização indicam quantos disparos podem ser feitos por unidade de tempo numa transição, como mostra a seguir:

- **Single-Serve.** Apenas um token é disparado por vez, ou seja, a capacidade de um lugar/transição é 1.
- **Multiple-server.** É possível fazer k disparos por vez, ou seja, a capacidade de um lugar/transição é um k inteiro.
- **Infinite-Server.** É possível fazer infinitos disparos de uma única vez.

2.7.4 Redes de Petri Determinísticas e Estocásticas

Nesta pesquisa adotamos uma extensão das redes de Petri temporizadas, denominada de Redes de Petri Determinísticas e Estocásticas (DSPNs) [Ger00, ASM$^+$09], visto que os modelos obtidos pelo processo de mapeamento são compostos por transições exponenciais e determinísticas. Formalmente, as DSPNs podem ser definidas, conforme descreve a Definição 2.2.

Definição 2.2. (Redes de Petri Determinísticas e Estocásticas) Uma DSPN é uma tupla, $DSPN = (P, T, I, O, H, \Pi, G, M_0, Atts)$, onde:

- $P = \{p_1, p_2, ..., p_n\}$ é o conjunto finito de lugares,
- $T = \{t_1, t_2, ..., t_m\}$ é o conjunto finito de transições imediatas, determinísticas e exponenciais,
- $I \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ é a matrix que representa os arcos de entrada (podem ser dependente da marcação),
- $O \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ é a matrix que representa os arcos de saída (podem ser dependente da marcação),
- $H \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ é uma matrix que representa os arcos inibidores (podem ser dependente da marcação),
- $\Pi \in \mathbb{N}^m$ é um vetor que associa o nível de prioridade para cada transição,
- $G \in (\mathbb{N}^n \rightarrow \{true, false\})^m$ é o vetor que associa uma função de guarda para cada transição que pode depender da marcação atual,

- $M_0 \in \mathbb{N}^n$ é um vetor que define a marcação inicial para cada lugar (estado inicial),
- $Atts = (Dist, W, Markdep, Policy, Concurrency)^m$ compreende o conjunto de atributos para as transições, onde:
  - $Dist \in \mathbb{N}^m \rightarrow \mathcal{F}$ é uma função de distribuição de probabilidade associada ao tempo de uma transição, sendo que o domínio de $\mathcal{F}$ é $[0, \infty)$ (a distribuição pode ser dependente de marcação),
  - $W \in \mathbb{N}^m \rightarrow \mathbb{R}^+$ é a função peso, que associa um peso (w_t) às transições imediatas e uma taxa λ_t às transições temporizadas, onde:
    $$W(t) = \begin{cases} w_t \geq 0, & \text{se } t \text{ é uma transição imediata;} \\ \lambda_t > 0, & \text{caso contrário.} \end{cases}$$
  - $Markdep \in \{constant, enabdep\}$ define se a distribuição de probabilidade associada ao tempo de uma transição é constante (*constant*) ou dependente de marcação (*enabdep*),
  - $Policy \in \{prd, prs\}$ é a política de preempção adotada pela transição (*prd - preemptive repeat different* corresponde à $resampling$ e prs *-preemptive resume* possui significado idêntico à *age memory policy*),
  - $Concurrency \in \{ss, is\}$ é o grau de concorrência das transições, onde ss representa a semântica do servidor único e is significa a semântica do servidor infinito (mesmo sentido dos modelos de fila).

2.8 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os principais conceitos que envolvem esta pesquisa. Primeiramente, os sistemas distribuídos foram apresentados, destacando suas principais características, assim como os principais mecanismos de tratamento de interrupções. Subsequentemente, a SysML foi introduzida, apresentando os diagramas (interno de bloco, estados e atividades) que são mapeados em um modelo analítico. Após isso, o *profile* MARTE foi apresentado, detalhando as anotações que são usadas neste trabalho. Por fim, as redes de Petri também foram introduzidas, mostrando que elas são uma ferramenta bem estabelecida para modelagem e análise de vários tipos de sistemas, tais como sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e estocásticos.

CAPÍTULO 3

TRABALHOS RELACIONADOS

We must believe that we are gifted for something, and that this thing, at whatever cost, must be attained.

—MARIE CURIE

O trabalho desenvolvido nesta pesquisa é de natureza interdisciplinar envolvendo várias áreas, tais como Engenharia de *Software*, Avaliação de Dependabilidade, entre outros. Dessa forma, este capítulo foi dividido em seções para proporcionar uma melhor leitura do conteúdo, visto que não existem trabalhos que propõem o mapeamento dos diagrams da SysML, anotados de acordo com MARTE, em modelos DSPN a fim de estudar os mecanismos de tratamento de interrupções implantados nos sistemas distribuídos.

3.1 MAPEAMENTO

A integração dos modelos analíticos (ex.: redes de Petri, cadeia de Markov ou redes de fila) com os modelos semiformais, tais como UML ou SysML é muito importante. Modelos analíticos são apoiados por fundamentos matemáticos sólidos, que suportam sua semântica precisa, estimulam a avaliação de desempenho/dependabiliadde e fornecem suporte para verificações de propriedades qualitativas e análises. Esses modelos formais, no entanto, não são intuitivos e requerem um considerável esforço por parte dos projetistas para entenderem a notação usada. Por outro lado, os modelos semiformais têm sido largamente adotados no ciclo de vida de desenvolvimento de sistema, devido principalmente à sua notação amigável e intuitiva. Assim, é sensato adotar o uso colaborativo dos modelos semiformais e formais. Dentre as linguagens semiformais de modelagem usadas para especificar sistemas, UML é a mais amplamente utilizada pela comunidade de engenharia de *software* e a indústria. A essência da UML, no entanto, é modelar sistemas baseados em *software* [AMN+10, Gue08]. Quando se está modelando sistemas distribuídos, nos quais é necessário se ter uma abordagem interdisciplinar envolvendo as mais diversas áreas (*software*, mecânica, elétrica e eletrônica), SysML é considerada mais adequada para modelar esses tipos de sistemas heterogêneos, pois ela foi desenvolvida para esse propósito.

Em [AMN+10, AMCN09, AMC+08, AMC+09], Andrade *et al.* descrevem o processo de mapeamento dos diagramas da SysML, anotados de acordo com o *profile* MARTE, em redes de Petri temporizadas para análises e verificações dos sistemas embarcados de tempo-real. Essa abordagem, no entanto, apenas foca na análise de caminhos para o tempo de execução e consumo de energia. Pooley and King [KP99, KP00] apresentam

como o Diagrama de Colaboração da UML (UML-CD) em combinação com o Diagrama de Estados da UML (UML-STM) podem ser sistematicamente transformados em uma rede de Petri estocástica (do inglês *Stochastic Petri Nets* - SPN). Nessa abordagem, os UML-STMs são transformados em modelos SPNs, e estes são combinados através do UML-CD, formando um único modelo. Os estados do UML-STM são mapeados em lugares no modelo SPN e as transições são mapeadas em transições no modelo SPN. Em resumo, uma abordagem intuitiva é apresentada.

Em [TZ05, TZ06, TZH05], Trowitzsch *et al.* abordam a derivação do UML-STM em uma SPN para avaliação de desempenho de sistemas de tempo-real. Essa abordagem consiste na decomposição do UML-STM em elementos básicos, tais como estados, transições e elementos especiais (ex.: sincronização entre regiões e variável contador). Esses elementos, por sua vez, são mapeados em representações SPNs correspondentes, considerando tempos exponenciais e determinísticos. O UML SPT [OMG03] é usado como linguagem de especificação para as restrições dos sistemas de tempo-real. Esse trabalho aparenta ser muito interessante, porém os autores não levam em consideração os aspectos de disponibilidade dos sistemas.

Messeguer *et al.* [MCBD02] apresentam como obter um modelo GSPN (*Generalized Stochastic Petri Net*) a partir da descrição de um sistema, expresso por um conjunto de UML-STMs. A abordagem consiste em três passos básicos: primeiramente, os elementos do UML-STM, tais como atividades internas (*entry*, *do* e *exit*), transições internas, eventos deferidos, estado inicial e final são mapeados em modelos GSPN; após isso, essas GSPNs são compostas para gerar um único modelo GSPN representando cada UML-STM; por fim, os models GSPN, representando cada UML-STM são compostos para formar um único modelo GSPN representando todo o sistema. Lopez-Grao [LGMC04] *et al.* discutem a integração do diagrama de atividades da UML (UML-AD) com o UML-STM para aplicações na Engenharia de Desempenho de *Software* (*Software Performance Engineering*). Eles propõem a combinação dos modelos GSPN obtidos a partir dos UML-ADs com os modelos GSPN obtidos a partir dos UML-STMs. A combinação dessas GSPNs é feita através da superposição de transições e lugares. Os autores afirmam que o modelo final (combinado) aumenta a expressividade e permite a avaliação de sistemas complexos, uma vez que o UML-STM modela a comunicação entre os componentes do sistema, enquanto o UML-AD modela o processamento interno dentro desses componentes. Em [BDM02], o diagrama de sequência da UML (UML-SD) é usado em conjunto com o UML-STM para avaliação de desempenho de sistemas. Essa abordagem é semelhante a abordagem apresentada em [LGMC04], porém o UML-SD é usado ao invés do UML-AD. Campos *et al.* [CM06] apresentam um estudo de caso, onde os diagramas da UML (UML-STM, UML-AD e UML-SD) são usados em conjunto para descrever as características do sistema. Para cada diagrama, as regras de mapeamento descritas nos artigos anteriores são usadas [MCBD02, LGMC04, BDM02]. Os modelos GSPN finais são usados para a avaliação de desempenho do sistema modelado. Embora essas abordagens pareçam ser interessantes, os autores focam apenas em desempenho.

Embora esta pesquisa foque na análise quantitativa (ex.: disponibilidade, *downtime*

e custo), existem algumas abordagens que almejam análises qualitativas [BP01b, BP01a, DdSS04, LPK+00]. Em [BP01b, BP01a], Baresi e Pezzé apresentam um conjunto de regras para traduzir uma especificação descrita em UML para uma rede de Petri predicado/transição. Nessa abordagem, os autores propõem a transformação do diagrama de classe em conjunto com o UML-STM em uma rede de Petri predicado/transição. Para cada método de uma classe, é criado um par de lugares de rede de Petri. Um dos lugares indica a requisição do método e o outro indica o retorno do método após sua execução. Além disso, também são identificadas as chamadas externas que cada classe realiza. Para isso, são acrescentados pares de lugares correspondentes às chamadas efetuadas pela classe. Após isso, os estados do UML-STM são mapeados em lugares da Rede de Petri e as transições são mapeadas em transições de redes de Petri. Então, os modelos de estados que representam cada classe são integrados, a fim de obter-se uma única rede de Petri. Por fim, simulações, análises de alcançabilidade e verificações do modelo são realizadas.

Similarmente, em [DdSS04], o autor apresenta uma abordagem parecida com o trabalho de Baresi, na qual é proposta a transformação do diagrama de classe em conjunto com o UML-STM em uma rede de Petri predicado/transição. No entanto, se por um lado Baresi e Pezzé empregam regras de mapeamento para traduzir os UML-STM desenhados pelo projetista em redes de Petri, em [DdSS04] os autores sugerem que o comportamento de cada classe seja diretamente descrito pelo projetista por meio de uma rede de Petri. Já em [LPK+00], Lee *et al.* propõem a derivação de cenários em redes de Petri temporizadas. Nessa abordagem, o diagrama de caso de uso é usado para elicitar os requisitos e criar os cenários. Os cenários são representados pelos diagramas de sequência. Uma vez criados os cenários, eles são transformados em redes de Petri temporizadas, em que a linha de vida de uma entidade (*lifeline*) é representada por dois lugares e os eventos de chegada e saída na linha de vida são representados por transições. As análises são realizadas de forma a encontrar informações erradas ou esquecidas nos modelos gerados.

Além disso, existem outras abordagens que almejam a transformação de modelos semiformais para modelos formais, no entanto a UML ou SysML não é utilizada como linguagem de especificação [AMN+05, AMN+06, VCF+06]. Em [AMN+05, AMN+06], Amorim *et al.* propõem um conjunto de passos para o mapeamento do *Live Sequence Chart* (LSC) em uma representação *Coloured Petri Net* (CPN) [JKW07] equivalente. A LSC permite especificar comportamentos proibidos (anti-cenários), bem como modelar o que deve ocorrer. O objetivo desse trabalho é realizar análises e verificações das propriedades dos sistemas embarcados. Essas análises e verificações são realizadas através do CPN *Tools* [RWL+03]. Por fim, Vijaykumar *et al.* [VCF+06] apresentam o processo de transformação dos *Statechart* [Har87] em uma cadeia de *Markov*.

Embora o processo de mapeamento das linguagens semiformais para os modelos analíticos já tenham sido bastante discutidos na literatura, o mapeamento apresentado neste trabalho difere em vários aspectos. (i) A abordagem proposta foca no mapeamento dos diagramas da SysML para análise de disponibilidade. (ii) Anotações de alocação são definidas para lidar com as possíveis dependências dos elementos do sistema (ex.:

data center e máquinas virtuais). (iii) O comportamento dinâmico dos mecanismos de tratamento de interrupções é modelado através dos diagramas de atividades, sendo este incorporado ao modelo de disponibilidade. (iv) Este trabalho define uma abordagem hierárquica para lidar com os problemas de complexidade tanto a nível dos diagramas da SysML, quando a nível dos modelos DSPN.

3.2 AVALIAÇÃO DE DISPONIBILIDADE

Apesar de existirem vários estudos que propõem o mapeamento de especificações de alto nível para modelos analíticos a fim de realizar análises qualitativas e quantitativas dos sistemas de informação, poucos trabalhos têm focado na avaliação de disponibilidade dos sistemas distribuídos, e nenhum deles tem considerado a combinação dos diagramas da SysML, das anotações de MARTE e dos modelos DSPN para o estudo de diferentes mecanismos de tratamento de interrupções (ex.: recuperação de desastre, função de auto *scaling*, entre outros) implantados nos sistemas distribuídos (ex.: computação em nuvens, virtualização, entre outros).

A forma mais comum dos provedores de serviços em nuvem avaliar questões de disponibilidade de seus serviços é baseada em dados empíricos através de painéis (*dashboards*) [Eck10]. Essa abordagem não é adequada, pois as estatísticas obtidas de tais painéis não garantem a disponibilidade no futuro. Dessa forma, o uso de modelos analíticos para avaliar os *trade-offs* das infraestruturas em nuvens e responder questões do tipo "o que aconteceria se" são essenciais. Alguns poucos estudos têm endereçado a avaliação disponibilidade de serviços em nuvem. Em [JJH+10], os autores apresentam uma técnica de regeneração dinâmica para os componentes de *software* de um sistema utilizando virtualização. Essa técnica propõe restaurar a redundância do sistema sempre que ocorrem falhas, utilizando os recursos remanescentes de modo que os objetivos de disponibilidade e desempenho sejam sempre atendidos. Addis *et al.* [AAPZ10] estudam a gestão dinâmica de recursos em infraestrutura de serviços em nuvem sob restrições de disponibilidade. Em [ZZLK10], Zheng *et al.* propõem um *framework* denominado de FTCloud, que almeja ranquear as melhores técnicas de tolerância a falha para a construção de aplicativos em nuvem. Embora esses trabalhos apresentem abordagens interessantes, elas não cobrem uma abordagem na qual os projetista possam analisar diferentes cenários, analiticamente, a partir do conhecimento deles.

Desastres naturais ou desastres causados pelo homem, tais como inundações, furacões, tornados, terremotos ou terrorismo podem atingir as empresas em qualquer lugar, a qualquer momento, com pouco ou nenhum aviso. Para evitar prejuízos ainda maiores devido a tais eventos, a maioria das grandes empresas gastam em torno de 2% a 4% do seu orçamento na área de TI em mecanismos de recuperação de desastres [Smi03]. A fim de diminuir o *downtime* ocasionado por tais desastres, vários trabalhos têm sido desenvolvidos ao longo dos anos, focando em soluções de recuperação de desastre. Wood *et al.* [WCR+10] propõem o uso de plataformas de computação em nuvens como um serviço de recuperação de desastre devido ao modelo de pagamento (*pay-as-you-go*) que pode redu-

zir os custo, assim como aumentar a capacidade de recuperação dos recursos após uma catástrofe. Em [RCOW12], Rajagopalan *et al.* apresentam o *design* e implementação de *SecondSite*, que consiste de um serviço de tolerância a desastres baseado numa infraestrutura nas nuvens. *SecondSite* permite que as máquinas virtuais em execução sejam replicadas em localizações geograficamente dispersas. Em [LTY09], os autores propõem um modelo de rede de Petri representando uma solução de recuperação de desastre local. O modelo proposto leva em consideração virtualização e rejuvenescimento de *software*, a fim de diminuir os danos causados pelos desastres. A quantidade de soluções para o tratamento de interrupções causadas por desastres é realmente muito grande. Assim, a análise quantitativa de parâmetros, tais como disponibilidade do sistema e custo relacionado às operações de recuperação de interrupções, podem prover importantes informações para os projetistas relacionadas à melhor solução para a empresa. O trabalho proposto nesta pesquisa, diferentemente das abordagens apresentadas anteriormente, propõe um *framework* que permite aos projetistas criarem e analisarem diferentes mecanismos de recuperação de desastres implantados nos sistemas distribuídos através de uma linguagem de alto nível descrita através de SysML e MARTE.

A disponibilidade dos sistemas distribuídos também pode ser afetada por interrupções brandas, como as falhas causadas pelos os *bugs* de envelhecimento. Essas falhas ocorrem, principalmente, devido ao acumulo de defeitos no sistema [TGA10]. Os mecanismos de tratamento de interrupção utilizados devido a esse fenômeno de envelhecimento são chamados de rejuvenescimento de *software*. Modelos analíticos que descrevem o rejuvenescimento de *software* têm sido extensivamente desenvolvido nos últimos anos. O primeiro trabalho a estudar tal comportamento através de SPNs foi desenvolvido por Garg *et al.* [GPTT95]. Nesse trabalho, os autores desenvolveram um mecanismo de rejuvenescimento baseado em tempo, onde o tempo ótimo de rejuvenescimento foi computado. Vaidyanathan *et al.* [VHHT01] apresentaram a avaliação de sistemas em *cluster* com rejuvenescimento de *software* usando SRNs (*Stochastic Reward Nets*). Esse trabalho levou em consideração as técnicas de rejuvenescimento baseado em tempo e em condição. Em [DGPT00], Dohi *et al.* derivam analiticamente o tempo ótimo de rejuvenescimento, a fim de maximizar a disponibilidade do sistema, em que o comportamento de envelhecimento do *software*, bem como o rejuvenescimento são modelados através de processos semi-Markovianos. Embora esses modelos sejam úteis para evitar as interrupções devido aos *bugs* de envelhecimento, os mesmos não são fáceis de serem obtidos pelos projetistas, os quais não possuem conhecimento em modelagem estocástica.

3.3 FERRAMENTAS

Para que o mapeamento dos diagramas da SysML em DSPNs seja mais praticável, é necessário o suporte de ferramentas apropriadas. As atividades que necessitam do apoio de uma ferramenta são: (i) modelagem dos sistemas distribuídos através dos diagramas da SyML e das anotações de MARTE, (ii) mapeamento dos diagramas anotados em DSPNs correspondentes, e (iii) análise dos modelos resultantes. Para a parte de modelagem, existem várias ferramentas comerciais, tais como Papyrus, Visual Paradigm for UML e

Topcased. Papyrus [GDTS11] é uma ferramenta de código aberto (*opensource*), que é baseada no ambiente Eclipse, e está sob a licença EPL (*Eclipse Public License*). Essa ferramenta oferece suporte para todos os diagramas da SysML e anotações de MARTE, a fim de permitir a modelagem de sistemas complexos. Visual Paradigm for UML [Par10] é uma ferramenta CASE com várias opções de modelagem que suporta tanto os diagramas da SysML, quanto os diagramas da UML. Ela é uma ferramenta comercial que também oferece suporte à geração automática de código a partir dos modelos de alto nível descritos em SysML ou UML. Topcased [FGC+06] é um ambiente dedicado principalmente ao estudo dos sistemas críticos que suporta a especificação dos mesmos através dos diagramas da SysML.

Além disso, existem várias ferramentas que apoiam a análise de modelos DSPN [SMT+10, ZKHH06, STP96, CMT89]. ASTRO [SMT+10] é uma ferramenta de modelagem de dependabilidade e sustentabilidade, que permite a utilização de diagramas de bloco de confiabilidade (RBD), redes de Petri estocásticas (SPN) e modelos de alto nível para avaliação de infraestruturas *data center*. Essa ferramenta também permite a avaliação de dependabilidade de sistemas genéricos para usuários que conheçam princípios de modelagem através de SPN e RBD. TimeNet (*Timed Net Evaluation Tool*) [ZKHH06] é uma ferramenta de modelagem desenvolvida pela Technische Universitat de Berlin, a qual suporta modelos de rede de Petri estocásticas incluindo as DSPNs. SHARPE (*Symbolic Hierarchical Automated Reliability and Performance Evaluator*) [STP96] e SPNP (*Stochastic Petri Net Package*) [CMT89] são ferramentas desenvolvidas pela Universidade de Duke nos EUA. Essas ferramentas têm sido usadas para análise de desempenho, dependabilidade e performabilidade de uma grande variedade de sistemas.

Algumas ferramentas também foram desenvolvidas para apoiar o uso integrado de modelos semiformais e modelos analíticos [DPP+05, GMM06, TJZ07, RKK08, AANM12]. Essas ferramentas são muito importantes, pois podem evitar erros que podem ser incluídos durante o mapeamento manual, bem como, diminuir o tempo para realizar tal mapeamento, já que o mesmo é realizado automaticamente por uma ferramenta. Em [DPP+05], os autores apresentam uma ferramenta chamada de ArgoPerformance que permite o mapeamento automático de diagramas da UML anotados em SPNs para avaliação de desempenho. Semelhantemente Gómez-Martínez *et al.* [GMM06] apresentaram uma ferramenta chamada de ArgoSPT, que permite o mapeamento automático de diagramas da UML anotados em SPNs. As funcionalidades dessa ferramenta são baseadas na teoria dos trabalhos publicados anteriormente [MCBD02, LGMC04, BDM02]. Trowitzsch *et al.* [TJZ07] apresentaram uma ferramenta que automaticamente gera modelos SPN a partir dos diagrama de estados anotados da UML. Essa ferramenta também implementa as funcionalidades baseadas na teoria descrita nos trabalhos anteriores [TZ05, TZ06, TZH05]. Em [RKK08], os autores descrevem a ferramenta ADAPT que permite o mapeamento de Linguagem de Projeto e Análise de Arquitetura (AADL) em GSPNs. Andrade *et al.* [AANM12] apresentam uma ferramenta denominada Calau. Essa ferramenta auxilia os projetistas no mapeamento do SysML-STM, anotado de acordo com o *profile* MARTE, em uma rede de Petri temporizadas com o intuito de realizar análises de caminhos críticos para o tempo de execução e o consumo de energia.

Diferentemente dos trabalhos apresentados anteriormente, para o melhor do nosso conhecimento, não existe nenhuma ferramenta que suporte o uso integrado dos diagramas da SysML, das anotações de MARTE e das DSPNs para análise de dependabiliade dos sistemas distribuídos. Além disso, o protótipo desenvolvido neste trabalho chamado de OpenMADS [AAM+13] é hospedado na *Web*, assim, elimina o custo de instalar, atualizar e configurar a ferramenta por parte dos usuários. Ademais, OpenMADS pode ser o usado para modelar sistemas de propósito geral. Mais informações relacionadas a ferramenta desenvolvida ver o Capítulo 5.

3.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os principais trabalhos correlatos ao estudo proposto. Embora existam vários trabalhos na literatura que propõem o mapeamento de linguagens de alto nível para modelos analíticos, nenhum desses trabalhos foca na análise de dependabilidade dos sistemas distribuídos, considerando mecanismos de tratamento de interrupções. Além disso, vale ser ressaltado que também existem vários trabalhos que objetivam a análise de disponibilidade dos sistemas de informação de forma em geral, porém, para o melhor do nosso conhecimento, nenhum deles permite que os projetistas, os quais não possuem (ou possuem pouca) expertise em modelagem estocástica, possam se beneficiar da abordagem proposta.

CAPÍTULO 4

MAPEAMENTO DOS DIAGRAMAS DA SYSML EM DSPN

A person who never made a mistake never tried anything new.
—ALBERT EINSTEIN

Este capítulo descreve o *framework* proposto, o qual é composto por um conjunto de passos, indo desde a modelagem das especificação de alto nível, passando pelo mapeamento dessas especificações, até a análise dos modelos analíticos obtidos. Isto é, esse *framework* permite que os projetistas, os quais não possuem (ou possuem pouca) expertise em modelagem estocástica, possam estudar os mecanismos de tratamento de interrupções e as infraestrutura dos sistemas distribuídos quantitativamente, a partir de uma linguagem de alto nível, descrita através dos diagramas anotados da SysML. Dentre os diagramas da SysML, os diagramas de estados, atividades e bloco interno foram adotados neste trabalho, visto que eles são ideais para representar as infraestruturas dos sistemas distribuídos (SysML-STM e SysML-IBD) e os mecanismos de tratamento de interrupções (SysML-AD). No entanto, durante o processo de mapeamento, notou-se que, em alguns casos, existem dependências entre os elementos dos diagramas (ex.: hospedagem, *standby* etc.). Dessa forma, criou-se um método para compor e sincronizar os modelos obtidos pelo processo de mapeamento, a fim de representar tais dependências. Uma vez definido todo esse processo de mapeamento, também verificou-se que os diagramas da SysML e as redes de Petri geradas por tal processo podem ser grandes e complexos. Assim, foi criada uma abordagem hierárquica para aliviar tais problemas, tanto em nível dos diagramas da SysML (ex.: estruturas compostas) quanto em nível das redes de Petri (ex.: modelos de alto nível combinados com modelos baixo nível). Adicionalmente, este capítulo apresenta a análise de propriedades qualitativas dos modelos gerados pelo processo de mapeamento.

4.1 VISÃO GERAL

Este trabalho é baseado no esquema apresentado na Figura 4.1. As ferramentas criadas e/ou utilizadas deverão dar suporte à modelagem e à aferição das métricas dos modelos DPSN obtidos pelo processo de mapeamento. Os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos são modeladas usando os diagramas da SysML e as anotações do *profile* MARTE. Tais diagramas anotados são mapeados em componentes do modelo DSPN de disponibilidade. Os componentes, por sua vez, são sincronizados e combinados para formar um modelo DSPN único, que é utilizado para calcular as métricas e os indicadores selecionados para estudar tanto os mecanismos de tratamento de interrupções, quanto as infraestruturas distribuídas [AMKT11, MAKT11].

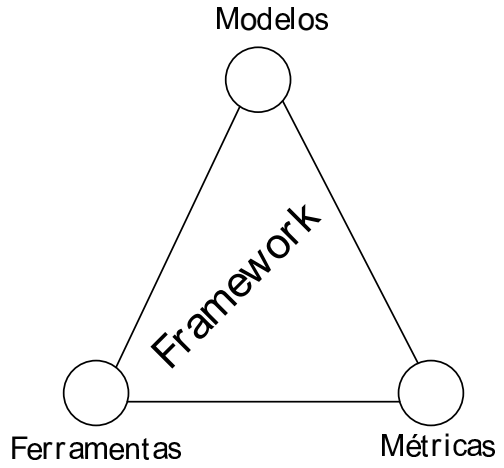


Figura 4.1: Esquema Simplificado do Trabalho Proposto.

Os diagramas da SysML e os modelos DSPN gerados pelo processo de mapeamento podem ser grandes e complexos. Dessa forma, propõem-se o uso de uma abordagem hierárquica para aliviar tais problemas. Essa abordagem pode ser aplicada tanto aos diagramas da SysML, quando aos modelos DSPN. Com relação aos diagramas da SysML, as estruturas compostas dos diagramas de atividades e estados podem ser usadas a fim de decompor os modelos em partes menores, seguindo o princípio de dividir para conquistar [PP05]. Note que essas estruturas têm o objetivo de prover uma melhor organização/divisão dos elementos dos diagramas. Consequentemente, podem ajudar a diminuir a complexidade e o tamanho dos modelos, facilitando todo o processo de mapeamento e análise.

Por outro lado, para evitar problemas de *largeness* (explosão do espaço de estados) [STP96] e *stiffness* (parâmetros de magnitudes muito diferentes) [MMT94] dos modelos DSPN, uma abordagem ideal consiste na utilização de modelos hierárquicos. Isto é, os modelos de disponibilidade que representam tanto os mecanismos de tratamento de interrupções quanto as infraestruturas dos sistema distribuídos podem ser representadas por modelos sem espaço de estados (ex.: diagrama de bloco ou árvore de falhas) em conjunto com os modelos de espaço de estados, e.g., redes de Petri ou cadeia de Markov. Os modelos sem espaço de estados têm sido largamente usados devido à sua eficiência e simplicidade. Porém, esses modelos são inadequados para capturar todas as dependências de falha e reparo que são inerentes aos sistemas distribuídos. Já os modelos de espaço de estados podem considerar essas dependências, permitindo, assim, a representação de mecanismos de tratamento de interrupções e infraestruturas distribuídas sofisticadas. No entanto, tais modelos de espaço de estados podem não ser efetivos devido à explosão de estados [TAM12]. A solução adotada neste trabalho consiste na partição dos modelos complexos em submodelos independentes[1], em que esses submodelos são analisados e os resultados são utilizados como entrada para o modelo de mais alto nível, que, por sua vez, calcula a disponibilidade de todo o sistema.

[1]Submodelos independentes são modelos na qual a execução de um não afete a execução de outro.

4.2 FRAMEWORK PROPOSTO

A Figura 4.2 apresenta o *framework* do trabalho proposto. Esse *framework* suporta três tipos de diagramas da SysML: diagrama de bloco interno (SysML-IBD), diagrama de estados (SysML-STM) e diagrama de atividades (SysML-AD). O SysML-IBD é utilizado para representar as configurações estáticas dos sistemas distribuídos. O SysML-STM é usado para descrever as transições de estados de um elemento específico dos sistemas distribuídos. O SysML-AD é usado para representar o fluxo de atividades dos mecanismos de tratamento de interrupções. Um subconjunto das anotações do *profile* MARTE também é usado para representar os aspectos quantitativos das infraestruturas modeladas (ver Seção 2.4 do Capítulo 2). As dependências entre os elementos dos diagramas são especificadas tanto por anotações denominadas *alocações*, quanto por propriedades dos blocos.

O processo de modelagem e avaliação do *framework* é dividido em 7 passos básicos: (i) modelar os sistemas usando os diagramas da SysML e as anotações de MARTE, (ii) mapear os diagramas anotados, (iii) compor os modelos DSPN, (iv) sincronizar os modelos DSPN, (v) calcular a disponibilidade dos submodelos, (vi) agregar as disponibilidade usando RBD e (vii) calcular as métricas de interesse. No primeiro passo, os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos são modelados usando os diagramas da SysML e as anotações de MARTE. Em seguida, esses diagramas são mapeados em modelos DSPN de disponibilidade, denominados de componentes do modelo. Os componentes do modelo descritos através dos SysML-IBDs representam os elementos dos sistemas distribuídos, enquanto os componentes do modelo descritos através dos SysML-STMs representam as transições de estados desses elementos. Já o modelo gerado a partir do SysML-AD, o qual é chamado de *Rede de Atividades*, representa o fluxo das atividades dos mecanismos de tratamento de interrupções.

No terceiro passo, faz-se a composição dos componentes dos modelos gerados a partir dos SysML-IBDs e SysML-STMs, de acordo com as anotações de alocações. O modelo resultante da composição é chamado de *Rede do Sistema*. A *Rede do Sistema* representa as configurações dos sistemas distribuídos. Note que as marcações da *Rede do Sistema* podem ser afetadas pelos mecanismos de tratamento de interrupções descritos através dos SysML-ADs. Por exemplo: um servidor representado através de uma *Rede do Sistema* pode mudar do estado *down* para o estado *up* (operacional) através da execução de uma atividade do mecanismo de tratamento de interrupção. Dessa forma, é necessário sincronizar a *Rede de Atividades* com a *Rede do Sistema*.

No quarto passo, a *Rede de Atividades* é sincronizada com a *Rede do Sistema* através da identificação dos relacionamentos entre as atividades da *Rede de Atividades* e as transições de estados da *Rede do Sistema*. Note que o *framework* proposto auxiliará os projetistas nesse processo, bem como na definição das funções de guarda através do uso de anotações, tais como ≪controle≫, ≪decisionInput≫ e expressões booleanas (ex.: {Disp="Falso"} ou {Disp="Verdade"}). Após isso, o projetista verifica se existem submodelos independentes, visto que os modelos DSPN obtidos pelo processo de mapea-

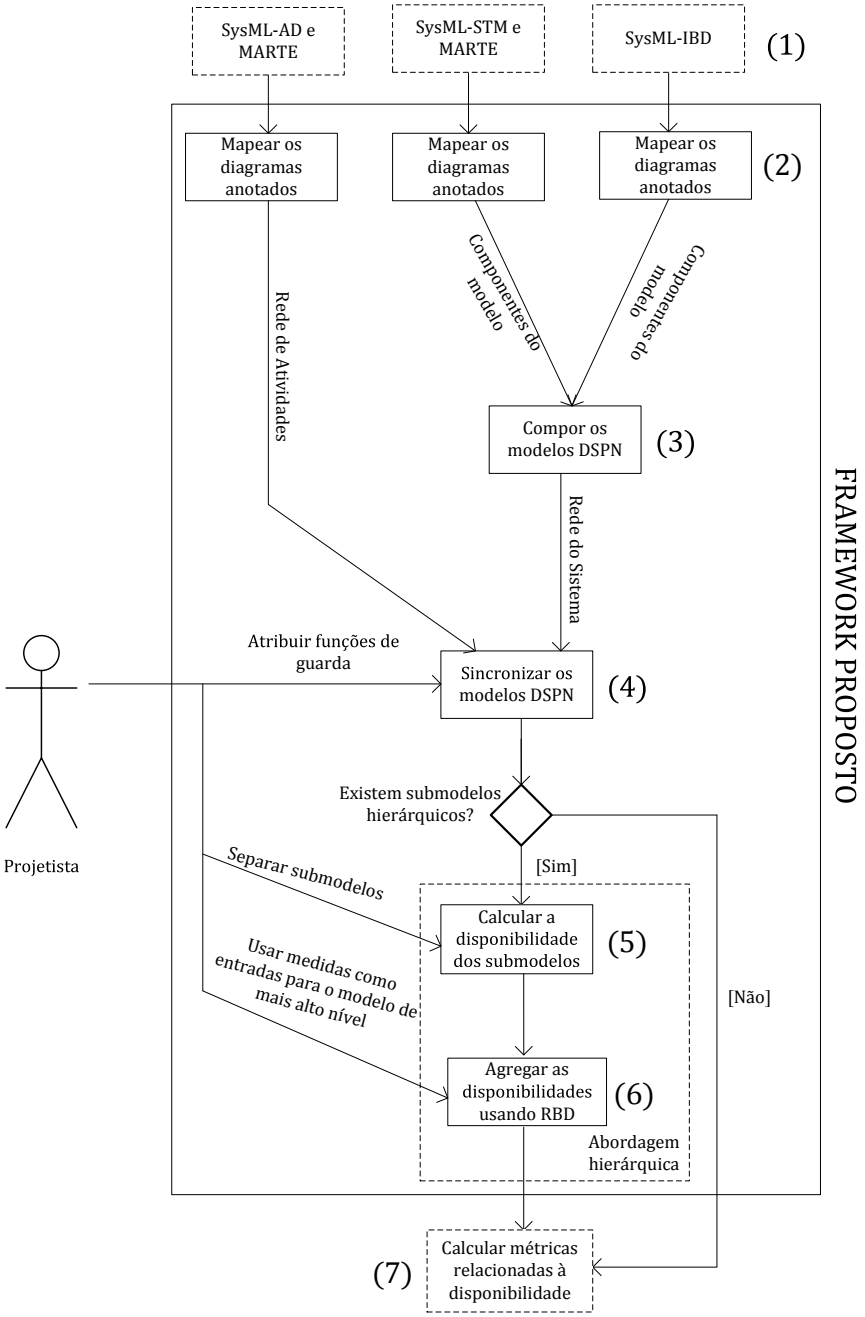


Figura 4.2: *Framework* Proposto.

mento, composição e sincronização podem ser grandes e complexos, resultando em problemas de *largeness* e *stiffness*. Caso não exista, as métricas relacionadas a disponibilidade (ex.: *downtime*, disponibilidade estacionária, custo etc.) são calculadas usando a ferramenta desenvolvida, chamada de OpenMADS [AAM+13], ou usando outras ferramentas disponíveis no meio acadêmico, e.g., TimeNet [ZKHH06], ASTRO [SMT+10], SHARPE [STP96] ou SPNP [CMT89]. Caso exista, é utilizada uma abordagem hierárquica, em que

os modelos são separados em submodelos independentes e as disponibilidades são calculadas para cada um deles. Após isso, o diagrama de bloco de confiabilidade é usado para agregar as disponibilidade dos subsistemas a fim de calcular a disponibilidade de todo o sistema. A ferramenta desenvolvida, OpenMADS, não suporta o uso do diagrama de bloco de confiabilidade, sendo necessário o uso de outras ferramentas que disponibilizam tal recurso (ex.: ASTRO ou SHARPE). Por fim, no passo 7, as métricas de interesse são calculadas. Todos os passos do *framework* proposto são descritos em detalhes abaixo.

4.3 MAPEAMENTO DOS DIAGRAMAS DA SYSML EM MODELOS DSPN

Esta seção detalha o processo de mapeamento dos diagramas da SysML, anotados de acordo com o *profile* MARTE, em uma rede de petri determinística e estocástica.

4.3.1 Mapeamento do SysML-IBD

Cada bloco do SysML-IBD é mapeado em dois lugares e duas transições representando o comportamento de falha e reparo do elemento do sistema no qual o bloco representa. A Figura 4.3 apresenta um exemplo desse mapeamento, em que o bloco Servidor *Web* é mapeado em um componente do modelo. No componente do modelo (ver Figura 4.3 (b)), a operacionalidade do servidor é indicada por um token no lugar P_{up}. Já a indisponibilidade do servidor é dada por um token no lugar P_{down}. As transições T_{falhar} e $T_{reparar}$ denotam, respectivamente, a falha e o reparo do servidor *Web*. Isto é, a transição T_{falhar} dispara quando o servidor se torna *down* (indisponível), e um token de P_{up} é removido e um token em P_{down} é depositado. A transição $T_{reparar}$ dispara quando o servidor é reparado, então, um token de P_{down} é removido e um token em P_{up} é depositado. Os tempos atribuídos às transições são baseados nas propriedades dos blocos (ver o compartimento *values*). O MTTF representa tempo médio até a falha, enquanto o MTTR representa o tempo médio de reparo.

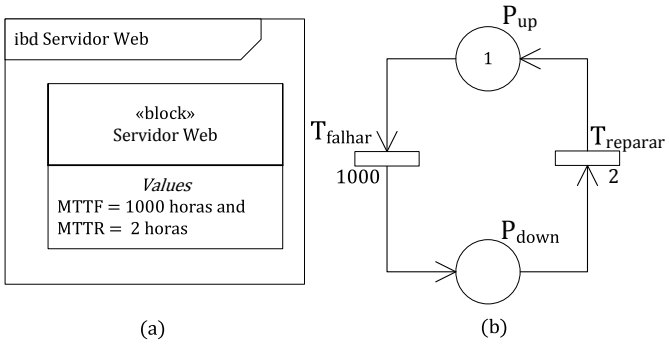


Figura 4.3: Mapeamento do SysML-IBD.

A Figura 4.4 (a) apresenta um exemplo de um sistema de aplicação *Web* composto por um balanceador de cargas, três servidores *Web* e um servidor banco de dados. Cada bloco é mapeado de acordo com as regras apresentadas anteriormente. Isto é, eles são mapeados

em lugares e transições representando o comportamento de falha e reparo do elemento do sistema (ver Figura 4.4 (b)). Note que os blocos são considerados independentes uns dos outros. O número "3"no cabeçalho do bloco Servidor *Web* representa o nível de redundância do componente. Esse nível de redundância é mapeado no número de tokens no lugar P_{SWup} do componente do modelo.

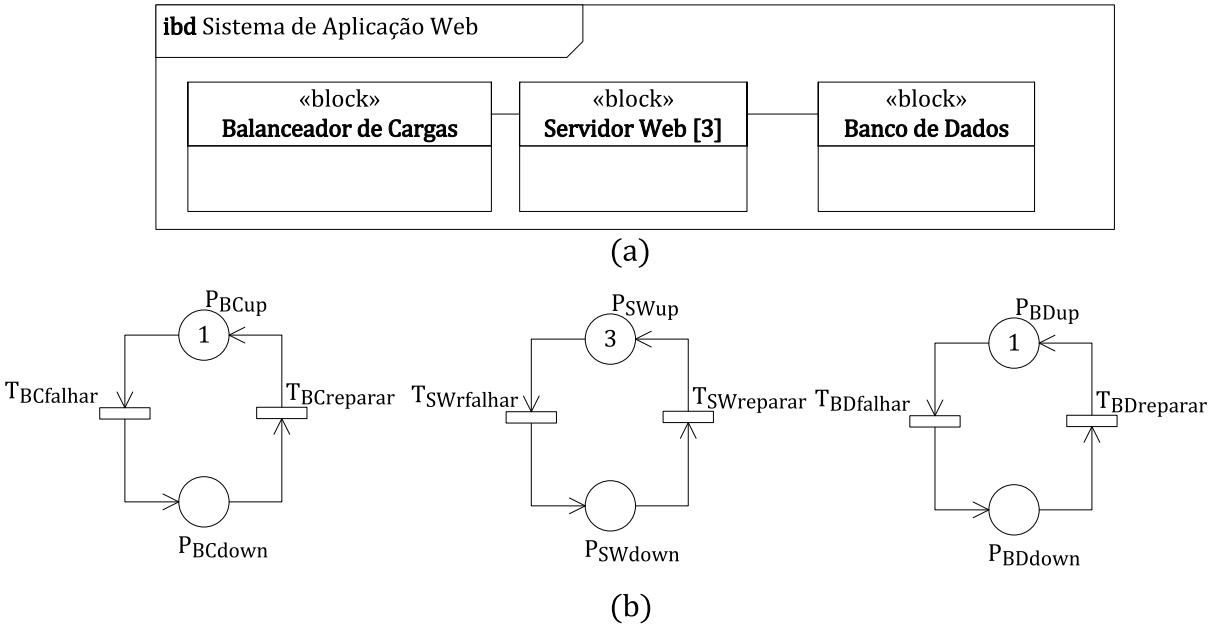


Figura 4.4: Mapeamento do Exemplo de um Sistema de Aplicação Web.

4.3.2 Mapeamento do SysML-STM

A Figura 4.5 (a) apresenta um exemplo do comportamento de falha e reparo de um servidor *Web* descrito através do SysML-STM. O servidor inicia no estado *Desligado*. Uma vez nesse estado, o comando de inicialização pode ser invocado e, então, o servidor assume o estado *Operational*. A partir desse estado, o servidor pode falhar. Se o mesmo falhar, o servidor assume o estado *Falhou*. Quando a falha for detectada, então o servidor assume o estado *Detectado*. Em seguida, o servidor é reparado e retorna para o estado *Operational*. Por fim, o servidor pode ser desligado através do evento *Desligar*. A Figura 4.5 (a) também ilustra o uso das anotações de disponibilidade especificadas por MARTE. O estereótipo *PAStep* e o valor marcado *HostDemand* são usados. O estereótipo descreve uma ação, enquanto o valor marcado representa o nome da propriedade e o valor atribuído à mesma. Por exemplo, a transição *Falhar* é anotada com o estereótipo *PAStep*, significando que o evento de falha tem um tempo exponential de 2000 horas. O valor marcado é dado por $\{HostDemand = (exp(2000), h)\}$. A propriedade booleana "*Disp*"foi definida para especificar o estado de disponibilidade dos elementos do sistema. O valor dessa propriedade é definida como "*verdade*"apenas quando o sistema funcionar corretamente no estado. No exemplo, o estado *Operacional* é o único estado na qual o

servidor está disponível. Além disso, para especificar as transições dos SysML-STMs que são afetadas pelas atividades dos mecanismos de tratamento de interrupções, os quais são descritos através dos SysML-AD, foi introduzido um novo estereótipo denominado ≪controle≫. Na Figura 4.5 (a), por exemplo, o estado do servidor mudará de *Operacional* para *Desligado* através da execução de uma atividade do mecanismo de tratamento de interrupções. Esse processo de sincronização é detalhada na Seção 4.3.5.

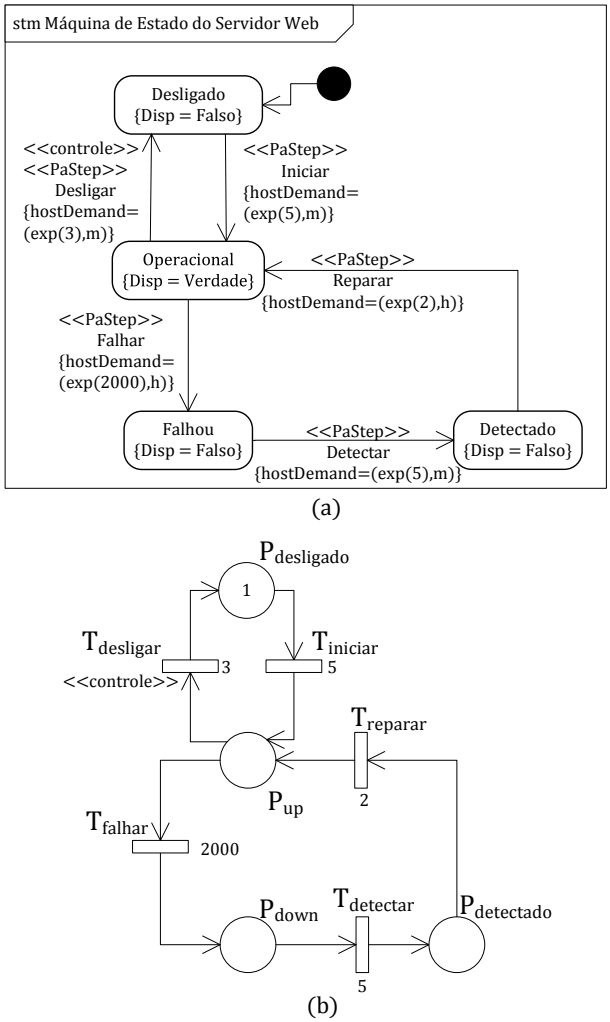


Figura 4.5: Mapeamento do SysML-STM.

O modelo DSPN do SysML-STM pode ser obtido através da conversão de cada elemento do diagrama de estados em respectivos elementos da DSPN, em que cada estado é mapeado em um lugar e cada transição anotada do SysML-STM é mapeada em uma transição temporizada com arcos de entrada e saída. A Figura 4.5 (b) representa a DSPN do comportamento de falha e reparo do servidor *Web* descrito na Figura 4.5 (a). O servidor inicia no estado *Desligado*, indicado por um token no lugar $P_{desligado}$. Quando o comando de inicialização é invocado, a transição $T_{iniciar}$ é disparada e um token de $P_{desligado}$ é removido e um token em P_{up} é depositado. A transição T_{falhar} é disparada

quando o servidor se tornar indisponível (*Falhou*) e, então, um token de P_{up} é removido e um token em P_{down} é depositado. $T_{detectar}$ é disparado quando a falha é detectada, resultando na remoção de um token de P_{down} e no depósito de um token em $T_{detectado}$. Após a falha ter sido detectada, o servidor pode ser reparado através do disparado da transição $T_{reparar}$. Consequentemente, um token de $T_{detectado}$ é removido e um token em P_{up} é depositado. Por fim, o servidor pode ser desligado através do disparo da transição $T_{desligar}$. Note que as transições $T_{iniciar}$, $T_{desligar}$, T_{falhar}, $T_{detectar}$ e $T_{reparar}$ são atribuídas com tempos exponenciais de 5 minutos, 3 minutos, 2000 horas, 5 minutos e 2 horas, respectivamente. Essas atribuições são baseadas nas anotações de MARTE. Além disso, a transição $T_{desligar}$ também é atribuída com o estereótipo ≪controle≫, significando que a execução de uma atividade do mecanismo de tratamento de interrupções irá resultar na mudança de estado do servidor.

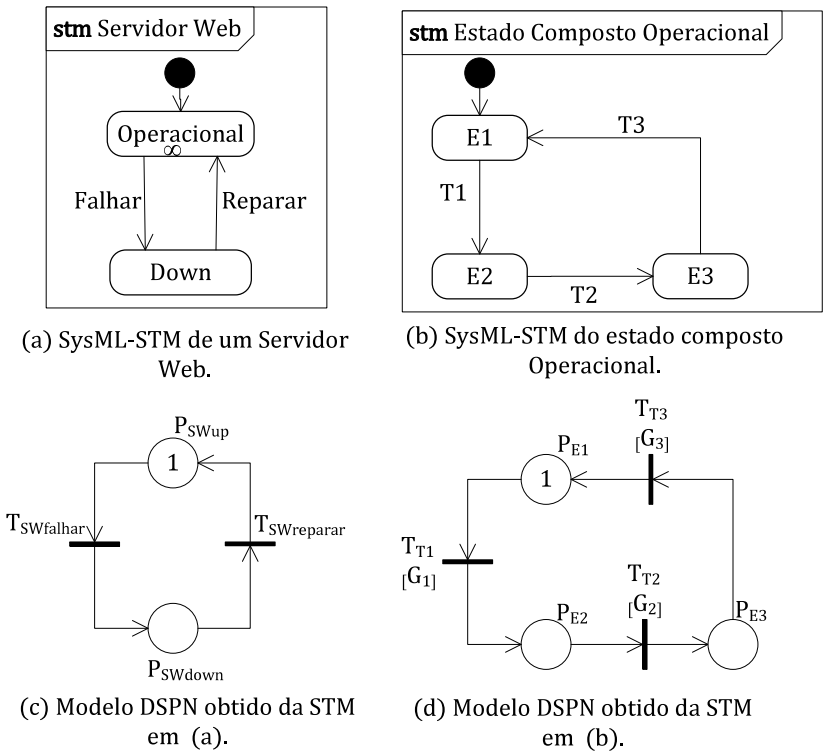
Os estado compostos são definidos como estados que contêm subestados (estados aninhados) [OMG03]. Essas estruturas compostas são usadas pelos projetistas para simplificar a descrição dos sistemas complexos, nos quais as transições internas comuns são agregadas em estados compostos. Esses estados são identificados através da imagem do símbolo infinito (∞) dentro deles. A Figura 4.6 (a) apresenta um exemplo de um SysML-STM contendo um estado composto, chamado de *Operational*. Esse estado composto *Operational* é detalhado na Figura 4.6 (b). O mapeamento desse exemplo é semelhante aos apresentados anteriormente, em que ambos os diagramas (SysML-STM do servidor *Web* e SysML-STM do estado composto) são mapeados em dois modelos DSPN distintos (ver Figuras 4.6 (c) e (d)). No entanto, as transições T_{T1}, T_{T2} e T_{T3} que representam os eventos de mudança de estado dos subestados ($E1$, $E2$ e $E3$) são desabilitadas enquanto o servidor não assumir o estado *Operacional*, indicado por um token no lugar P_{SWup}. Para capturar tal comportamento, as funções de guarda $G1$, $G2$, e $G3$ são atribuídas, respectivamente, às transições T_{T1}, T_{T2} e T_{T3} (ver Figura 4.6 (d)). As funções de guarda são listadas na Tabela 4.1. É importante ser ressaltado que as transições dos SysML-STMs são mapeadas em transições imediatas, visto que as mesmas não possuem restrições de tempo associadas a elas.

Tabela 4.1: Funções de Guarda para a Figura 4.6 (d).

Guarda	Função
G_1	$\sharp P_{SWup} = 1$
G_2	$\sharp P_{SWup} = 1$
G_3	$\sharp P_{SWup} = 1$

4.3.3 Mapeamento do SysML-AD

Um exemplo de um diagrama de atividades representando a manutenção de um servidor é mostrado na Figura 4.7 (a). A ação de evento de tempo (elemento descrito através de uma



(a) SysML-STM de um Servidor Web.

(b) SysML-STM do estado composto Operacional.

(c) Modelo DSPN obtido da STM em (a).

(d) Modelo DSPN obtido da STM em (b).

Figura 4.6: Mapeamento do Estado Composto do SysML-STM.

ampulheta) representa um período de espera de 24 horas antes que o status do servidor seja checado. O tempo de espera desse evento é especificado utilizando as anotações do *profile* MARTE. Após esse tempo, o status do servidor é verificado. Em seguida, a condição do nó de decisão é analisada, a fim de determinar qual fluxo de atividade será seguido. Se o servidor estiver funcionando [Sim], o mesmo será desligado de acordo com a atividade *Shutdown*, e o SysML-AD alcança o seu fim. Caso contrário [Não], nenhuma atividade é executada e o SysML-AD finaliza suas atividades. Como pode ser observado na Figura 4.7 (a), as atividades *Checar Status do Servidor* e *Shutdown* possuem anotações de disponibilidade especificadas através de MARTE. Para essas atividades foram atribuídos tempos exponenciais de 1 e 2 minutos, respectivamente. Note que esses tempos também são especificados através de MARTE. Semelhantemente ao SysML-STM, o estereótipo ≪controle≫ também foi usado para especificar atividades que afetem os estados dos sistemas. O servidor da Figura 4.5 (a) muda do estado *Operacional* para o estado *Desligado* pela execução da atividade *Shutdown* (ver Figure 4.7(a)).

O diagrama de atividades pode ser mapeado em um modelo DSPN convertendo cada tipo de nó em um conjunto de transições, lugares e funções de guarda. A Figura 4.7 apresenta o resultado do mapeamento de um exemplo utilizando as regras descritas na Figura 4.8. Os lugares P_{ini} e P_{in_clock} possuem 1 token cada, e a transição determinística T_{clock} dispara sempre a cada 'd' unidades de tempo depositando um token em P_{out_clock}. Quando a transição T_{wait} está habilitada em resposta a remoção do token em P_{in_clock}, a operação de manutenção representada pelo SysML-AD começa suas atividades checando

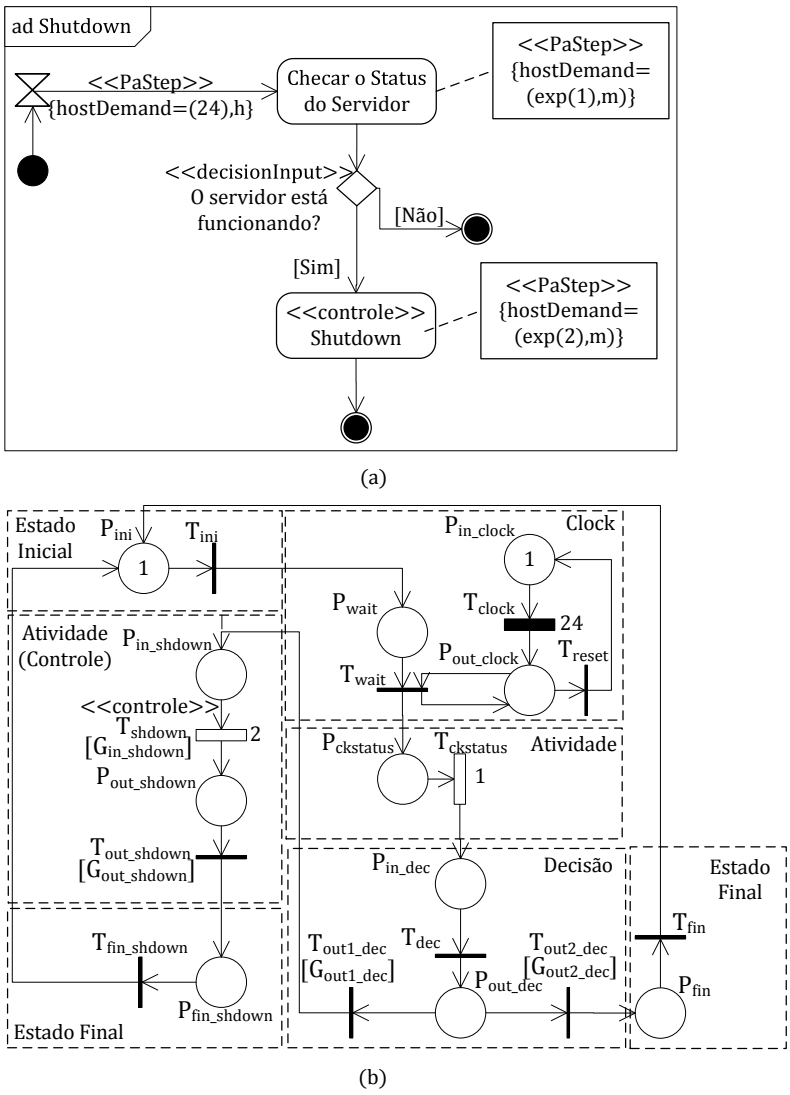


Figura 4.7: Mapeamento do SysML-AD.

o estado do servidor através da transição$T_{ckstatus}$. Após isso, o nó de decisão decide se o servidor está funcionando ou não. Note que existem dois fluxos de atividades representados pelas transições imediatas T_{out1_dec} e T_{out2_dec}, e cada uma dessas transições possuem uma função de guarda que corresponde à condição de decisão. Se o servidor estiver no estado operational (UP), então a transição T_{out1_dec} é disparada. Em seguida, o *Shutdown* é executado através da transição T_{shdown} e o SysML-AD alcança seu fim. Caso contrário, nenhuma atividade é executada e o SysML-AD finaliza suas atividades. As transições dos estados finais incluem arcos de saída para o estado inicial, já que consideramos que o *clock* é executada repetidamente. Note que a transição T_{shdown} é anotada com o estereótipo ≪controle≫. Essa anotação vem do diagrama de atividades e significa que a execução da ação *Shutdown* irá resultar na mudança de estado do sistema. A transição T_{clock} é atribuída com um tempo determinístico de 24 horas, e as transições T_{shdown} and

$T_{ckstatus}$ são atribuídas com os tempos exponenciais de 1 e 2 minutos, respectivamente. Todas essas atribuições são baseadas nas anotações do *profile* MARTE.

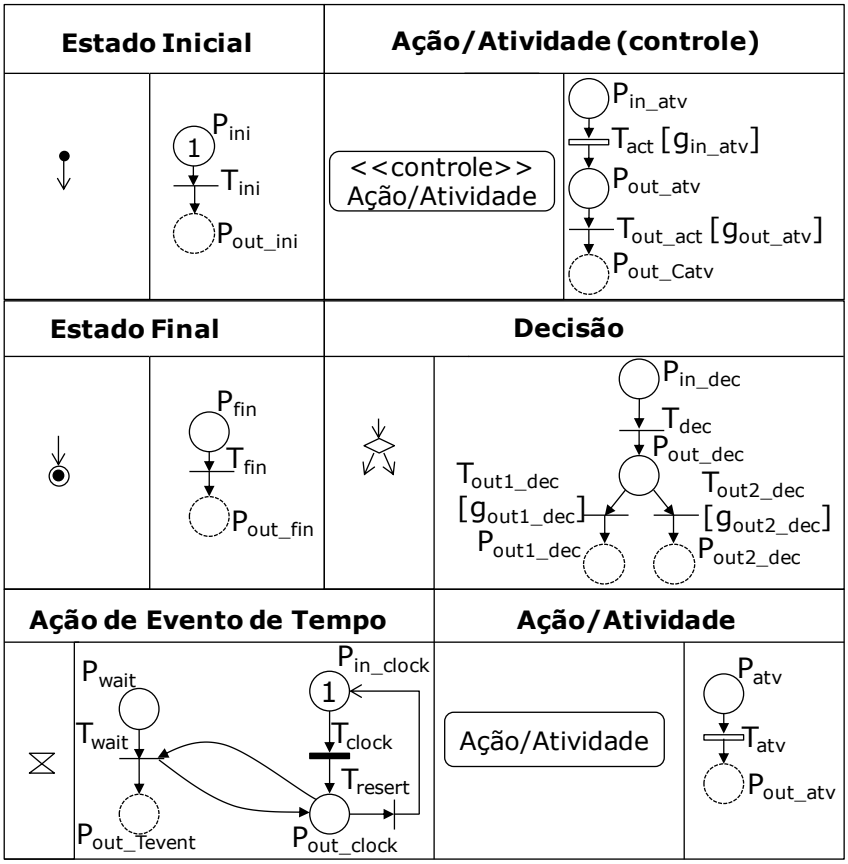


Figura 4.8: Regras de Mapeamento do SysML-AD.

Uma ação representa um passo único dentro de uma atividade, enquanto uma atividade pode conter ações de vários tipos. No entanto, uma ação de chamada de comportamento (*CallBehaviorAction*) consiste de uma ação que é definida em mais detalhes em outro diagrama de atividade. Essas ações de chamadas de comportamento podem ser identificadas pelo símbolo do garfo (⋔) no canto inferior direito da ação. A Figura 4.9 (a) apresenta um exemplo de um mecanismo de *failover*, na qual a atividade *Enviar Alerta* é detalhada através do diagrama da Figura 4.9 (b). O modelos DSPN de ambos os diagramas são obtidos através das regras de mapeamento apresentadas anteriormente (ver Figuras 4.9 (c) e (d)). A única diferença está na necessidade de sincronizar as atividades de ambos os diagramas através das funções de guardas G_{alerta}, G_{ini} e $G_{fin\_RestServer}$. Essas funções de guarda são detalhadas na Tabela 4.2.

4.3.4 Composição dos Modelos

A linguagem de modelagem SysML suporta várias opções de notações para representar relacionamentos que mapeiam um elemento do modelo em outro. Esse tipo de notação é

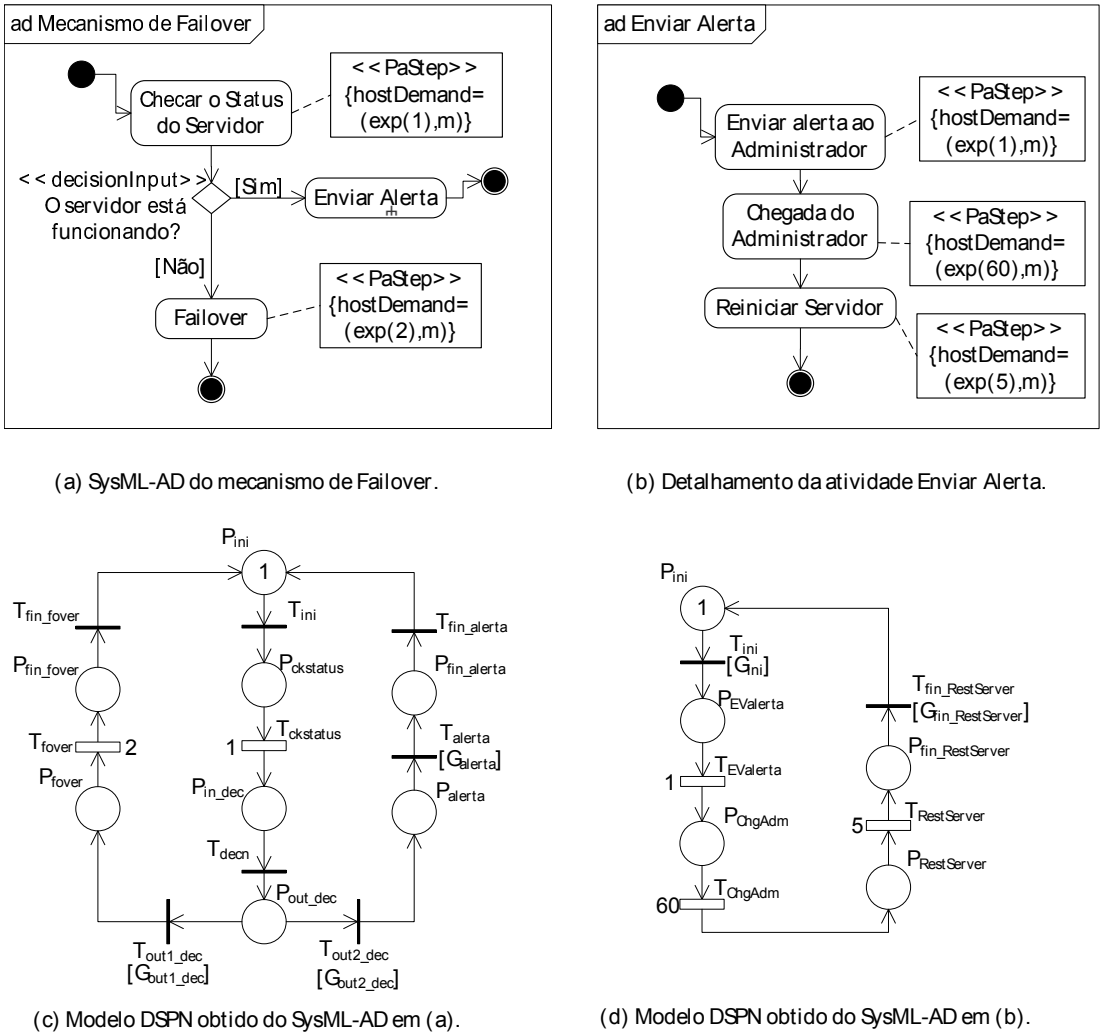


Figura 4.9: Mapeamento da Ação de Chamada de Comportamento do SysML-AD.

Tabela 4.2: Funções de Guarda para a Figura 4.9 (d).

Guarda	Função
G_{alerta}	$\sharp P_{fin\_RestServer} = 1$
G_{ini}	$\sharp P_{alerta} = 1$
$G_{fin\_RestServer}$	$\sharp P_{alerta} = 0$

chamada de alocação, sendo usada para prover flexibilidade para os diagramas da SysML. Esta tese adota três tipos de alocações: *allocatedFrom*, $\ll hosted \gg$ e $\ll standby \gg$. Essas alocações são usadas para guiar o processo de composição e sincronização dos componentes dos modelos obtidos a partir dos diagramas da SysML. A alocação *allocedFrom* define relacionamentos entre os blocos e os diagramas da SysML (SysML-STM e SysML-AD). A alocação entre um bloco e um diagrama de estados é usada para detalhar o bloco em

termos de estados, enquanto a alocação entre um bloco e um diagrama de atividades é usada para representar os mecanismos de tratamento de interrupções (ex.: *failover*, *restart* etc.) que podem afetar os estados de um elemento do sistema. Note que apenas as alocações entre os blocos e os diagramas de estados são usadas durante o processo de composição dos modelos, já que a alocação entre um bloco e um diagrama de atividades é usada durante a sincronização dos modelos.

A Figura 4.10 apresenta o processo de composição dos modelos DSPN através do uso da alocação *allocatedFrom*. Esse tipo de alocação é utilizada para detalhar os blocos em termos de estado. Na Figura 4.10 (a), o bloco *Servidor Web* é detalhado em termos de estados pelo o ≪*stm*≫ *Máquina de Estados Web*, conforme descrito no compartimento *allocatedFrom* do bloco. O processo de composição dos modelos consiste em dois passos básicos. Primeiramente, o bloco e o SysML-STM são mapeados de acordo com as regras de mapeamento apresentadas anteriormente (ver Figura 4.10 (b)). Após isso, o componente do modelo obtido a partir do SysML-IBD é substituído pelo componente do modelo gerado a partir do SysML-STM, visto que o último provê mais detalhes do elemento sendo modelado (ver Figura 4.10 (c)). O modelo resultante é chamado de *Rede do Sistema*.

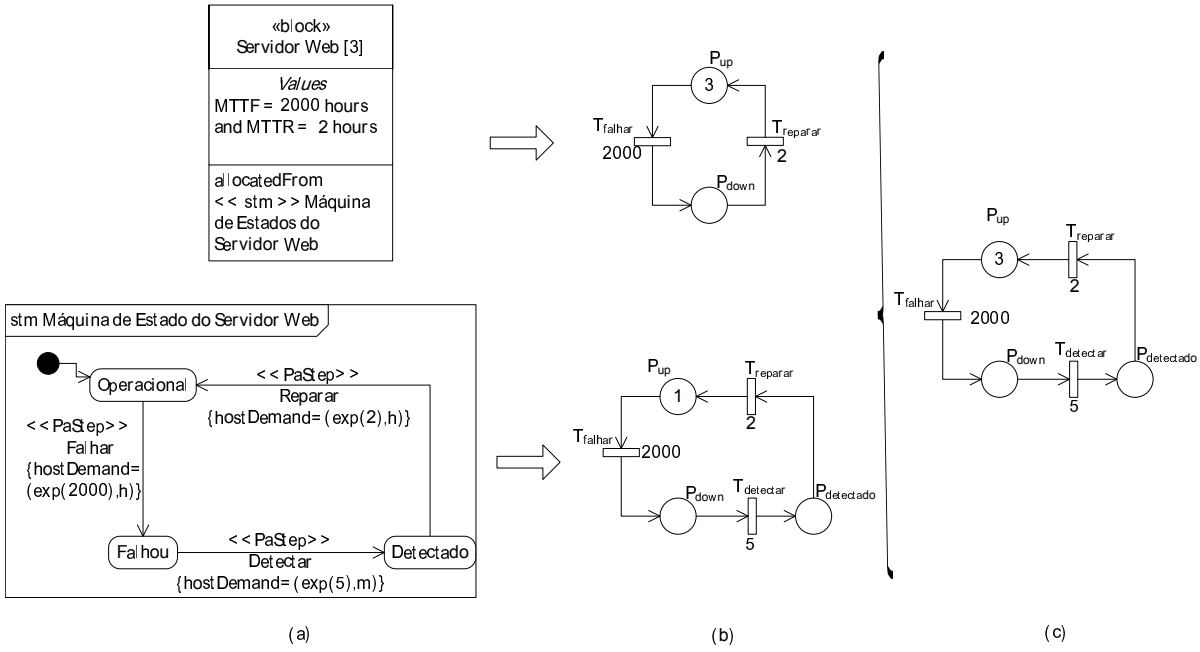


Figura 4.10: Processo de Composição dos Modelos DSPN Gerados a partir do SysML-IBD e SysML-STM.

Uma alocação ≪hosted≫ representa uma dependência de hospedagem entre os elementos do sistema. Por exemplo: uma dependência entre máquinas virtuais e *data center* (ver Figura 4.11 (a)). Se o *data center* (o elemento que hospeda) parar de funcionar, as máquinas virtuais (os elementos hospedados) também se tornarão indisponíveis ao mesmo tempo. Esse tipo de dependência pode ser representada através da inclusão de transições imediatas e funções de guarda no componente do modelo do elemento que está sendo

hospedado. A Figura 4.11 apresenta um exemplo desse processo de composição. Primeiramente, os blocos são mapeados de acordo com as regras de mapeamento apresentadas anteriormente (ver Figura 4.11 (b)). Em seguida, para o estado operational do componente do modelo das máquinas virtuais (lugar P_{MVup}), uma transição imediata T_{MVdown} e uma função de guarda G_{MVdown} são criadas para depositar tokens no lugar P_{MVdown}, caso o *data center* se torne indisponível (ver Figura 4.11 (c)). Uma vez que as transições do servidor *Web* nunca disparam enquanto o *data center* estiver indisponível, a transição $T_{MVreparar}$ é desabilitada através da função de guarda $G_{MVreparar}$.

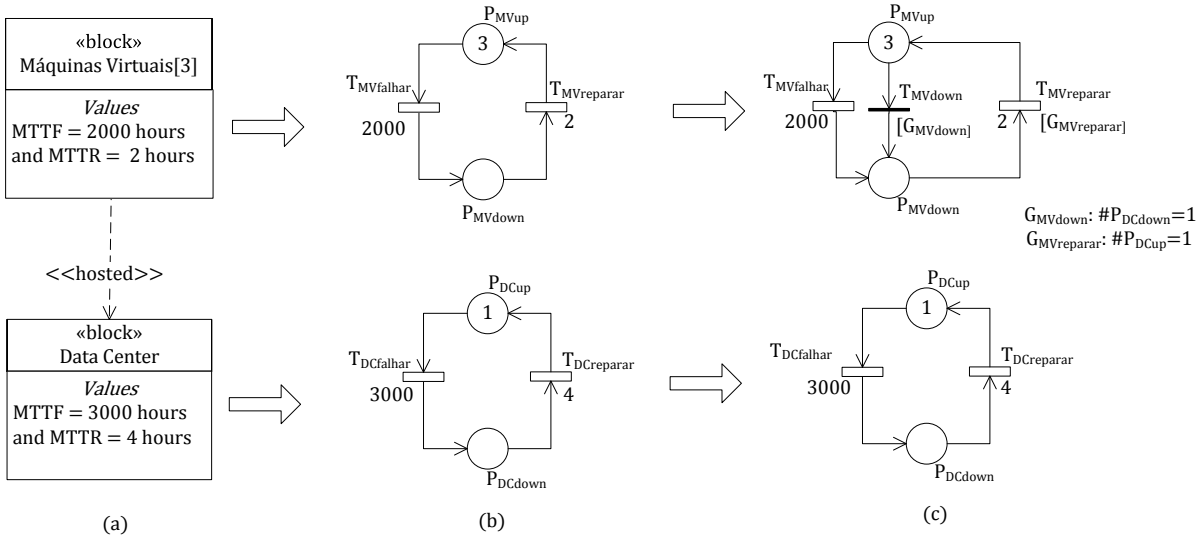


Figura 4.11: Processo de Composição dos Componentes do Modelo Através da Alocação ≪hosted≫.

Uma alocação ≪*standby*≫ define um relacionamento de *failover* entre um elemento *standby* (elemento redundante em estado de prontidão) e um elemento ativo. Quando o elemento ativo falhar, o elemento *standby* assume a operação do elemento ativo. Após o elemento ativo ser reparado, o elemento *standby* retorna para o estado de *standby*. Esse relacionamento de *failover* pode ser representado através da inclusão de funções de guarda no modelo do elemento *standby*. A Figura 4.12 apresenta um exemplo desse processo de composição. Antes de mais nada, dois tipos de blocos são considerados: o bloco tradicional ≪block≫ e o bloco estereotipado com ≪hot standby≫. O bloco ≪hot standby≫ foi definido para poder representar o elemento *standby*. O bloco *BD* é mapeado de acordo com as regras de mapeamento apresentadas anteriormente. Já o mapeamento do bloco *BD de Reserva* (estereotipado com ≪hot standby≫) gera um modelo DSPN distinto que considera o relacionamento de *failover*, conforme apresentado na Figura 4.12. Os tempos atribuídos às transições do modelo do bloco *BD de Reserva* são baseadas nas propriedades definidas no bloco (ver compartimento *values*). *Fover* e *Fback*, representam, respectivamente, o tempo de *failover* e *failback* do BD de reserva. MTTF e $MTTF_{hot}$ representam o tempo médio para a falha do BD de reservar no estado operacional e o tempo médio para a falha do BD de reserva no estado *standby*, respectivamente. Por fim, MTTR e hot, representam, respectivamente, o tempo médio de reparo e o tempo de *restart* do BD

de reserva. Para as transições que representam as operações de *failover* e *failback* do BD de reserva ($T_{DHfover}$ e $T_{DHfback}$), duas funções de guarda são definidas para especificar as condições de execução dessas operações. A transição $T_{DHfover}$ é habilitada pela função de guarda $G_{DHfover}$ quando um token é depositado em P_{BDdown}. Semelhantemente, a função de guarda $G_{DHfback}$ é habilitada quando um token é depositado em P_{BDup}.

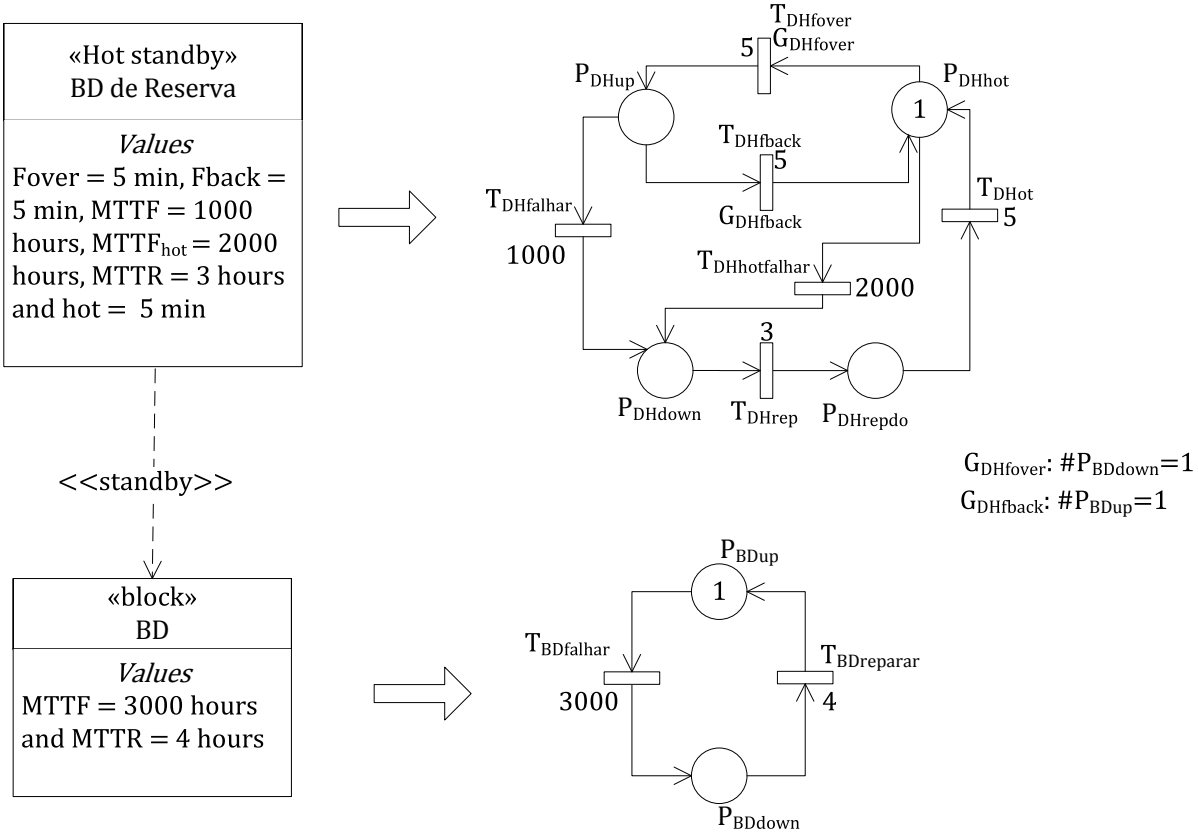


Figura 4.12: Processo de Composição dos Componentes do Modelo Através da Alocação ≪standby≫.

4.3.5 Sincronização dos Modelos

Existem dependências bidirecionais entre as *Redes do Sistema* e as *Redes de Atividades*, em que uma atividade na *Rede de Atividades* pode induzir à mudança de estados na *Rede do Sistema*. Esse tipo de atividade é identificada através do esterótipo ≪controle≫. Além disso, o fluxo das atividades modeladas através da *Rede de Atividades* pode mudar, dependendo da marcação da *Rede do Sistema*. Essa dependência pode ser incorporada na *Rede de Atividades* através da atribuição de funções de guarda as transições associadas.

4.3.5.1 Sincronização de uma Atividade na *Rede de Atividades* Se uma atividade na *Rede de Atividades* afeta a transição de estados na *Rede do Sistema*, então,

funções de guarda para habilitar a transição de estados são requeridas. Para cada transição estereotipada com ≪controle≫, os projetistas precisam encontrar a transição correspondente na *Rede do Sistema*. Vamos considerar o exemplo da Figura 4.13. Para sincronizar a *Rede de Atividades* e a *Rede do Sistema*, a transição T_y é expandida com uma transição imediata e um lugar com um arco inibidor, como apresentado na parte direita da Figura 4.13. A transição imediata T_{in_y} e o lugar P_{in_y} são inseridos antes da transição T_y. Eles representam as atividades de invocar e executar o estado, respectivamente. Para todas as transições relacionadas, o *framework* proposto gera 4 funções de guarda para habilitar as transições em uma ordem consistente. A primeira função de guarda G_{in_y} representa o disparo de uma atividade. A segunda função de guarda G_{in_x} garante o início da transição de estados. A terceira função de guarda G_{out_y} representa o fim de uma atividade. Por fim, a quarta função de guarda G_{out_x} assegura que a transição de estados está completa.

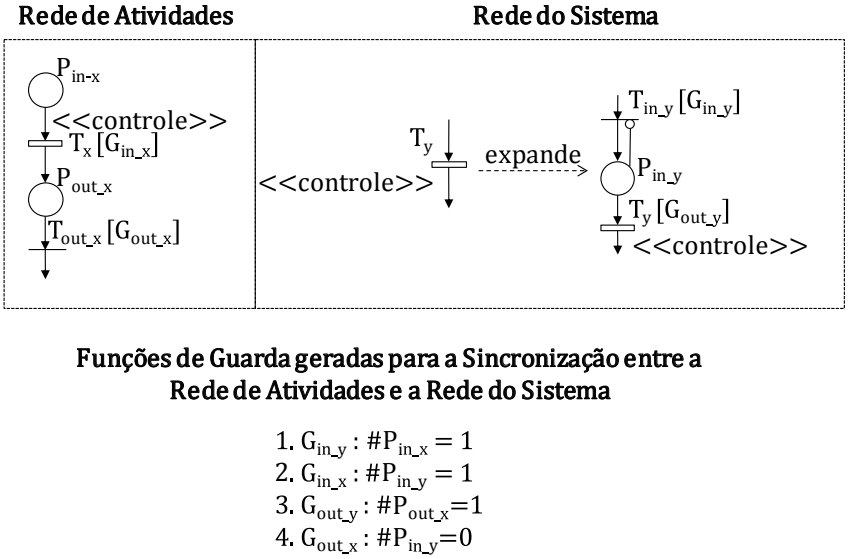


Figura 4.13: Sincronização de T_x na *Rede de Atividades* e T_y na *Rede do Sistema*.

A Figura 4.14 apresenta um exemplo de sincronização entre as transições $T_{desligar}$ da *Rede do Sistema* (ver Figura 4.5 (b)), e a transição T_{shdown} da Rede de atividades (ver Figura 4.7 (b)). Note que as transições são estereotipadas com ≪controle≫. A transição $T_{desligar}$ é expandida com uma transição imediata $T_{in\_desligar}$ e um lugar $P_{in\_desligar}$. Quatro funções de guarda são geradas automaticamente de acordo com as regras apresentadas anteriormente. É importante ser ressaltado que é possível ter mais de uma transição que precisa ser sincronizada para uma atividade. Assim, o processo de sincronização precisa ser estendido para endereçar tal necessidade.

4.3.5.2 Sincronização de uma Condição de Decisão Se o fluxo de controle de um nó de decisão do SysML-AD depende dos estados do sistema, então funções de guarda para a *Rede de Atividades* são definidas baseadas na marcação da *Rede do Sistema*. Os

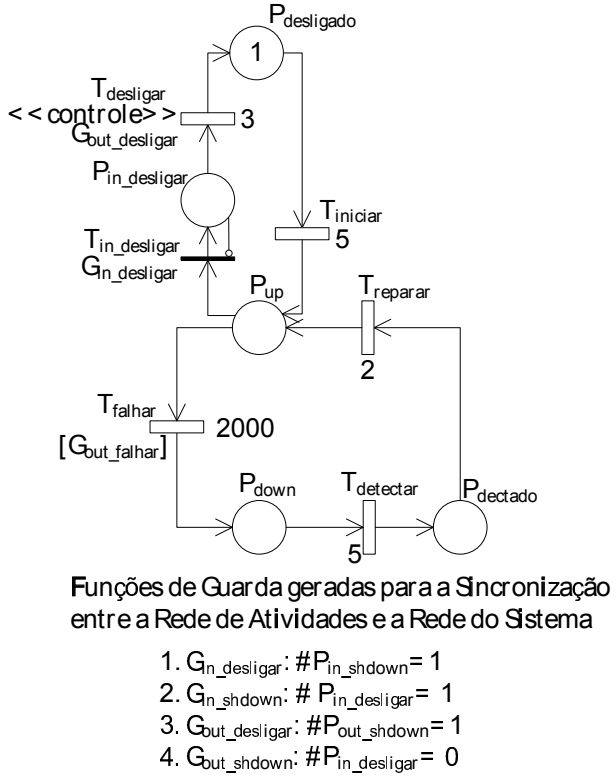


Figura 4.14: Um Exemplo de uma *Rede do Sistema* Expandida.

projetistas precisam definir as funções de guarda para os nós de decisão estereotipados com ≪decisionInput≫. Além disso, os SysML-STMs possuem anotações booleanas de "*falso*"e "*verdade*"indicando a disponibilidade do sistema nos estados. Tais anotações auxiliam os projetistas na criação das funções de guarda. Se controle de fluxo não depende do estado do sistema, a definição de funções de guarda não é necessária na avaliação de disponibilidade. Por exemplo, a Tabela 4.3 descreve as funções de guarda para as transições T_{out1_dec} e T_{out2_dec} (ver Figura 4.7(b)). Essas transições representam a decisão de realizar o *shutdown* ou não. No entanto, a realização de tal atividade depende da disponibilidade do servidor. Dessa forma, com base no número de tokens no lugar P_{up} (ver Figura 4.14), que indica a operacionalidade do servidor, são criadas as funções de guarda. É importante ser ressaltado que as anotações booleanas atribuídas aos estados do SysML-STM (ver Figura 4.5 (a)) ajudam os projetistas na identificação dos estados nos quais o servidor está disponível.

Tabela 4.3: Um Exemplo de Funções de Guarda para o ≪decisionInput≫.

Guarda	Função
G_{out1_dec}	$\sharp P_{up} = 1$
G_{out2_dec}	$\sharp P_{up} = 0$

4.3.6 Abordagem Hierárquica

Os sistemas distribuídos são grandes e complexos, consistindo de vários módulos e/ou componentes que interagem. Na análise de disponibilidade e performabilidade de tais sistemas complexos, os modelos sem espaço de estados são computacionalmente eficientes, mas possuem um poder de expressividade limitado. Já os modelos de espaço de estados são expressivos, porém, computacionalmente complexos. Adicionalmente, essa complexidade cresce exponencialmente com o tamanho do modelo. Essa explosão do espaço de estados precisa ser resolvida, a fim de modelar sistemas complexos usando os modelos de espaço de estados. Assim, a solução adotada nesta pesquisa, consiste no uso de modelos hierárquicos. Há duas razões fundamentais para a utilização de hierarquia: (i) reduz a complexidade dos modelos de modo que a construção e análise por parte dos projetistas seja viável e (ii) facilita a identificação de gargalos nos subsistemas ou componentes.

Portanto, nesta tese, a abordagem hierárquica é usada para os modelos DSPN obtidos pelo processo de mapeamento. Antes de mais nada, é importante ser ressaltado que essa abordagem só pode ser aplicada se existirem submodelos independentes. A Figura 4.15 apresenta um exemplo ilustrativo da utilização dessa abordagem. Primeiramente, os modelos DSPN são obtidos pelo processo de mapeamento apresentado anteriormente. Após isso, esses modelos são separados em subsistemas independentes e as disponibilidades são calculadas para cada um deles. Esses submodelos são identificados e analisados pelos projetistas e as métricas obtidas são usadas como entrada para cada bloco do modelo RBD. Por fim, o RBD é usado para calcular a disponibilidade de todo o sistema. A disponibilidade do sistema modelado através do RBD contendo n blocos consiste do produto da disponibilidade de cada bloco individual, A_i $(i = 1, 2, ..., n)$. É importante ser ressaltado que essa abordagem requer que os projetistas tenham um maior conhecimento em modelagem analítica, já que eles são responsáveis pela decomposição e agregação dos modelos DSPN.

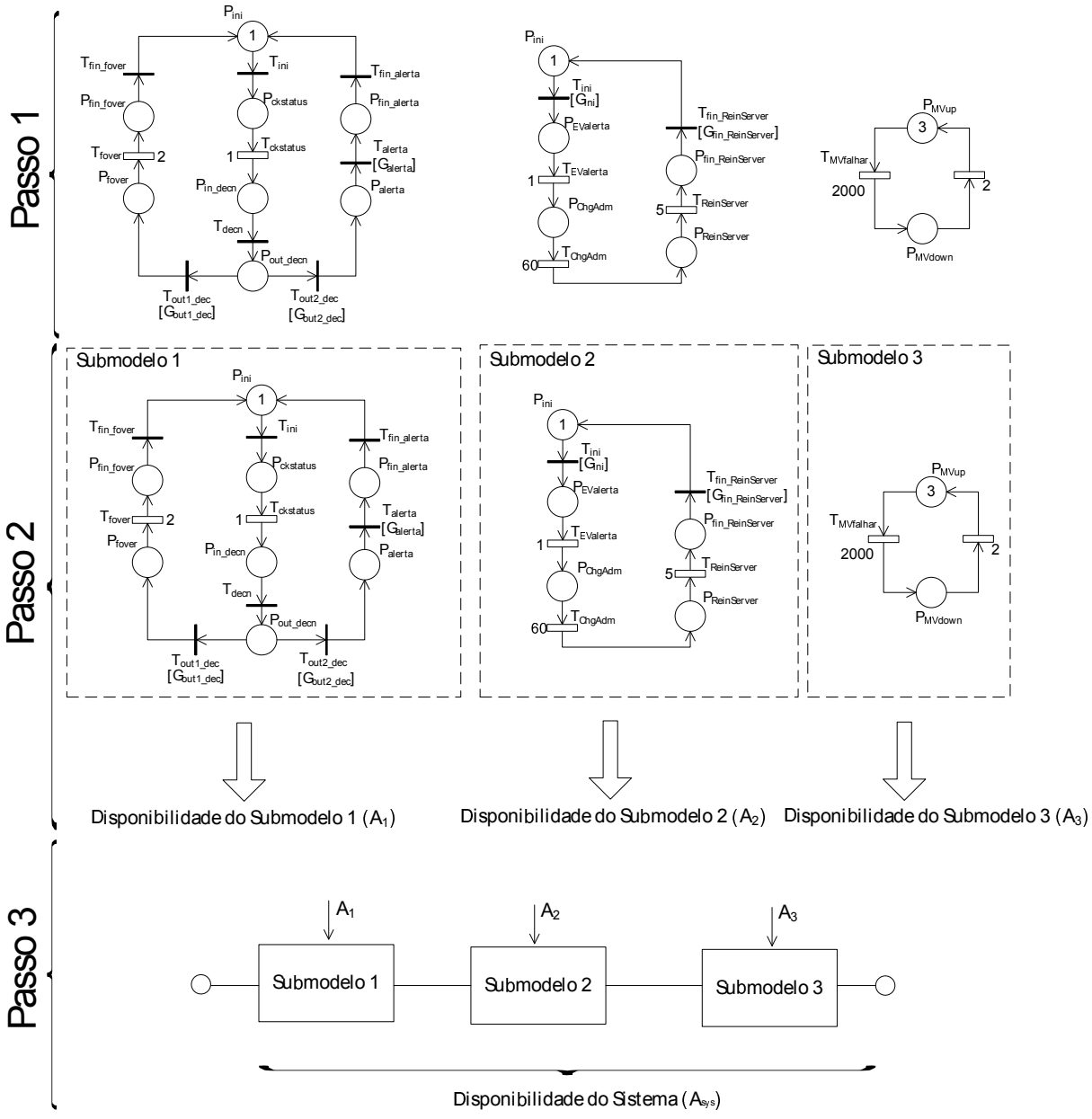


Figura 4.15: Um Exemplo Ilustrativo do Uso da Abordagem Hierárquica.

4.4 ANÁLISE DE PROPRIEDADES QUALITATIVAS

Os modelos DSPN resultantes do processo de mapeamento podem ter suas métricas (ex.: disponibilidade) obtidas tanto no regime transiente, quanto no regime estacionário. O cálculo destas métricas pode ser feito via simulação ou análise da cadeia de Markov embutida da DSPN [MBC+95, BGdT98]. Contudo, é necessário que esses modelos possuam algumas propriedades fundamentais, tais como conservação e repetividade. Uma RdP é considerada conservativa se o número total de tokens permanecer constante ou igual ao seu valor inicial, enquanto uma RdP é dita repetitiva se existe uma sequência de transições disparáveis, associada a uma dada marcação, na qual todas as transições são executadas um número de vezes infinito [Mur89]. Assim, caso os modelos DSPN obtidos pelo processo de mapeamento possuam essas duas propriedade, terão um gráfico de alcançabilidade finito e pelo menos um *home state* válido [MBC+95]. Consequentemente, eles poderão ser utilizados para o cálculo numérico das métricas no regime estacionário. A seguir, é demonstrado que todos os modelos criados a partir do *framework* proposto possuem essas propriedades. Como as guardas, arcos inibidores e prioridades não são levados em conta na análise de propriedades qualitativas, eles são desconsiderados nas análises a seguir.

Com o intuito de simplificar a análise das propriedades dos modelos DSPN, as técnicas de redução são aplicadas nos mesmos. Note que tais técnicas preservam as propriedades analisadas (conservação e repetitividade), sendo, portanto, passíveis de serem aplicadas [Mur89]. Estas técnicas consistem em reduzir o tamanho do modelo, utilizando um conjunto de regras de redução de maneira tal que a rede inicial e a rede reduzida tenham as mesmas propriedades (ver Seção 2.7.2 do Capítulo 2). A redução é feita através da remoção de lugares e/ou transições da RdP que não afetam as propriedades. Tal procedimento é muito comum na análise de modelos complexos.

4.4.1 Diagrama de Bloco Interno

O bloco é a unidade básica do SysML-IBD, sendo ele mapeado em dois lugares e duas transições representando o comportamento de falha e reparo do elemento do sistema o qual o bloco representa (ver Figura 4.3). Para facilitar a análise do modelo DSPN, o mesmo é reduzido através da aplicação de técnicas de redução, conforme discutido acima. A Figura 4.16 apresenta o processo de redução do modelo DSPN do bloco, onde a Figura 4.16 (b) consiste do modelo resultante da aplicação de técnicas de redução usando a fusão de transição em série. Observa-se que o modelo resultante é conservativo e repetitivo, já que o número de tokens se mantém constante e a transição pode ser disparada ilimitadamente. Note que independentemente da quantidade de blocos que o SysML-IBD contenha, os modelos DSPN resultantes sempre apresentarão as propriedades de conservação e repetividade, já que a geração das DSPN para cada bloco não requer a fusão de transições ou lugares.

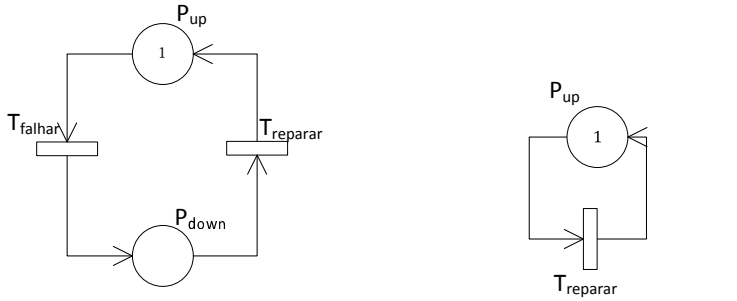


Figura 4.16: Redução do Modelo DSPN do SysML-IBD

4.4.2 Diagrama de Estados

O SysML-STM é utilizado para modelar o comportamento de falha/reparo dos elementos das infraestruturas distribuídas, já que o foco deste trabalho é analisar questões de disponibilidade. Dessa forma, os modelos projetados apresentam um comportamento cíclico, sempre retornando para o estado inicial, após uma sequência de eventos. O modelo DSPN do diagrama de estados da SysML é obtido através da conversão de cada elemento do diagrama de estados em respectivos elementos da DSPN. Isto é, cada estado é mapeado em um lugar e cada transição anotada é mapeada em uma transição temporizada com arcos de entrada e saída (ver Seção 4.3.2).

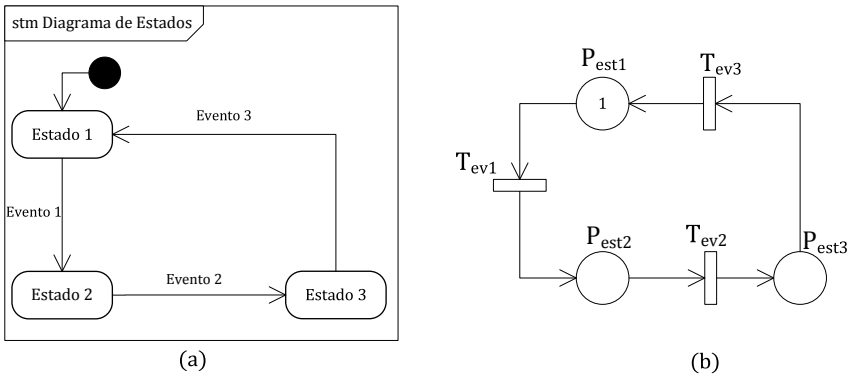


Figura 4.17: Mapeamento de um SysML-STM Composto por um Conjunto de Estados e Transições.

A Figura 4.17 apresenta o mapeamento de um SysML-STM composto por um conjunto de estados e transições. A Figura 4.18 (b) apresenta o modelo resultante da aplicação das técnicas de redução sobre o modelo DSPN da Figura 4.17 (b). Os modelos da Figura 4.18 (b) e da Figura 4.16 (b) são isomórficos. Por conseguinte, o modelo da Figura 4.18 (b) também apresenta as propriedades de conservação e repetitividade. Considerando que os diagramas de estado utilizados neste trabalho são cíclicos (ex.: comportamento de falha/reparo), então, independente da quantidade de estados e transições

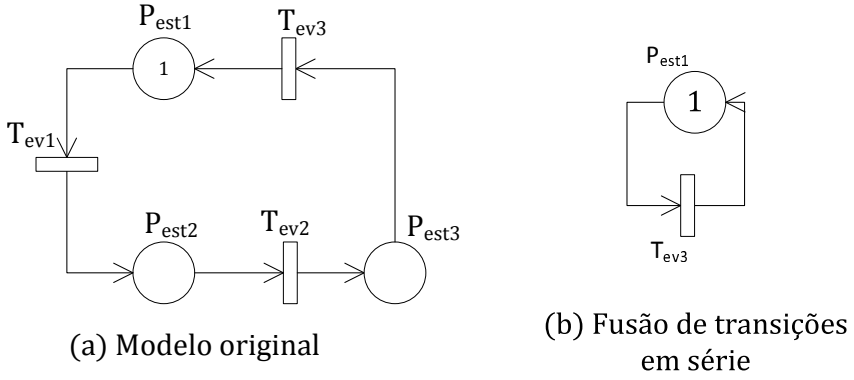


Figura 4.18: Redução do Modelo DSPN do SysML-STM Composto por um Conjunto de Estados e Transições.

que o diagrama possua, o modelo DSPN resultante do processo de mapeamento sempre poderá ser reduzido ao modelo DSPN da Figura 4.18 (b).

O mapeamento de um SysML-STM composto por estados que contêm mais de uma transição de entrada e saída é apresentado na Figura 4.19. A Figura 4.20 (b) apresenta o modelo DSPN resultante após a aplicação das devidas reduções sobre o modelo DSPN da Figura 4.19 (b). Observa-se que o modelo reduzido também apresenta as propriedades de conservação e repetitividade. Note que o disparo de qualquer transição não altera o número de tokens no modelo DSPN reduzido e todas as transições podem ser disparadas ilimitadamente.

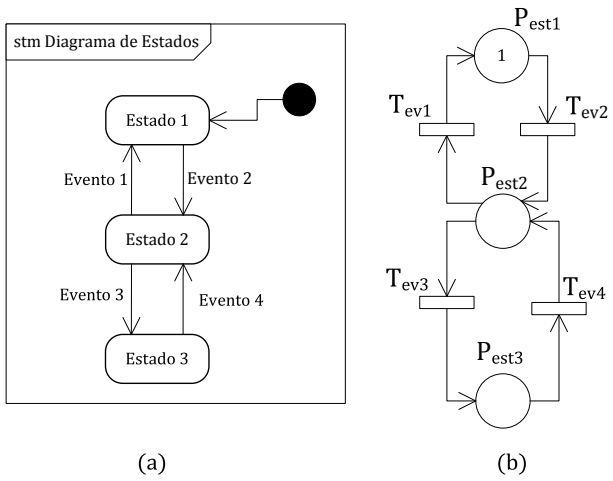


Figura 4.19: Mapeamento do SysML-STM Composto por Estados que Contém Mais de uma Transição de Entrada e Saída.

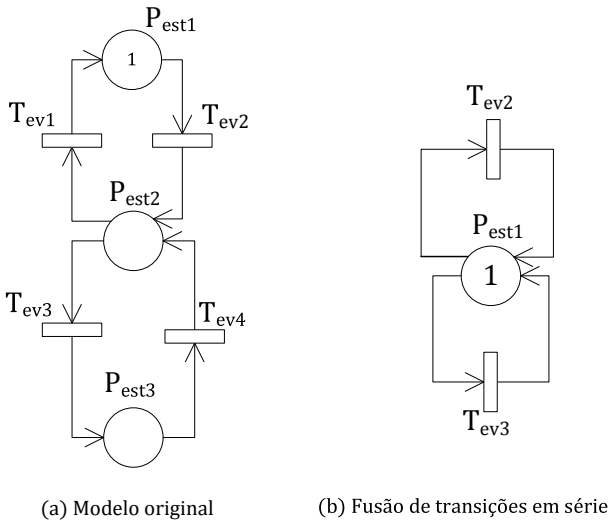


Figura 4.20: Redução do Modelo DSPN do SysML-STM Composto por Estados que Contém Mais de uma Transição de Entrada e Saída.

4.4.3 Diagrama de Atividades

Os modelos DSPN obtidos pelo mapeamento dos diagramas de atividades da SysML são gerados através de sucessivas fusões das interfaces dos fragmentos DSPN definidos para cada elemento do SysML-AD (ver Figura 4.8). Estas interfaces estão destacadas com linhas tracejadas nos modelos apresentados na Figura 4.8. Por exemplo, ao compor o fragmento DSPN de uma atividade com o fragmento DSPN de um estado inicial, o lugar P_{out_ini} do estado inicial e o lugar P_{atv} da atividade são fundidos. Os modelos resultantes dessas fusões também devem apresentar as propriedades de conservação e repetitividade para que possam ser analisados via análise estacionária. A seguir, mostraremos que os modelos obtidos pelo processo de mapeamento do diagrama de atividades possuem essas propriedades.

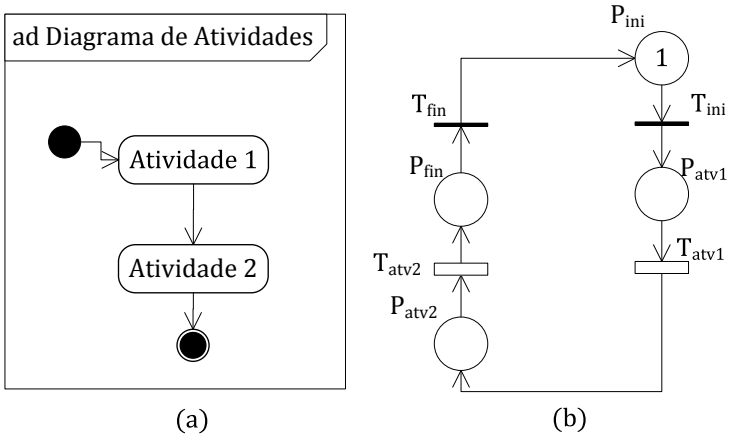


Figura 4.21: Mapeamento do SysML-AD Composto por Atividades Sequenciais.

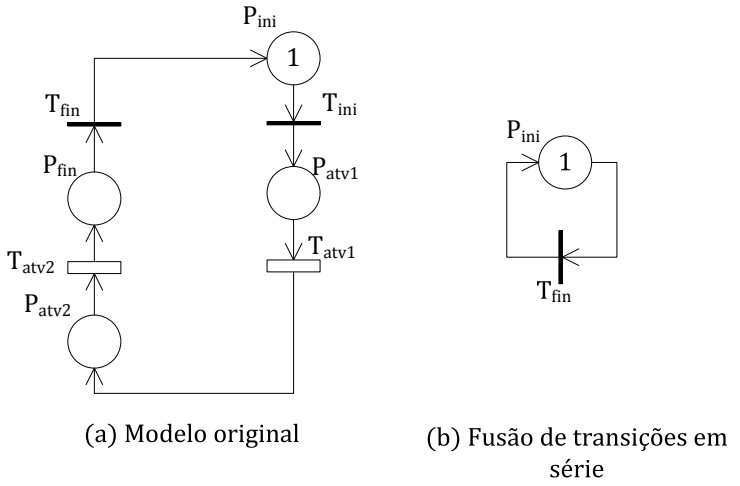


Figura 4.22: Redução do Modelo DSPN do SysML-AD Composto por Atividades Sequenciais.

A Figura 4.21 apresenta o mapeamento de um conjunto de atividades sequenciais do SysML-AD em um modelo DSPN, sendo esse modelo DSPN obtido de acordo com as regras de mapeamento apresentadas anteriormente. Antes de mais nada, note que a transição do estado final inclue um arco de saída para o estado inicial, já que consideramos que todos os SysML-ADs são executados repetidamente. A Figura 4.22 (b) apresenta o modelo resultante das sucessivas reduções do modelo DSPN da Figura 4.21 (b). Como os modelos DSPN apresentados na Figura 4.22 (b) e na Figura 4.16 (b) são isomórficos, conclui-se que o modelo da Figura 4.22 (b) também é conservativo e repetitivo. É importante ser ressaltado que o modelo DSPN de uma atividade estereotipada com ≪controle≫ pode ser reduzido ao modelo DSPN da atividade sem este esteriótipo, através da fusão de transições em série (ver Figura 4.8). Outro aspecto importante de ser ressaltado é que, independente da quantidade de atividades em sequência do diagrama de atividades, o modelo DSPN resultante do processo de mapeamento sempre poderá ser reduzido ao modelo DSPN da Figura 4.22 (b), que é conservativo e repetitivo.

O mapeamento de uma ação de evento de tempo seguido pela execução de duas atividades sequenciais é apresentado na Figura 4.23. Neste trabalho, consideramos que para cada diagrama de atividades pode existir apenas uma ação de evento de tempo. A Figura 4.24 apresenta o passo a passo de como as técnicas de redução foram aplicadas ao modelo DSPN da Figura 4.23 (b). O modelo DSPN da Figura 4.24 (d) representa modelo reduzido. Como os modelos DSPN da Figura 4.16 (b) e da Figura 4.24 (d) são isomórficos, o modelo DSPN da Figura 4.24 (d) também apresenta as propriedades de conservação e repetitividade. Note que, se consideramos um diagrama de atividades composto por uma ação de evento de tempo e um conjunto de atividades em sequência, o modelo DSPN resultante do processo de mapeamento sempre poderá ser reduzido ao modelo DSPN da Figura 4.24 (d).

A Figura 4.25 apresenta o mapeamento de um diagrama de atividades, considerando uma condição de guarda, a qual decide qual será a próxima atividade a ser executada.

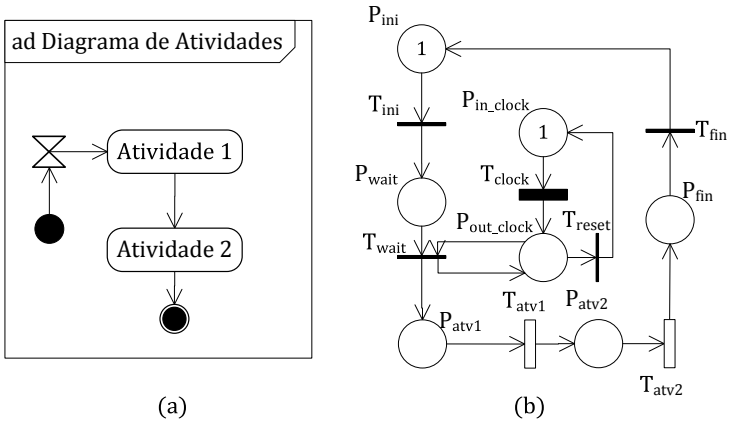


Figura 4.23: Mapeamento do SysML-AD Composto por uma Ação de Evento de Tempo e Atividades Sequencias.

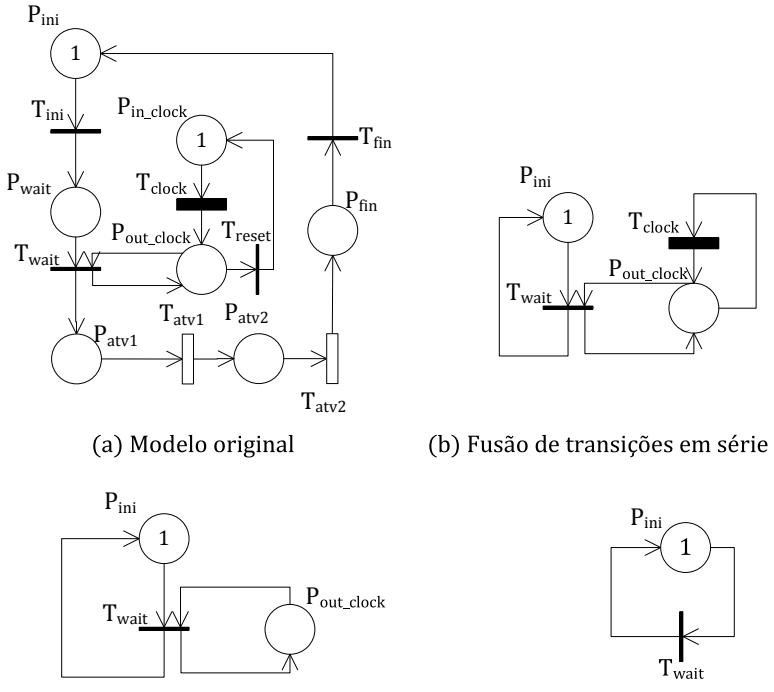


Figura 4.24: Redução do Modelo DSPN do SysML-AD Composto por uma Ação de Evento de Tempo e Atividades Sequências.

Nesse exemplo, são consideradas duas saídas condicionais. A Figura 4.26 apresenta o resultado da aplicação das técnicas de redução. Na Figura 4.26 (b), apresenta-se o modelo DSPN resultante das sucessivas reduções. Como este modelo e o modelo DSPN da Figura 4.20 (b) são isomórficos, este também é conservativo e repetitivo. Se considerarmos mais de duas condições de saída para a decisão, então o modelo DSPN resultante conterá um *self-loop* para cada saída condicional. Por inferência direta, este modelo também

possuirá as propriedades de conservação e repetitividade.

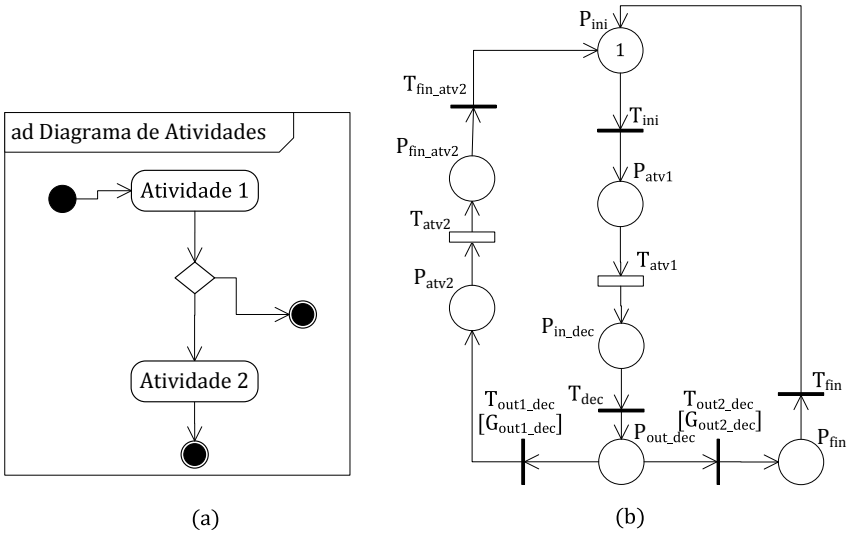


Figura 4.25: Mapeamento do SysML-AD Considerando Condições de Guarda.

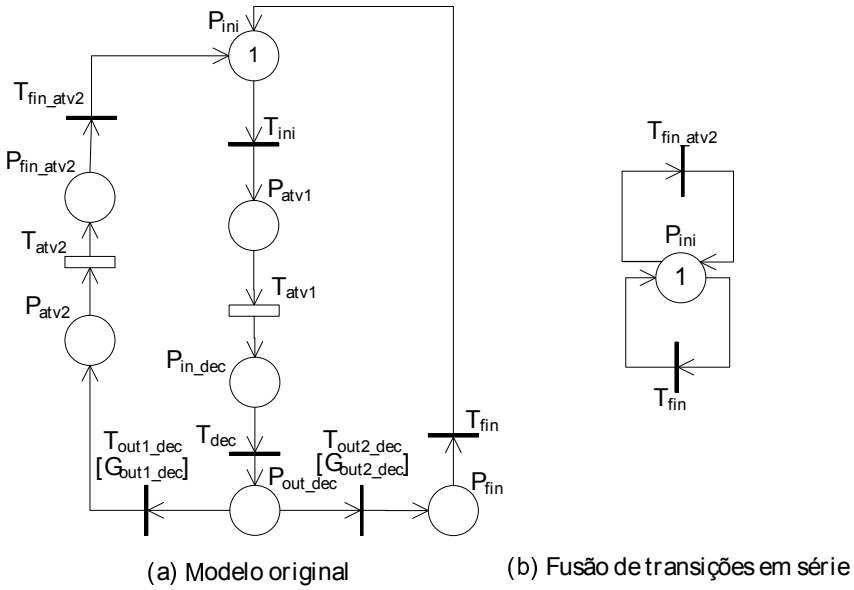


Figura 4.26: Redução do Modelo DSPN do SysML-AD Considerando Condições de Guarda.

4.4.4 Composição e Sincronização

As composições dos modelos DSPN são utilizadas para resolver problemas de dependências entre os elementos do sistema. Tais dependências são representadas através de notações chamadas de alocação. Mais especificamente, três tipos de alocações são usadas neste trabalho (*allocatedFrom*, $\ll hosted \gg$ e $\ll standby \gg$). A Figura 4.27 apresenta um exemplo do processo de composição utilizando a alocação *allocatedFrom*. Note que apenas um

modelo DSPN é obtido por esse processo de composição (ver Figura 4.27 (c)). Esse modelo resultante do processo de composição e o modelo apresentado na Figura 4.18 (a) são isomórficos. Consequentemente, ele também apresenta as propriedades de conservação e repetitividade. Visto que a composição usando a alocação *allocatedFrom* consiste na substituição do modelo DSPN do SysML-IBD pelo modelo DSPN do SysML-STM, então, conforme apresentado acima, independente do número de transições e estados que o SysML-STM possua, o modelo resultante sempre será conservativo e repetitivo.

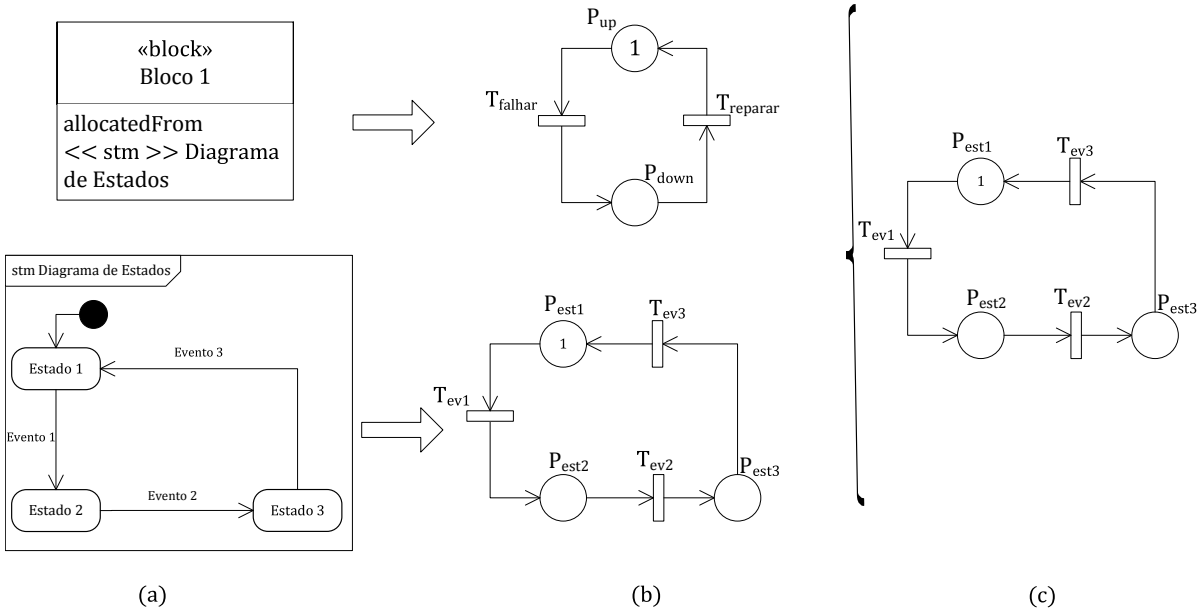


Figura 4.27: Processo de Composição dos Modelos DSPN Gerados a partir da Alocação *allocatedFrom*.

A alocação do tipo ≪hosted≫ é utilizada para representar as dependências de hospedagens entre os elementos do sistema. A Figura 4.11 apresenta um exemplo do uso desta alocação. Note que dois modelos DSPN são obtidos pelo processo de composição utilizando ≪hosted≫ (ver Figura 4.11 (c)). O modelo DSPN da parte inferior representa o bloco *Data Center*, enquanto o bloco da parte superior representa o bloco *Máquinas Virtuais*. O modelo DSPN do bloco *Data Center* e o modelo DSPN do SysML-IBD apresentado anteriormente (ver Figura 4.16) são isomórficos. Consequentemente, ele pode ser reduzido ao modelo apresentado na Figura 4.16 (b), que possui as propriedades de conservação e repetitividade. Já o modelo do bloco *Máquinas Virtuais* difere do modelo do SysML-IBD devido a uma transição extra entre os lugares P_{MVup} e P_{MVdown}. A Figura 4.28 apresenta o passo a passo de como as técnicas de redução foram aplicadas ao modelo DSPN do bloco *Máquinas Virtuais*, resultando no modelo DSPN da Figura 4.28 (d). Os modelos resultantes do processo de redução são isomórficos aos modelos apresentados na Figura 4.16 (b). Portanto, são conservativos e repetitivos.

A alocação ≪*standby*≫ define um relacionamento de *failover* entre um elemento *standby* e um elemento ativo. A Figura 4.12 apresenta um exemplo do uso dessa alocação, em que dois modelos DSPN são obtidos pelo processo de composição. O modelo DSPN

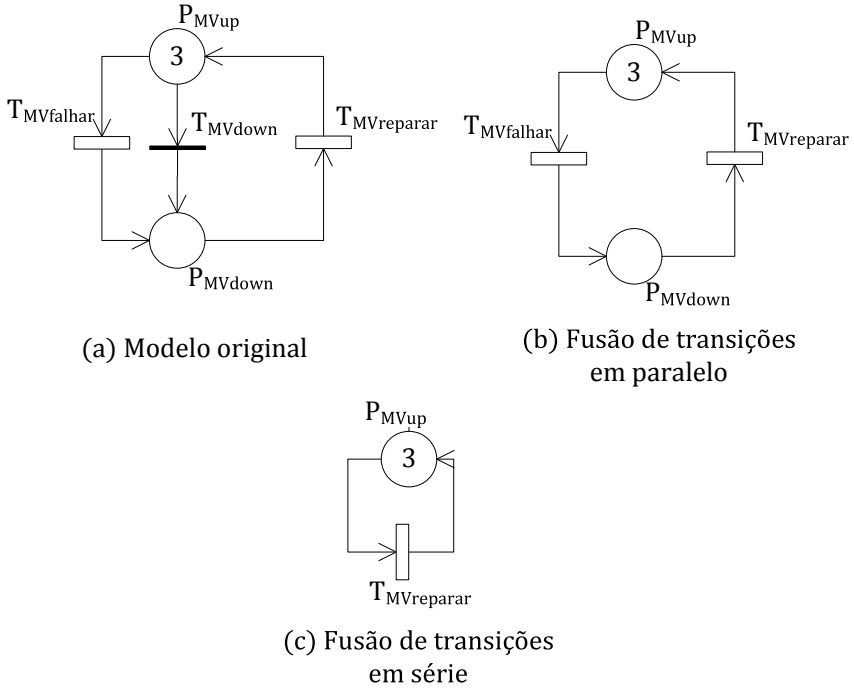


Figura 4.28: Redução do Modelo DSPN das *Máquinas Virtuais*.

da parte inferior representa o bloco *BD*, enquanto o componente *hot standby* da parte superior representa o *BD de Reserva*. Aplicando-se as técnicas de redução sobre o modelo DSPN do BD, obtém-se o modelo apresentado na Figura 4.16 (b). Por sua vez, a Figura 4.29 apresenta a aplicação das técnicas de redução sobre o modelo DSPN do *BD de Reserva*. O modelo resultante (ver Figura 4.29 (d)) é isomórfico ao modelo apresentado na Figura 4.20 (b). Portanto, observa-se que os modelos obtidos através da composição dos componentes do modelo usando a alocação ≪standby≫ são conservativos e repetitivos.

A sincronização dos modelos é realizada para resolver problemas de dependências entre as *Redes do Sistema* e as *Redes de Atividades* obtidas pelo processo de mapeamento. Neste trabalho, dois tipos de sincronizações são consideradas: a sincronização de uma atividade na *Rede de Atividades* e a sincronização de uma condição de decisão (ver Seção 4.3.5). Para o primeiro caso, as transições envolvidas na sincronização são expandidas com uma transição imediata e um lugar com arco inibidor. Os modelos expandidos, por sua vez, podem ser reduzidos de acordo com as regras apresentas anteriormente para os diagramas de atividades e estados. Já para o segundo tipo de sincronização, funções de guardas são definidas para as *Redes do Sistema* baseadas nas marcações das *Redes de Atividades*. Note que, nesse caso, não há nenhuma modificação na estrutura da rede. Ou seja, apenas funções de guardas são criadas para as condições de decisão. Partindo do pressuposto de que essas funções de guarda também podem sempre ser disparadas, então as *Redes de Atividades* podem ser reduzidas de acordo com as regras apresentadas anteriormente para o diagrama de atividades. Consequentemente, os modelos gerados a partir da sincronização preservam as propriedades de conservação e repetitividade.

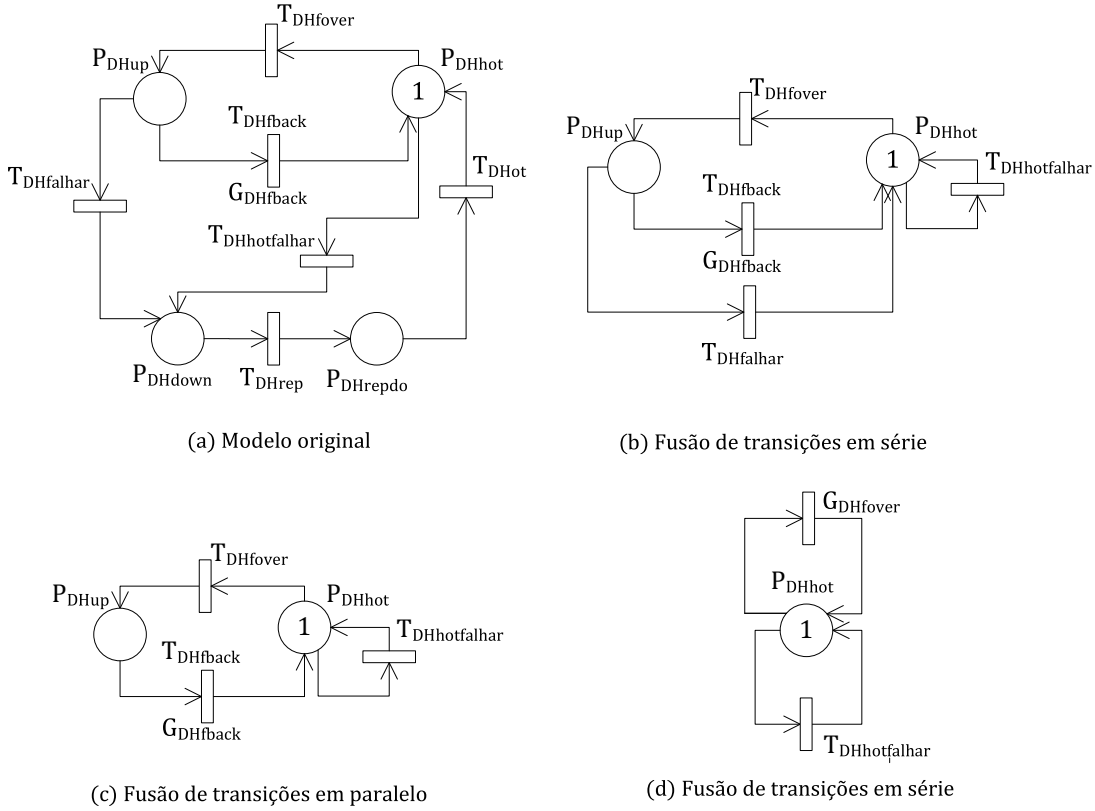


Figura 4.29: Redução do Modelo DSPN do *BD de Reserva*.

Conforme as exemplos apresentadas anteriormente, por inferência direta, o mapeamento, composição e sincronização dos diagramas da SysML sempre irão resultar em modelos DSPN que são conservativos e repetitivos. Assim, o cálculo de métricas pode ser feito tanto no regime transiente quanto no regime estacionário a partir da cadeia de Markov embutida. Vale salientar que os modelos apresentados neste trabalho podem ser utilizados como base para modelar as infraestruturas distribuídas e os mecanismos de tratamento de interrupção. Porém, caso sua estrutura seja alterada, ou as restrições apresentadas ao longo do texto não sejam respeitadas (ex.: valores das marcações iniciais ou funções de guarda), pode haver impacto sobre suas propriedades. Nestes casos, não há como garantir que o modelo resultante possuirá as propriedades observadas.

4.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o *framework* proposto para avaliação de dependabilidade das especificações dos sistemas distribuídos, levando em consideração os mecanismos de tratamento de interrupções. Em resumo, este *framework* baseia-se, principalmente, na composição sistemática dos modelos básicos gerados pelo processo de mapeamento dos diagramas da SysML, para, então obter-se um modelo que representa a especificação do sistema e, assim, realizar análises e verificações. Também foi apresentado que os modelos

gerados pelo processo de mapeamento são repetitivos e conservativos. Consequentemente, podem ser adotados para análises transientes e estacionárias.

CAPÍTULO 5

OPENMADS

Build your own dreams, or someone else will hire you to build theirs.
—FARRAH GRAY

A ferramenta OpenMADS (*An **OPEN**-source Tool for **M**odeling an **A**nalysis of **D**istributed **S**ystems*) [AAM+13] foi desenvolvida para suportar tanto o processo do mapeamento dos diagramas da SysML e anotações de MARTE, quanto as análises dos modelos DSPN obtidos por tal mapeamento. Essa ferramenta permite que os projetistas, os quais não têm (ou possuem pouca) experiência em modelagem formal, projetem e analisem os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos quantitativamente em uma plataforma de computação em nuvem. Os projetistas podem usar OpenMADS para: (i) modelar os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos usando os diagramas da SysML e as anotações de MARTE, (ii) gerar os modelos DSPNs através do processo de mapeamento automático, e (iii) estudar os mecanismos de mitigação e contingência das interrupções implantados nos sistemas distribuídos.

5.1 OPENMADS

Um *screenshot* da tela principal da ferramenta é mostrado na Figura 5.1. A ferramenta OpenMADS foi completamente escrita em Java, consequentemente, pode rodar em ambientes MAC, Linux ou Windows. A tela principal é composta por 3 partes principais: a barra de menu (parte superior), a área de modelagem (parte de baixo), e a barra de elementos (meio). Como pode ser observado na figura, também existem alguns menus disponibilizados pela ferramenta que auxiliam na construção dos modelos. No menu *File*, estão disponíveis algumas funcionalidades que, entre outras, permitem gravar e abrir os modelos da SysML. No menu *Diagram*, é possível escolher qual diagrama será mostrado pela ferramenta (diagrama de estados, diagrama de atividades, diagrama interno de blocos e redes de Petri). No menu *Edit*, um conjunto de funcionalidades para manipular os diagramas estão disponíveis, tais como operação de desfazer, selecionar todos os componentes do diagrama etc. No menu *View*, é possível alterar o plano de fundo da ferramenta. No menu *Arrange*, está disponível um conjunto de funcionalidades que permitem organizar os elementos dos diagramas. Por último, no menu *Help*, informações básicas acerca da ferramenta estão disponíveis. As funcionalidades implementadas na ferramenta são brevemente resumidas a seguir, conforme a Figura 5.2.

- **Editor da SysML e MARTE.** Os projetistas podem modelar as configurações

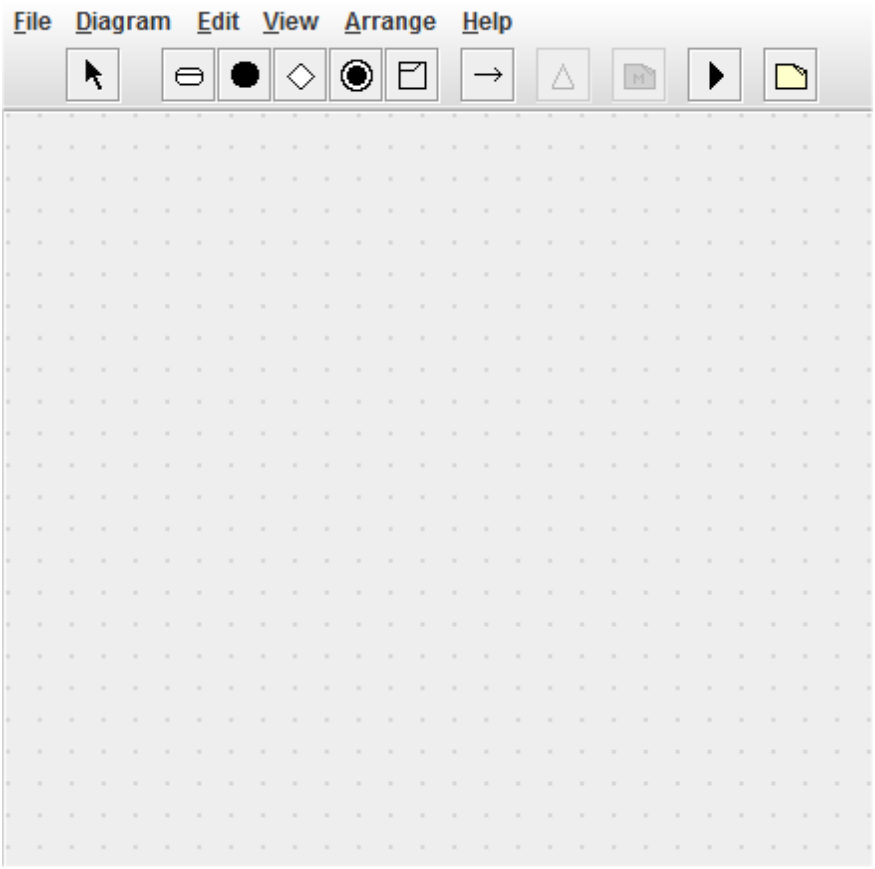


Figura 5.1: *Screenshot* da Tela Principal de OpenMADS.

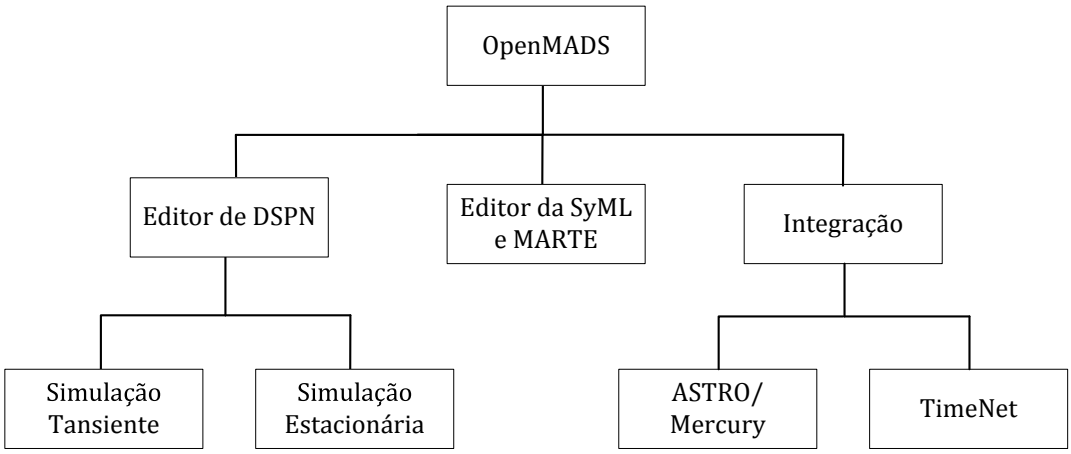


Figura 5.2: Funcionalidades da Ferramenta OpenMADS.

dos sistemas distribuídos, bem como os mecanismos de tratamento de interrupções usando OpenMADS. A ferramenta suporta três tipos diagramas da SysML: diagrama de atividades, diagrama estados e diagrama de bloco interno. OpenMADS também fornece suporte para as anotações do *profile* MARTE, as quais podem ser

atribuídas aos elementos dos diagramas. No entanto, apenas um subconjunto dessas anotações são providas por OpenMADS. Mais especificamente, a ferramenta implementa o estereótipo *PAStep* e o valor marcado *hostDemand*.

- **Editor de DSPN e Simulador Estocástico.** Esta área provê os modelos DSPN gerados automaticamente pelo processo de mapeamento dos diagramas anotados da SysML, ou criados do zero pelos os usuários. OpenMADS também permite avaliar a dependabilidade dos modelos usando técnicas de simulação transiente e estacionária. As métricas dependentes de tempo são obtidas através da simulação transiente, enquanto as métricas de estado estacionário são resultado da simulação estacionária.

- **Integração com ASTRO/Mercury e TimeNet.** A fim de permitir vários tipos de análises e simulações dos modelos DSPN, OpenMADS é integrado às ferramentas ASTRO/Mercury e TimeNet. Em outras palavras, OpenMADS permite aos projetistas gerarem arquivos de entrada para essas ferramentas. ASTRO/Mercury e TimeNet são ferramentas de modelagem analítica que proveem uma interface amigável de fácil uso para modelos híbridos, os quais incluem: redes de Petri, RBDs, e modelos de alto nível de *data center*.

A Figura 5.3 apresenta a arquitetura de OpenMADS. A partir dos diagramas da SysML, anotados de acordo com MARTE, OpenMADS cria um arquivo XML (*Extensible Markup Language*) parametrizado que irá servir de entrada para o *Módulo de Mapeamento*. O *Módulo de Mapeamento* implementa o processo de mapeamento dos modelos XML parametrizados para o formalismo DSPN. Durante o processo de mapeamento, cada elemento do diagrama da SysML é mapeado em partes do modelo de disponibilidade e essas partes são combinadas e sincronizadas para construir o modelo de disponibilidade completo. O arquivo XML dos modelos DSPN é usado para realizar análises de disponibilidade através das técnicas de simulação transiente e estacionária. Os resultados de tal análise são apresentados na GUI da ferramenta. Para os modelos DSPN, OpenMADS implementa o *Módulo de Integração*, que gera arquivos de entrada para as ferramentas ASTRO/Mercury e TimeNET. Essas ferramentas também auxiliam OpenMADS a executar análise hierárquica dos modelos DSPN, uma vez que OpenMADS não suporta os modelos sem espaço de estados. No entanto, é importante ressaltar que o processo de decomposição e agregação da abordagem hierárquica precisa ser realizado pelo projetista.

5.1.1 Editor de SysML e MARTE

Um exemplo da interface do editor do SysML-STM e das anotações de MARTE é apresentado na Figura 5.4. Esse editor pode ser usado para as seguintes atividade: (i) modelar os sistemas e os mecanismos de tratamento de interrupções usando os diagramas da SysML e as anotações de MARTE, (ii) exportar os *frames* dos diagramas da SysML e (iii) mapear os diagramas da SysML em modelos DSPN. Os diagramas da SysML são compostos por um conjunto de elementos básicos, tais como estado simples, transição, decisão, *frame*,

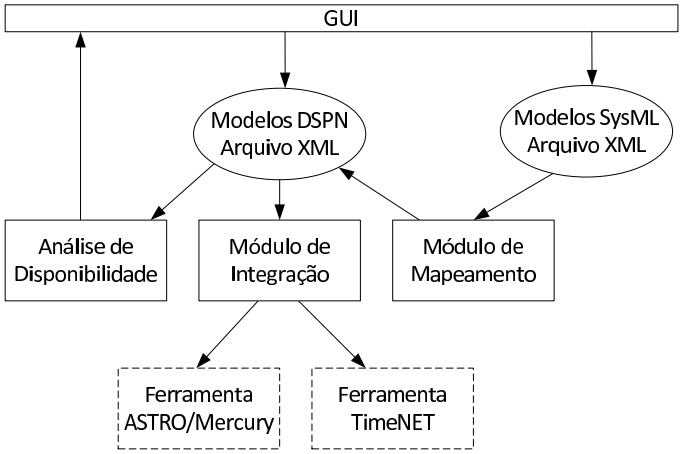


Figura 5.3: Arquitetura de *Software*.

atividade, decisão, estado inicial, estado final, entre outros. Esses elementos foram implementado com base na ferramenta ArgoUML [Tea13b]. OpenMADS também fornece suporte para as anotações do *profile* MARTE, as quais podem ser atribuídas aos elementos dos diagramas. Mais especificamente, OpenMADS implementa o estereótipo *PAStep* e os valores marcados *prob* e *hostDemand*. A funcionalidade de exportar *frames* permite que os projetistas possam exportar os *frames* dos SysML-STMs e SysML-ADs para a área de modelagem do SysML-IBD, a fim de associar os diagramas aos blocos através do uso de alocações. Por fim, a última funcionalidade desse editor compreende o botão de mapeamento. Ao clicar no mesmo, OpenMADS realiza o mapeamento dos diagramas da SysML em uma rede de Petri determinística e estocástica automaticamente, de acordo com as regras de mapeamento apresentadas anteriormente. As Figuras 5.5 e 5.6 apresentam, respectivamente, as interfaces dos diagramas de atividades e bloco interno. Note que esses diagramas possuem as mesmas funcionalidades apresentadas anteriormente para o diagrama de estados (ex.: elementos, MARTE, mapeamento etc). Todavia, é importante ser ressaltado que cada diagrama possui seus próprios elementos de modelagem.

5.1.2 Editor de DSPN e Simulador Estocástico

A Figura 5.7 apresenta um *screenshot* do editor das redes de Petri determinísticas e estocásticas (DSPN). Esse editor poder ser usado para três atividades básicas: (i) apresentar o modelo DSPN gerado pelo processo de mapeamento ou permitir que os usuários criem o modelo do zero, (ii) avaliar os modelos usando técnicas de simulação através da ferramenta OpenMADS ou (iii) exportar os modelos DSPN para as ferramentas ASTRO/Mercury ou TimeNet. Os modelos DSPN são compostos por lugares, transições, arcos e medidas de disponibilidade. Esses elementos são descritos em detalhes abaixo. A parte de avaliação compreende a execução dos modelos DSPN, que pode ser simulação transiente ou simulação estacionária. A funcionalidade de exportar gera arquivos de entrada para as ferramentas ASTRO/Mercury e TimeNet (ver Seção 5.1.3).

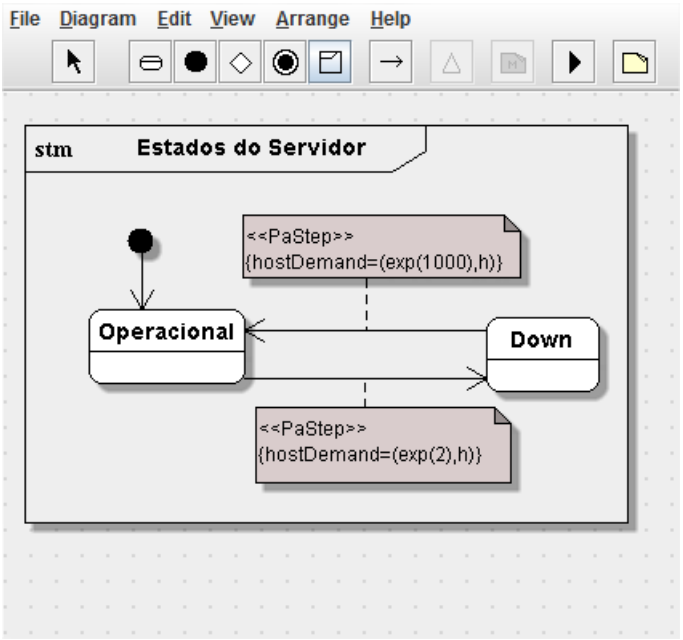


Figura 5.4: *Screenshot* do SysML-STM.

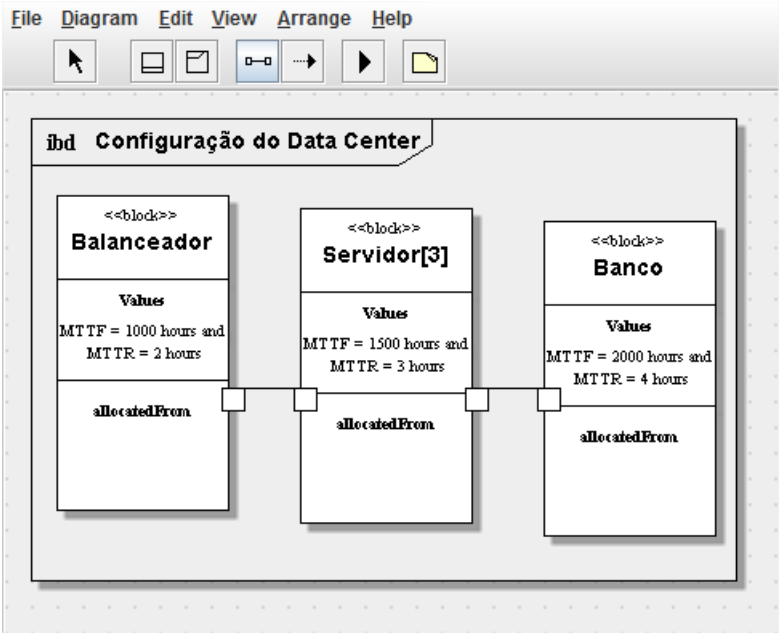


Figura 5.5: *Screenshot* do SysML-IBD.

Os componentes da área de desenho dos modelos DSPN são apresentados a seguir [Sil11].

1. **Componente lugar.** Os componentes que representam os lugares são representados por círculo, possuem duas informações principais:

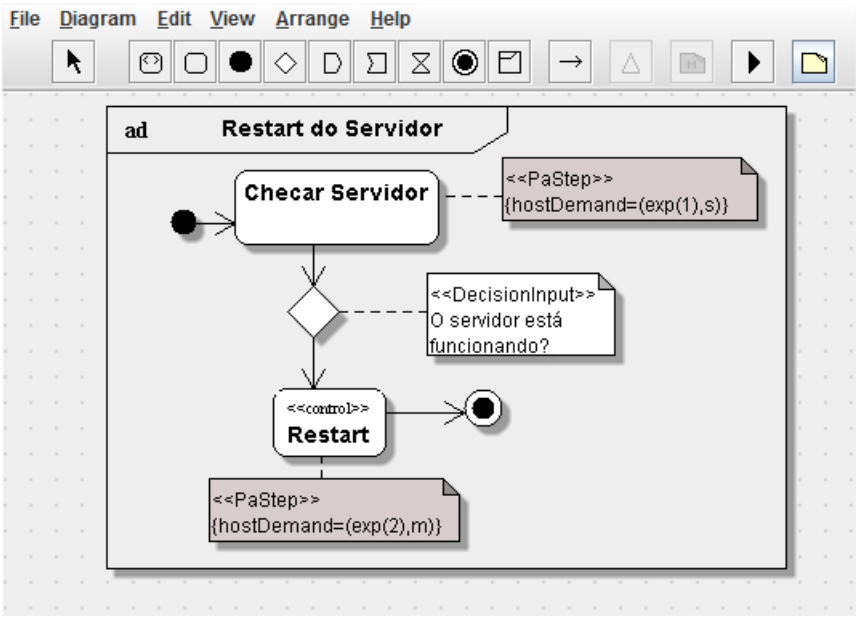


Figura 5.6: *Screenshot* do SysML-AD.

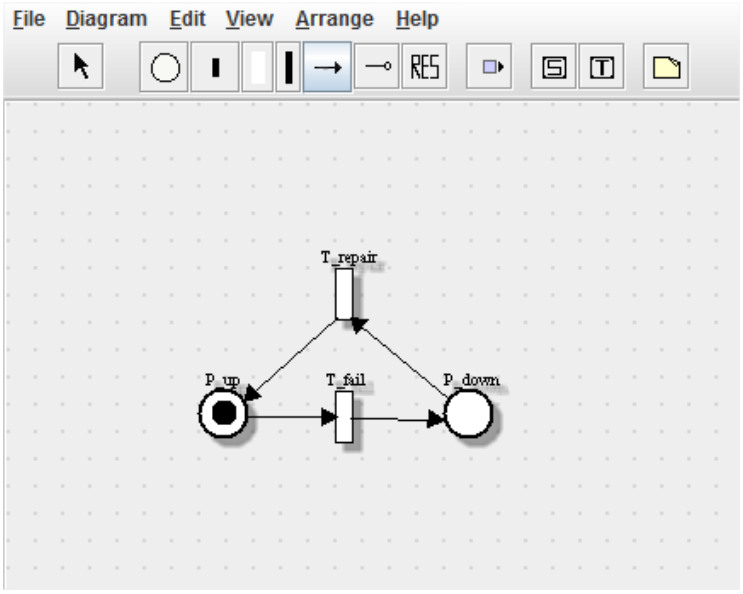


Figura 5.7: *Screenshot* do Editor de DSPN.

- **Nome.** É mostrado como um identificador junto ao componente. O nome é importante para definições de métricas e expressões de guarda.
- **Lugar.** Pode ser um número natural ou um rótulo pré-definido pelo usuário.

2. **Componente transição imediata.** Os componentes que representam transições imediatas são representados por retângulos estreito preenchidos, possuem três informações principais:

- **Prioridade.** É um número natural que define a precedência de disparo para as transições imediatas. O valor padrão é "um"e a prioridade aumenta com o aumento do valor.
- **Peso.** No caso de duas ou mais transições, de mesma prioridade, habilitadas simultaneamente, o peso é a informação utilizada para o cálculo da probabilidade de disparo das mesmas.
- **Função de habilitação.** A função de habilitação, também chamada de guarda, é uma expressão dependente de marcação que deve ser avaliada como verdadeira para que a transição possa disparar.

3. **Componente transição temporizada.** A definição de DSPN [Ger00] admite distribuições exponenciais e tempos determinísticos associados as transições temporizadas. Os componentes que representam as transições exponenciais e às transições determinísticas são representadas por retângulos não preenchidos e retângulos preenchidos, respectivamente. Além das informações relativas a cada distribuição de probabilidade associada, o tipo do serviço é um parâmetro utilizado nas transições temporizadas. Os tipos de serviço utilizados nas transições temporizadas são servidor infinito ou servidor simples. O servidor infinito considera a execução simultânea de ações, conforme o grau de habilitação [Mur89] da transição correspondente. O servidor simples considera execução sequencial de ações, ou seja, os tempos de disparo de cada transição são determinados sequencialmente.

4. **Métrica.** A métrica define o que deve ser computado durante a simulação. É possível calcular, por exemplo, a probabilidade de termos uma determinada quantidade de tokens num determinado lugar. Outros tipos de métricas também podem ser definidas, como: o número médio de tokens num determinado lugar ou taxa de disparo de uma determinada transição, por exemplo. As métricas possuem os seguintes atributos:

   - **Nome da métrica.** Identificador único da métrica.
   - **Expressão.** Uma gramática foi adotada [ZKHH06] para suportar a criação de métricas e expressões de guarda na ferramenta OpenMADS. Uma métrica é uma expressão que possui números, parâmetros de marcação e de tempo, operações algébricas e as seguintes medidas básicas:
     - $P\{< condicao\_logica >\}$. Corresponde à probabilidade de $< condicao\_logica >$ ser verdadeira.
     - $E\{< funcao\_marcacao >\}$. Valor esperado da função dependente de marcação $< funcao\_marcacao >$.

   Funções dependentes de marcação são definições do tipo $\#P_n$, e são relacionadas ao número de tokens no lugar P_n. Condições lógicas, geralmente, contêm comparações entre funções dependentes de marcação e números. Exemplos de métricas são $E\{\#P5\}$ e $P\{\#P2 > 0\}$.

OpenMADS adotou o pacote de simulação da ferramenta ASTRO/Mercury [Sil11] para avaliar os modelos DSPN gerados pelo processo mapeamento. A ferramenta disponibiliza duas formas de simulação: simulação transiente e simulação estacionária. A simulação estacionária permite a avaliação de desempenho/dependabiliadde do sistema depois que os efeitos transitórios iniciais se passarem e um estado estável seja alcançado, enquanto a a simulação transiente permite a análise do comportamento de um determinado sistema a partir do instante inicial até um determinado instante de tempo [Sil11]. O processo de simulação para ambas as formas é praticamente o mesmo, diferenciando apenas no critério de parada. O ambiente de simulação do OpenMADS (ver Figure 5.8) permite que os usuários ajustem os seguintes parâmetros:

- **Nível de confiança.** É a probabilidade de que o intervalo de confiança contenha o verdadeiro valor da métrica buscada .
- **Erro máximo relativo.** Indica o tamanho relativo do intervalo de confiança em termos percentuais, ou seja, é a razão entre tamanho do intervalo de confiança e o valor médio da amostra. Seja a a diferença do maior valor e menor valor do intervalo de confiança, e $\overline{x}$ o valor médio da amostra. O Erro máximo relativo ε é dado por:
$$\varepsilon = \frac{a}{\overline{x}} \tag{5.1}$$
- **Número mínimo de disparos para cada transição.** Pode ser utilizado como um limitante inferior no processo de simulação, dado que exige que cada transição dispare um determinado número de vezes para que a rodada de simulação acabe. Essa funcionalidade pode ser utilizada quando as taxas de disparo das diferentes transições da rede são muito diferentes.
- **Período de *warm-up*.** Refere-se ao número de disparos iniciais que precisam ser descartados para remoção do período transiente.
- **Tamanho da rodada.** Utilizado pela técnica das médias em lotes para determinar o tamanho de cada lote.
- **Tempo máximo de simulação (segundos)**. Tempo real medido em segundos para limitar a espera do usuário em relação ao processo de simulação. É utilizado caso se queira parar a simulação quando o tempo máximo de simulação for alcançado, mesmo que o processo de simulação ainda não tenha atingido um valor com o nível de confiança e erro desejados. Por exemplo, suponha que o usuário não queira esperar mais que duas horas para obter o resultado da simulação, ele deve especificar o tempo de 7200 segundos (duas horas) neste campo.
- **Habilitar experimentação de cenários.** Habilita a opção de experimentação de cenários. Ou seja, permite aos usuários realizar análise de sensibilidade.

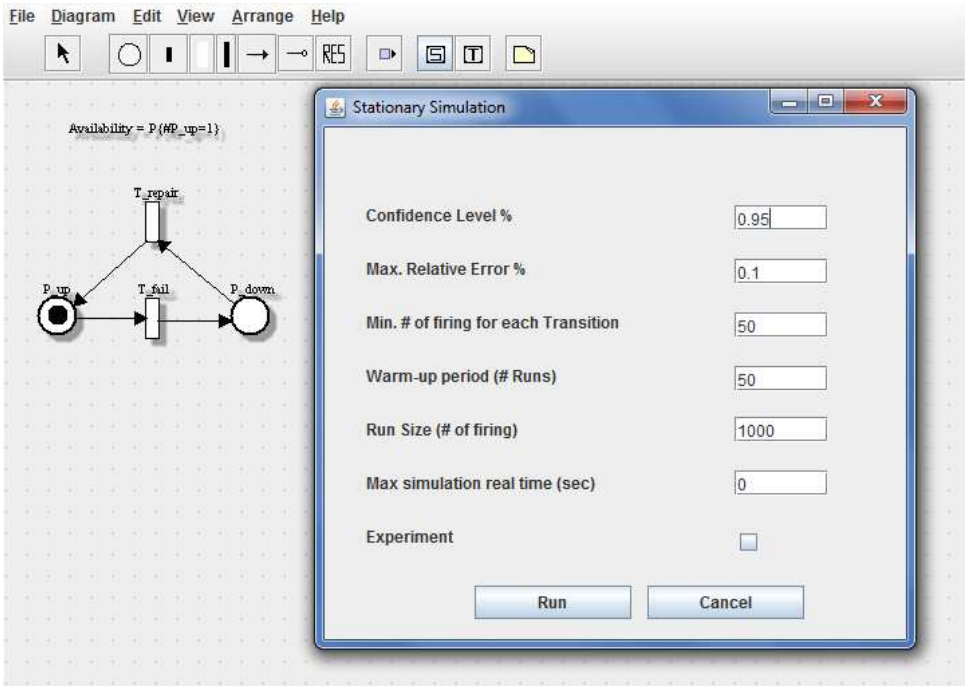
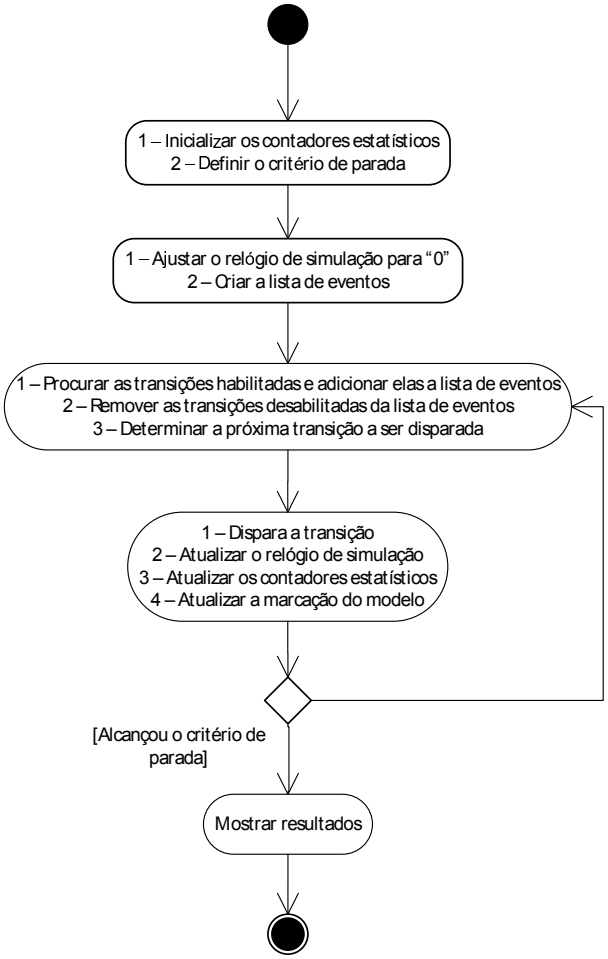


Figura 5.8: Parâmetros de Simulação.

Para um melhor entendimento, o processo de simulação é apresentado na Figura 5.9. Primeiramente, os contadores estatísticos (variáveis adotadas para armazenar informações estatísticas sobre o desempenho do sistema) são inicializadas e os parâmetros para o critério de parada são definidos. Após isso, o relógio da simulação (variável indicando o tempo atual da simulação) é ajustado para "0" e a lista de eventos é criada (ex.: lista que armazena as transições habilitadas). Em seguida, a simulação entra em um *loop* que corresponde à avaliação real do modelo DSPN. Em cada interação, as transições habilitadas são adicionadas à lista e as transições desabilitadas são removidas. A partir da lista de eventos, a *engine* de simulação seleciona a transição com o menor tempo, de acordo com uma variável aleatória, satisfazendo a distribuição da transição (ex.: distribuição exponencial ou determinística). A transição selecionada é disparada, e o clock da simulação, os contadores estatísticos e a marcação do modelo DSPN são atualizados. Se os critérios de simulação forem satisfeitos (ex.: o erro relativo é satisfeito), então a simulação é finalizada. A Figura 5.10 apresenta os resultados da simulação estacionária para o modelo DSPN apresentado na Figura 5.8. Note que a métrica adotada é a seguinte: $P\{\sharp P\_up = 1\}$. Isto é, a simulação calcula a probabilidade do modelo estar no lugar P_up quando ele alcançar a condição estacionária.

5.1.3 Integração com ASTRO/Mercury e TimeNet

O TimeNet (*Timed Net Evaluation Tool*) foi desenvolvido pela Universidade Técnica de Berlim no decorrer de vários projetos de pesquisa. Essa ferramenta tem sido largamente



Fonte: [SCT+13, p. 9]

Figura 5.9: Processo de Simulação.

usada no meio acadêmico para modelagem e avaliação de redes de Petri estocásticas com tempos de disparo das transições não exponencialmente distribuídos. Em outras palavras, o tempo de disparo das transições podem ser distribuído exponencialmente, determinístico, ou assumir uma outra distribuição mais geral [ZKHH06]. Por outro lado, o ASTRO/Mercury tem sido desenvolvido pelo grupo de pesquisa MODCS. Essa ferramenta suporta a avaliação de infraestrutura de *data centers* através de uma abordagem híbrida, a qual permite a utilização de RBD, DSPNs e modelos de alto nível para representar os *data centers* [SMT+10].

A ferramenta OpenMADS contém uma funcionalidade de exportar os modelos DSPN para as ferramentas ASTRO/Mercury e TimeNet. Tal integração permite que os projetistas possam considerar uma variedade enorme de algoritmos e métodos durante as análises (ver Figura 5.11). Além disso, essa integração pode ser usada a fim de realizar análises hierárquicas dos modelos DPSN obtidos pelo processo de mapeamento, uma vez que OpenMADS é limitado a realizar simulação estocástica das rede de Petri determinísticas

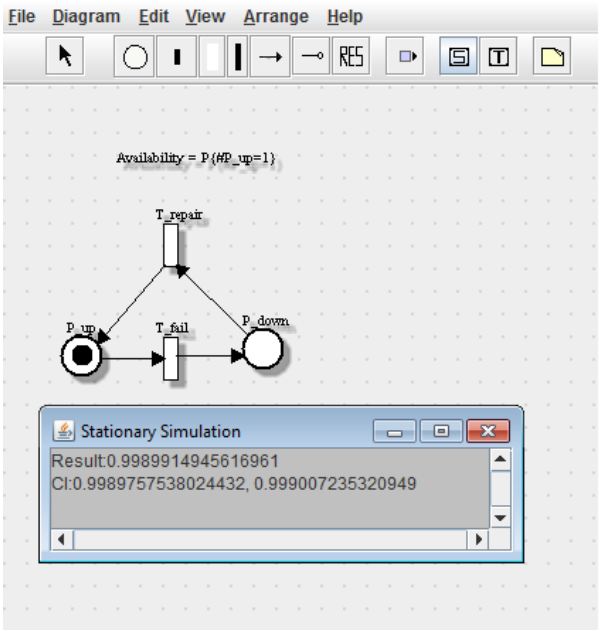


Figura 5.10: Avaliação Estacionária.

e estocásticas. É importante ser ressaltado que a ferramenta OpenMADS apenas gera os arquivos de entrada para as ferramentas ASTRO/Mercury e TimeNet. Ou seja, toda as análises e/ou simulações precisam ser realizadas através das próprias ferramentas.

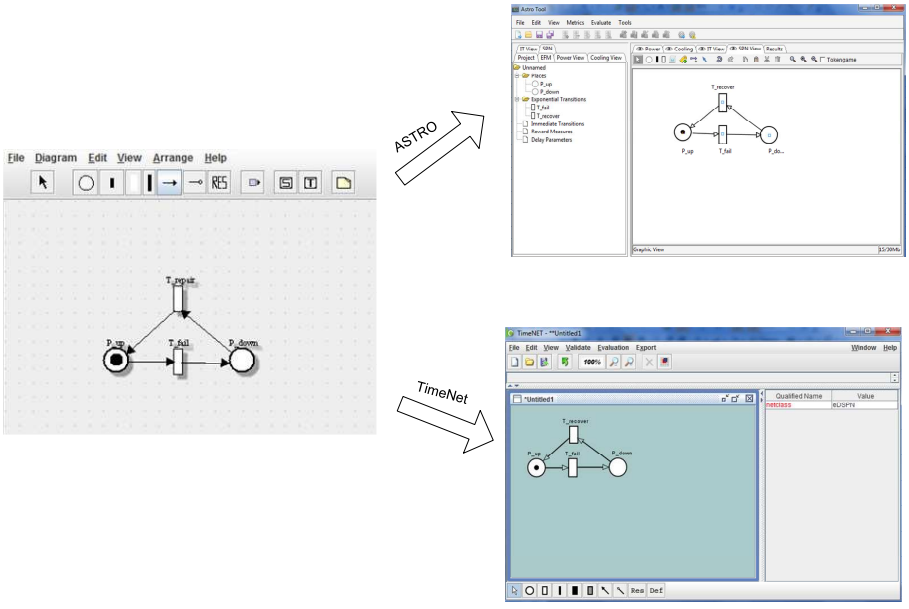


Figura 5.11: Integração do OpenMADS ao ASTRO/Mercury e TimeNET

5.2 UM EXEMPLO ILUSTRATIVO

A fim de demonstrar a aplicabilidade de OpenMADS, iremos apresentar um exemplo ilustrativo de um sistema de servidor *Web* composto por um balanceador de carga (*Load Balance (LB)*), seis servidores de aplicação *Web* e um Banco de Dados (BD) hospedados em um *data center* (ver Figura 5.12). Esse exemplo é um sistema distribuído típico, no qual cada servidor é executado na sua própria máquina virtual (VM) e divide os recursos do *data center*. O balanceador de carga distribui as requisições dos usuários uniformemente entre os servidores de aplicação *Web*, evitando que uma única VM se torne sobrecarregada enquanto as outras estão ociosas. Note que, no exemplo, consideramos seis máquinas virtuais executando a mesma aplicação *Web*. Três dessas VMs estão no estado *Hot Standby*. Isto é, se um dos servidores *Web* primário parar de funcionar, um servidor *Web Hot Standby* o substituirá. Uma vez que o servidor *Web* primário é reparado, então um servidor operacional retornará para o estado *Hot Standby*. Assumimos que o número mínimo de servidores *Web* operacionais é 3, sendo esse número de servidores mantido por um mecanismo de recuperação de interrupções automático. Se um servidor *Web* se tornar indisponível, o mecanismo de recuperação de interrupções, automaticamente, aumenta o número de servidores *Web* até 3 a fim de satisfazer os requisitos de performance. Ademais, o servidor de aplicação *Web* acessa o banco de dados, que está hospedado no mesmo *data center* para armazenar informações.

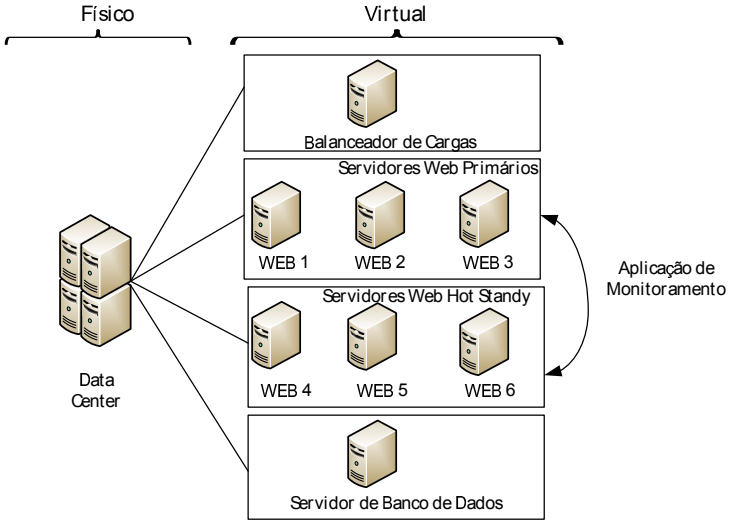


Figura 5.12: Sistema de Servidor *Web*.

5.2.1 Modelos da SysML e MARTE

As Figuras 5.13 e 5.14 apresentam os diagramas da SysML usados para representar o sistema de servidor *Web* hospedado num *data center*. A estrutura estática do sistema *Web* é descrita através do SysML-IBD da Figura 5.13 (d). O número "6" no cabeçalho do bloco *Servidor Web* significa o nível de redundância do componente do sistema. Múltiplos

compartimentos podem ser usados para descrever as características do bloco. O bloco *LB* possui o tempo médio até a falha (MTTF) de 8760 horas, e o tempo médio de reparo (MTTR) de 120 horas baseado nas propriedades do bloco. As alocações *allocatedFrom* e ≪*hosted*≫ são usadas para representar os relacionamentos entre os diagramas da SysML, conforme discutido anteriormente. Uma alocação *allocatedFrom* entre um bloco e um SysML-STM é usada para detalhar o bloco em termos de estados, enquanto uma alocação *allocatedFrom* entre um bloco e um SysML-AD é usada para representar os mecanismos de tratamento de interrupções que podem afetar os estados do sistema. A alocação ≪*hosted*≫ representa uma dependência de hospedagem entre os elementos do sistema. Se o elemento de hospedagem (*data center*) parar de funcionar, os elementos hospedados (balanceador de cargas, servidor *Web* e banco de dados) se tornam indisponíveis ao mesmo tempo.

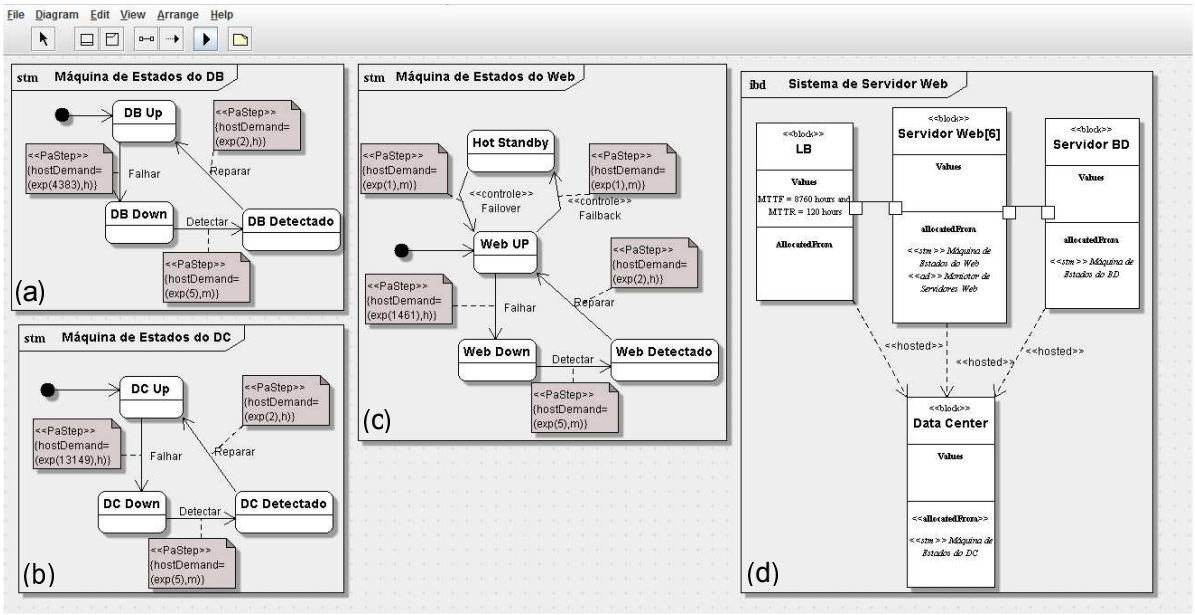


Figura 5.13: SysML-IBD e SysML-STMs para o Sistema de Servidor *Web*.

Os blocos *Servidor de BD*, *Data Center* e *Servidor Web* são detalhados, respectivamente, em termos de estados nas Figuras 5.13 (a), (b) e (c). Note que a ≪*stm*≫ *Máquina de Estados do BD* e a ≪*stm*≫ *Máquina de Estados do DC* possuem o comportamento de falha e reparo semelhante ao diagrama de estados descrito na Figura 4.5. Todavia, essas diagramas possuem diferentes anotação de disponibilidade associadas às transições. O comportamento de falha e reparo do *Servidor Web* (ver Figura 5.13 (c)) também é semelhante ao do servidor descrito na Figura 4.5. A diferença básica é que esse diagrama de estados possui um estado *Hot Standby* e duas transições (operações de *failover* e *failback*) conectando o estado *Web UP*. As transições *Failover* e *Failback* são estereotipados com ≪controle≫, significando que atividades do SysML-AD podem afetar os estados do sistema.

O SysML-AD usado para representar o comportamento de monitoramento do servidor

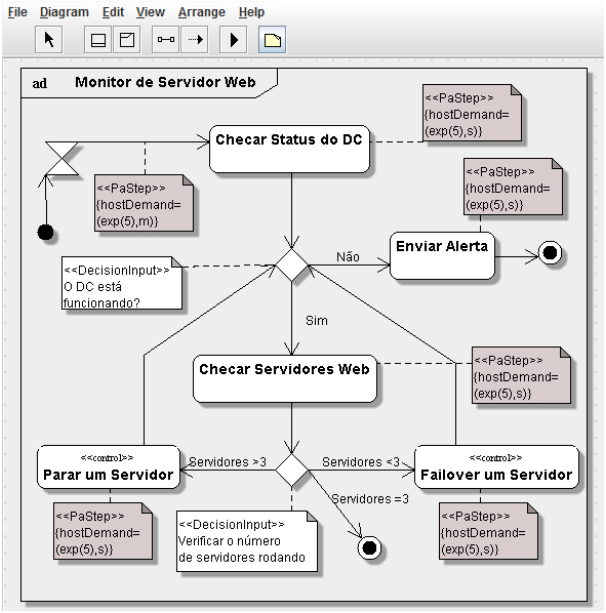


Figura 5.14: SysML-AD para o Sistema de Servidor *Web*.

Web é apresentado na Figura 5.14. Essa mecanismo de tratamento de interrupções é usada apenas para os *Servidores Web*. É importante ser ressaltado que consideramos que esse monitor está instalado num servidor a parte numa infraestrutura livre de falhas. O *Monitor de Servidor Web* checa o *data center* a cada 5 minutos. Se o *data center* não estiver funcionando adequadamente [Não], uma mensagem de alerta é enviada ao administrador de sistema que é responsável por intervenções manuais. Caso contrário [Sim], o número de servidores *Web* disponíveis é checado. Se o número de servidores *Web* disponíveis é 3, o mecanismo de monitoramento finaliza suas atividades e espera 5 minutos para checar novamente. Se o número de servidores é maior do que 3, o mecanismo de monitoramento para um dos servidores operacionais. Por fim, se o número de servidores é menor do que 3, o mecanismo de monitoramento aumenta o número de servidores *Web* até 3. Note que usamos o esterótipo ≪controle≫ para as atividades *Failover um Servidor* e *Parar um Servidor*. Isto é, os estados do *Servidor Web* (ver Figura 5.13(c)) são afetados pela execução dessas atividades.

5.2.2 Modelos DSPN

A Figura 5.15 apresenta os modelos DSPN do sistema de servidor *Web* das Figuras 5.13 e 5.14. Note que esses modelos foram obtidos através de um click no botão de mapeamento. O seguinte procedimento foi adotado para obter tais modelos automaticamente. Primeiramente, OpenMADS mapeia os diagramas da SysML, anotados de acordo com MARTE, em partes do modelo de disponibilidade chamados de componentes do modelo. Em seguida, OpenMADS compõe os componentes do modelo obtidos a partir do SysML-IBD e SysML-STM, seguindo as anotações de alocações e as propriedades dos blocos. O

modelo DSPN resultante é chamado de *Rede do Sistema*. Por fim, OpenMADS sincroniza a *Rede do Sistema* com a *Rede de Atividades* e apoia na geração das funções de guarda. A *Rede de Atividades* é obtida pelo mapeamento do SysML-AD. A Tabela 5.1 lista as funções de guarda utilizadas para o modelo resultante do mapeamento.

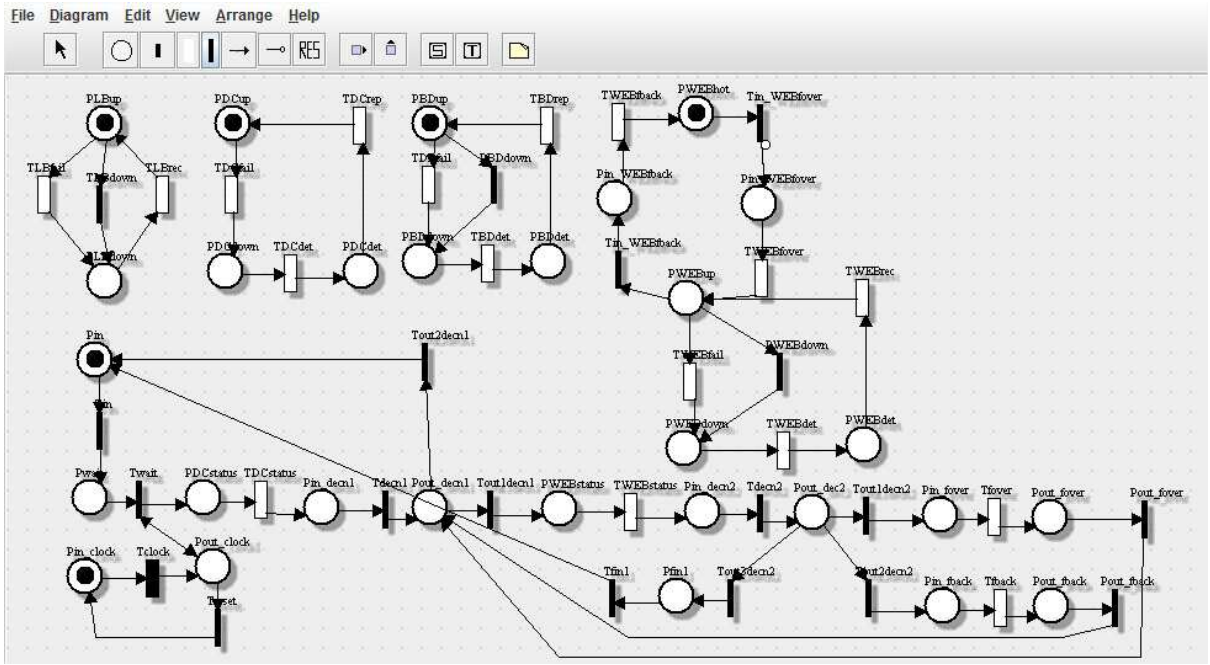


Figura 5.15: Modelos DSPN obtidos pelo Processo de Mapeamento.

5.2.3 Simulação Estocástica

A ferramenta OpenMADS foi usada para computar a disponibilidade dos modelos DSPN gerados pelo processo de mapeamento. Os valores *default* dos parâmetros usados na simulação são sumarizados na Tabela 5.2. Foram usados parâmetros de entrada arbitrários (porém razoáveis), visto que o propósito deste exemplo é mostrar a aplicabilidade da ferramenta. Note que todos esses parâmetros são definidos através das anotações de MARTE e das propriedades dos blocos. Neste exemplo, todas as transições são exponenciais com exceção da transição determinística T_{clock}. A Tabela 5.3 detalha a disponibilidade e o *downtime* do sistema de aplicação *Web* e dos componentes que compõem esse sistema. As disponibilidades são calculadas através das métricas definidas na Tabela 5.4. Note que a disponibilidade do sistema (A_{sys}) não é igual ao produto da disponibilidade de cada componente porque eles não são independentes. O resultado mostra que o Servidor *Web* possui a menor disponibilidade de todos os componentes do sistema, sendo, portanto, um dos gargalos da disponibilidade do ambiente modelado.

Também realizamos análise de sensibilidade dos models DSPN, em que o intervalo de monitoramento do servidor *Web* foi variado (T_{clock}). Como pode ser observado a partir da Figura 5.16, se o intervalo de monitoramento é próximo de zero, os servidores

Tabela 5.1: Funções de Guarda Geradas pelo Processo de Mapeamento.

Ação	Guarda	Função
Nos de decisão	G_{out1_dec1}	$\sharp P_{DCup}$=1
	G_{out2_dec1}	$\sharp P_{DCup}$=0
	G_{out1_dec2}	$\sharp P_{WEBup}$>3
	G_{out2_dec2}	$\sharp P_{WEBup}$<3
	G_{out3_dec2}	$\sharp P_{WEBup}$=3
Sincronização	$G_{in\_WEBFover}$	$\sharp p_{in\_Fover}$=1
	G_{in_Fover}	$\sharp P_{in\_WEBFover}$=1
	$G_{out\_WEBFover}$	$\sharp P_{out\_Fover}$=1
	G_{out_Fover}	$\sharp P_{in\_WEBFover}$=0
	$G_{in\_WEBFback}$	$\sharp p_{in\_Fback}$=1
	G_{in_Fback}	$\sharp P_{in\_WEBFback}$=1
	$G_{out\_WEBFback}$	$\sharp P_{out\_Fback}$=1
	G_{out_Fback}	$\sharp P_{in\_WEBFback}$=0
Composição dos Modelos	$G_{WEBdown}$	(♯('P_{DCdown}=1
	G_{DBdown}	$\sharp P_{DCdown}$=1
	G_{LBdown}	$\sharp P_{DCdown}$=1
	G_{WEBdet}	$\sharp P_{DCup}$=1
	G_{DBdet}	$\sharp P_{DCup}$=1
	G_{LBdet}	$\sharp P_{DCup}$=1

Web são checados constantemente evitando o *downtime* e, assim, resulta numa maior disponibilidade do sistema. Por outro lado, a medida que intervalo de monitoramento aumente, a disponibilidade do sistema diminui, já que as interrupções nos servidores *Web* podem levar mais tempo para serem detectadas. O *downtime* anual aumenta de 12 horas para 15,5 horas, se modificarmos o intervalo de monitoramento de 5 minutos para 200 minutos. Esse é um exemplo de várias conclusões importantes que podemos obter usando OpenMADS.

Tabela 5.2: Descrição dos Parâmetros de Entrada e seus Valores *Default*.

Parâmetro	Transição	Valor [horas]
Tempo Médio até a Falha do *Data Center*	T_{DCfail}	13149
Tempo de Reparo do *Data Center*	T_{DCrec}	3
Tempo de Detecção do *Data Center*	T_{DCdet}	0.08
Tempo Médio até a Falha do Banco de Dados	T_{DBfail}	4383
Tempo de Reparo do Banco de Dados	T_{DBrec}	2
Tempo de Detecção do Banco de Dados	T_{DBdet}	0.08
Tempo Médio até a Falha do Balanceador de Cargas	T_{LBfail}	8760
Tempo de Reparo do Balanceador de Cargas	T_{LBrec}	2
Tempo Médio até a Falha do Servidor Web	$T_{WEBfail}$	1461
Tempo de Reparo do Servidor Web	T_{WEBrec}	2
Tempo de Detecção do Servidor We	T_{WEBdet}	0.08
Tempo de *Failover* do Servidor Web	$T_{WEBfover}$	0.02
Tempo de *Failback* do Servidor Web	$T_{WEBfback}$	0.02
Tempo das Atividades	T_X	1.39×10^{-3}
Intervalo de Monitoramento	T_{clock}	0.42

Tabela 5.3: Disponibilidade em Estado Estacionário e *Downtime*.

	Disponibilidade	Downtime (horas por ano)
Balanceador de Cargas	0,9998313	1,477812
Servidor Web	0,9989158	9,497242
Banco de Dados	0,9993347	5,828028
Data Center	0,9996610	2,969640
Sistema (A_{sys})	0,9986098	12,178152

Tabela 5.4: Métricas para Calcular a Disponibilidade em Estado Estacionário.

Métrica	Função
Balanceador de Carga (R_{LB})	P{$\sharp P_{LBup}$=1}
Servidores Web (R_{WEB})	P{$\sharp P_{WEBup}$>2}
Banco de Dados (R_{BD})	P{$\sharp P_{DBup}$=1}
Data Center (R_{DC})	P{$\sharp P_{DCup}$=1}
Sistema (A_{sys})	P{(($\sharp P_{LBup}$=1) AND ($\sharp P_{DCup}$=1) AND ($\sharp P_{DBup}$=1) AND ($\sharp P_{WEBup}$>2))}

5.3 CONSIDERAÇÕES FINAIS

O mapeamento das especificações dos sistemas distribuídos, incluindo os mecanismos de tratamento de interrupções pode ser complexo e propenso a erros. Isto é, para que ele seja

mais praticável, o suporte de ferramentas pode ser necessário. Dessa forma, este capítulo introduziu a ferramenta OpenMADS que pode ser usada para auxiliar os projetistas na modelagem, no mapeamento e na análise das especificações dos sistemas distribuídos. O grande desafio do desenvolvimento desse ferramenta foi justamente em lidar com a complexidade da mesma, devido, principalmente, às conversões requeridas (Diagramas da SysML -> Redes de Petri), como também à disponibilização de uma interface gráfica online para os usuários.

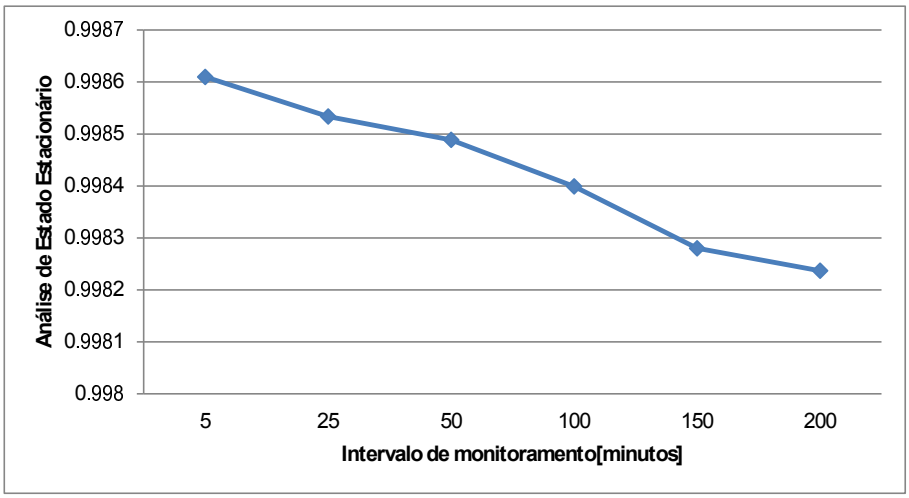


Figura 5.16: Análise de Sensibilidade.

CAPÍTULO 6

ESTUDO DE CASOS

Education is the most powerful weapon which you can use to change the world.

—NELSON MANDELA

Os exemplos apresentados neste capítulo são mecanismos de tratamento de interrupções reais implantado nos sistemas distribuídos. Neste sentido, procura-se demonstrar a aplicabilidade do *framework* que almeja auxiliar os projetistas, os quais não possuem (ou possuem pouca) expertise em modelagem estocástica a projetar e estudar as infraestruturas dos sistemas distribuídos e os mecanismos de tratamento de interrupções, a partir de especificações descritas em SysML e MARTE. Três estudos de casos são apresentados neste capítulo. O primeiro consiste do estudo da nuvem *open-source* Eucallyptus. Nesse estudo, são levados em conta os *bugs* de envelhecimento e seus respectivos mecanismos de mitigação de interrupções. Também foi utilizada uma abordagem hierárquica para os modelos obtidos pelo processo de mapeamento, visto que os mesmos podem ser decompostos em submodelos hierárquicos. O segundo consiste de uma nuvem no modelo público de infraestrutura como serviço (ex.: Amazon Web Services EC2), em que mecanismos de tratamento de interrupções próprios desse modelo são levados em consideração, tais como: zona de isolamento e mecanismo de auto *scaling*. Esses mecanismos são usados para garantir alta disponibilidade e continuidade dos serviços para os SD, mesmo diante de diferentes tipos de interrupções. O último consiste no estudo de um sistema de recuperação de interrupções severas utilizando uma infraestrutura de nuvem privada. Nesse estudo de caso, um mecanismo de monitoramento de desastres é utilizado para detectar as ocorrências de interrupções severas. Caso um desastre tenha sido detectado, o mecanismo de monitoramento realiza o *failover* do sistema hospedado na infraestrutura primária (*data center*) para uma infraestrutura secundária (nuvem privada), a fim de garantir interoperabilidade e disponibilidade dos serviços providos. Para esse estudo de caso, toda a infraestrutura real do sistema de recuperação de desastres foi montada e experimentos foram realizados para analisar a aplicabilidade do *framework* proposto.

6.1 INFRAESTRUTURA EUCALYPTUS

O Eucalyptus (*Elastic Utility Computing Architecture Linking Your Programs To Useful Systems*) é uma infraestrutura de *software* de código aberto que implementa estilos escaláveis de IaaS de nuvens privadas e híbridas [NWG+09]. A plataforma Eucalyptus surgiu a partir de um projeto desenvolvido no departamento de ciência da computação da Universidade da Califórnia, Santa Bárbara, com o intuito de realizar pesquisas na

área de computação em nuvem. Essa plataforma oferece serviços como armazenamento, virtualização de sistemas operacionais, *clusters*, entre outros. O Eucalyptus é compatível com várias distribuições Linux, tais como: Ubuntu, Red Hat Enterprise Linux (RHEL), CentOS, SUSE Linux Enterprise Server (SLES), openSUSE, Debian e Fedora e com uma variedade de tecnologias de virtualização, i.e., VMware, Xen e KVM. Há cinco componentes de alto nível da arquitetura Eucalyptus: o Controlador da Nuvem (consiste da porta de entrada na nuvem para os administradores, desenvolvedores e usuário final), o Controlador do *Cluster* (responsável pelo gerenciamento de um ou mais controlador do nó), o Controlador do Nó (gerencia o ciclo de vida das VMs em operação no nó), o Controlador de Armazenamento (fornece armazenamento permanente para ser usado por instâncias de VMs), e o Walrus (consiste de um serviço de armazenamento compatível com o Amazon S3) [Dan13].

A Figura 6.1 apresenta a infraestrutura Eucalyptus adotada neste estudo de caso. Essa nuvem privada tem uma estrutura simples, composta por um *FrontEnd*, com um controlador de armazenamento e um controlador de nó. Assim, considerou-se que o controlador da nuvem, o controlador do *cluster* e o *walrus* foram instalados numa mesma máquina física (*FrontEnd*). Já o controlador do nó e o controlador de armazenamento foram, cada um, considerados em máquinas físicas distintas, pois eles têm que gerenciar os recursos da máquina física para as respectivas VMs instaladas, bem como realizar o armazenamento pertencente as VMs. De acordo com os resultados preliminares apresentados em [AMM+11], o controlador do nó pode demonstrar um desempenho degradado e uma taxa de falha crescente em decorrência da criação e destruição de VMs (*bugs* de envelhecimento). Dessa forma, visando impedir a ocorrência de falhas devido esse comportamento de envelhecimento, os mecanismos de mitigação, chamados de rejuvenescimento, são executados periodicamente sobre o controlador do nó a fim de evitar o *downtime* da infraestrutura Eucalyptus. Este estudo de caso tem o objetivo de demonstrar que o rejuvenescimento baseado em tempo (mecanismo de mitigação de *bugs* de envelhecimento) pode ser efetivo para garantir alta disponibilidade e continuidade dos serviços providos pela a infraestrutura Eucalyptus. Ademais, uma abordagem hierárquica para os modelos obtidos pelo processo de mapeamento é adotada, onde os submodelos independentes são separados e a disponibilidade de todo o sistema é calculada usando o RBD.

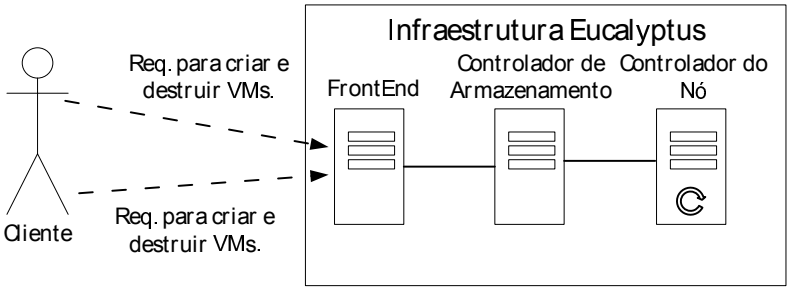


Figura 6.1: Infraestrutura Eucalyptus.

6.1.1 Modelos da SysML Anotados

O SysML-IBD da infraestrutura Eucalyptus é apresentado na Figura 6.2. Os blocos representam os elementos básicos do sistema, i.e., *FrontEnd*, Controlador de Armazenamento e Controlador do Nó. Os compartimentos dos blocos são usados para representar as propriedades dos blocos (MTTF e MTTR), bem como as alocações deles a outros diagramas (*allocatedFrom*). O bloco *Controlador do Nó* é detalhado em termos de estados pelo SysML-STM da Figura 6.3. O controlador inicia no estado *Operational*. Uma vez nesse estado, o controlador pode falhar ou sofrer envelhecimento (*aging*). Se o mesmo falhar, o controlador assume o estado *Falhou*. Quando a falha for detectada, então o controlador assume o estado *Detectado*. Após isso, o controlador é reparado e retorna para o estado *Operational*. Se o controlador assumir o estado de *Envelhecido*, o mesmo pode ser rejuvenescido e retorna para o estado *Operational*. Note que as anotações do *profile* MARTE foram usadas para especificar os tempos associados aos eventos de falha com ou sem *aging*, de detecção e de reparo. Além disso, o rejuvenescimento é executado de acordo com o esterótipo ≪controle≫, que é acionado pelo mecanismo de mitigação apresentado a seguir.

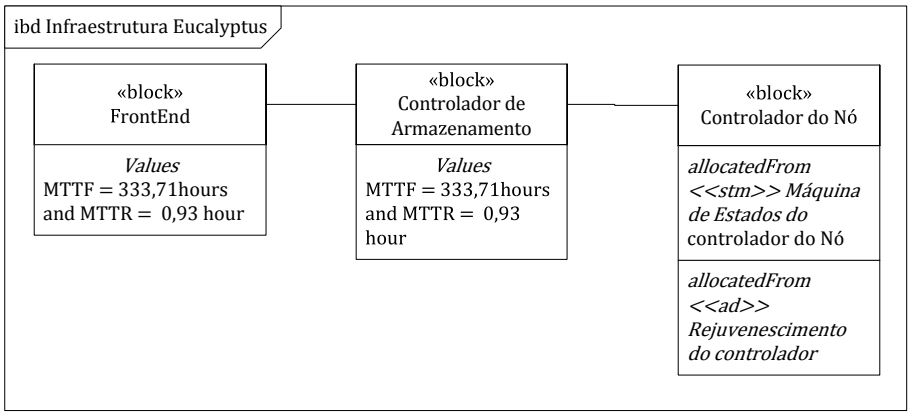


Figura 6.2: SysML-IBD da Infraestrutura Eucalyptus.

Neste trabalho, foi adotado o mecanismo de mitigação de *bugs* de envelhecimento baseado em tempo. Ou seja, o controlador do nó é rejuvenescido em intervalos de tempo fixos. Esse mecanismo é apresentado na Figura 6.4. É importante ser ressaltado que o rejuvenescimento baseado em tempo é aplicado apenas ao controlador do nó, pois ele é o único a apresentar indícios de envelhecimento de *software*, conforme descrito em [AMM+11]. A ação de evento de tempo (elemento descrito através de uma ampulheta) representa que a atividade *Checar Status do Controlador* é executada a cada 24 horas. Esse tempo é especificado através de MARTE. Caso o servidor não esteja funcionando [Não], uma mensagem de alerta é enviada para o administrador do servidor que é responsável por executar alguma intervenção manual e o SysML-AD alcança o seu fim. Caso contrário [Sim], o mesmo será rejuvenescido de acordo com a atividade *Rejuvenescimento* e o SysML-AD alcança o seu fim. As atividades *Checar Status do Controlador*, *Enviar Mensagem de Alerta* e *Rejuvenescimento* possuem o tempo exponential de 1 se-

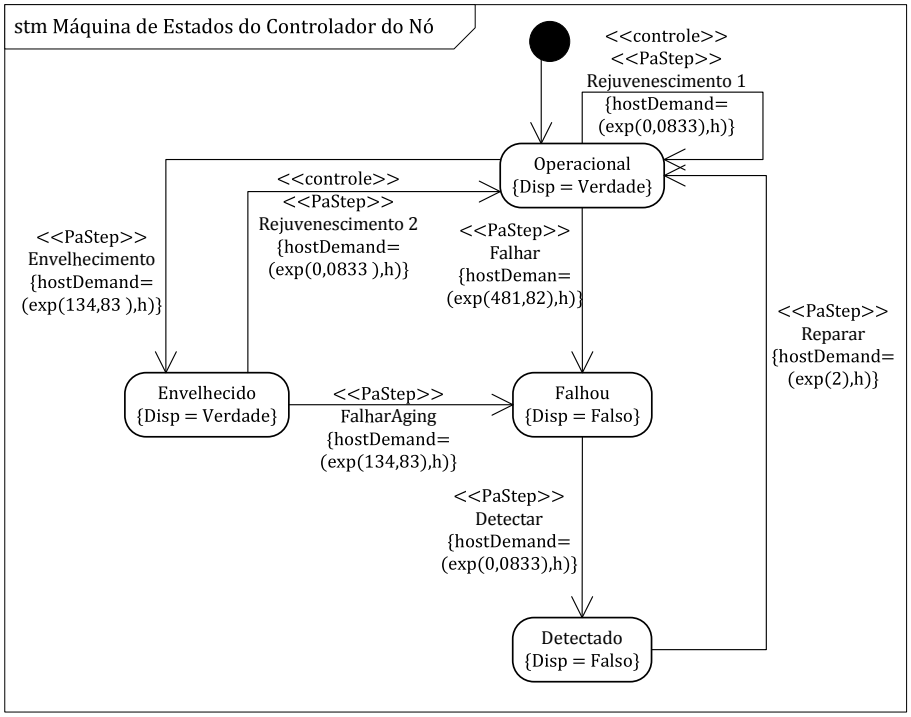


Figura 6.3: SysML-STM do Controlador do Nó.

gundo cada. Esses tempos também são especificados através de MARTE. A origem dos parâmetros utilizados neste estudo de caso é discutida na Seção 6.1.3. Ademais, o estereótipo ≪controle≫ foi usado para especificar as atividades que afetam os estados do servidor descritos pelo SysML-STM. Nesse caso, o controlador descrito na Figura 6.3 pode mudar dos estados *Operacional* e *Envelhecido* para o estado *Operacional* pela execução da atividade *Rejuvenescimento* (ver Figura 6.4). Note que o controlador, mesmo estando no estado *Operacional*, pode ser rejuvenescido devido ao mecanismo de rejuvenescimento baseado em tempo adotado neste caso.

6.1.2 Mapeamento dos Diagramas da SysML Anotados em DSPNs

A Figura 6.5 descreve os modelos DSPN obtidos pelo processo de mapeamento dos diagramas da SysML que representam a infraestrutura Eucalyptus adotada neste estudo de caso. Esses modelos foram obtidos conforme os passos definidos no Capítulo 4. Primeiramente, os diagramas da SysML, anotados através de MARTE, são mapeados em DSPNs. Após isso, as DSPNs obtidas são compostas, de acordo com a alocação *allocatedFrom*. É importante ser ressaltado que a composição usando a alocação *allocatedFrom* consiste na substituição do modelo DSPN do SysML-IBD pelo modelo DSPN do SysML-STM, já que o último possui mais detalhes. Por último, a *Rede do Sistema* é sincronizada com a *Rede de Atividades* através da definição de funções de guarda.

A Tabela 6.1 apresenta as funções de guarda utilizadas para a sincronização entre

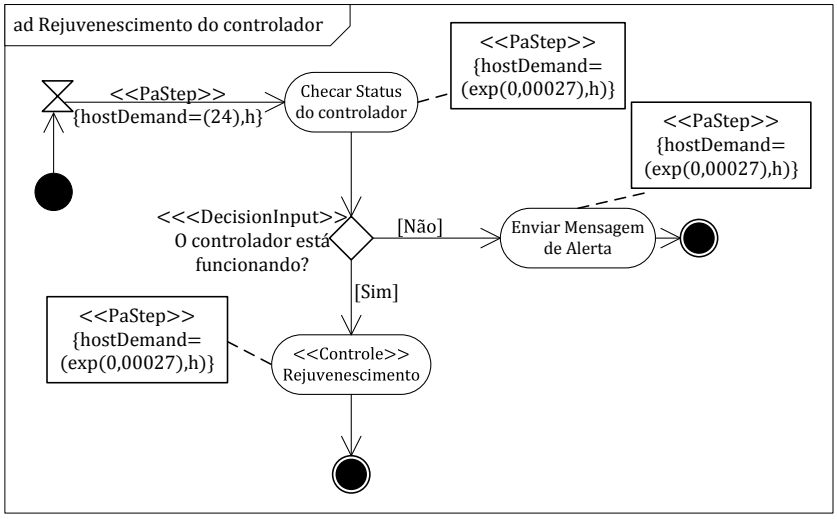


Figura 6.4: SysML-AD para o Rejuvenescimento Baseado em Tempo.

a *Rede do Sistema* (obtida pela composição do bloco *Controlador do Nó* e o ≪*stm*≫ *Máquina de Estados do Controlador do Nó*) e a *Rede de Atividades*. Note que a transição T_{SVrej} da *Rede de Atividades* é sincronizada com as transições T_{rejuv2} e T_{rejuv1} da *Rede do Sistema*, visto que a atividade *Rejuvenescimento* do SysML-AD pode habilitar o disparo dos eventos *Rejuvenescimento 1* e *Rejuvenescimento 2* do SysML-STM. Como explicado no Capítulo 4, as transições T_{rejuv2} e T_{rejuv1} são expandidas com uma transição imediata e um lugar com respectivos arcos. Após isso, são criadas funções de guarda para todas as transições relacionadas (G_{in_rejuv1}, G_{out_rejuv1}, G_{in_rejuv2}, G_{out_rejuv2}, G_{in_SVrej} e G_{out_SVrej}). Como o rejuvenescimento pode ser executado a partir de dois lugares (P_{in_rejuv1} e P_{in_rejuv2}), as funções de guarda G_{in_SVrej} e G_{out_SVrej} foram modificadas para considerar ambos os lugares. Por fim, duas funções de guarda (G_{out1_dec1} e G_{out2_dec1}) foram definidas para determinar o fluxo de execução das atividades de acordo com a condição do nó de decisão.

Tabela 6.1: Funções de Guarda para a Sincronização entre a *Rede do Sistema* e a *Rede de Atividades*.

Guarda	Função
G_{out1_dec1}	($\sharp P_{up}$=1) OR ($\sharp P_{enve}$=1)
G_{out2_det1}	($\sharp P_{up}$=0) AND ($\sharp P_{enve}$=0)
G_{in_rejuv1}, G_{in_rejuv2}	$\sharp p_{in\_SVrej}$=1
G_{in_SVrej}	($\sharp P_{in\_rejuv1}$=1) OR ($\sharp P_{in\_rejuv2}$=1)
G_{out_rejuv2}, G_{out_rejuv1}	$\sharp P_{out\_SVrej}$=1
G_{out_SVrej}	($\sharp P_{in\_rejuv1}$=0) AND ($\sharp P_{in\_rejuv2}$=0)

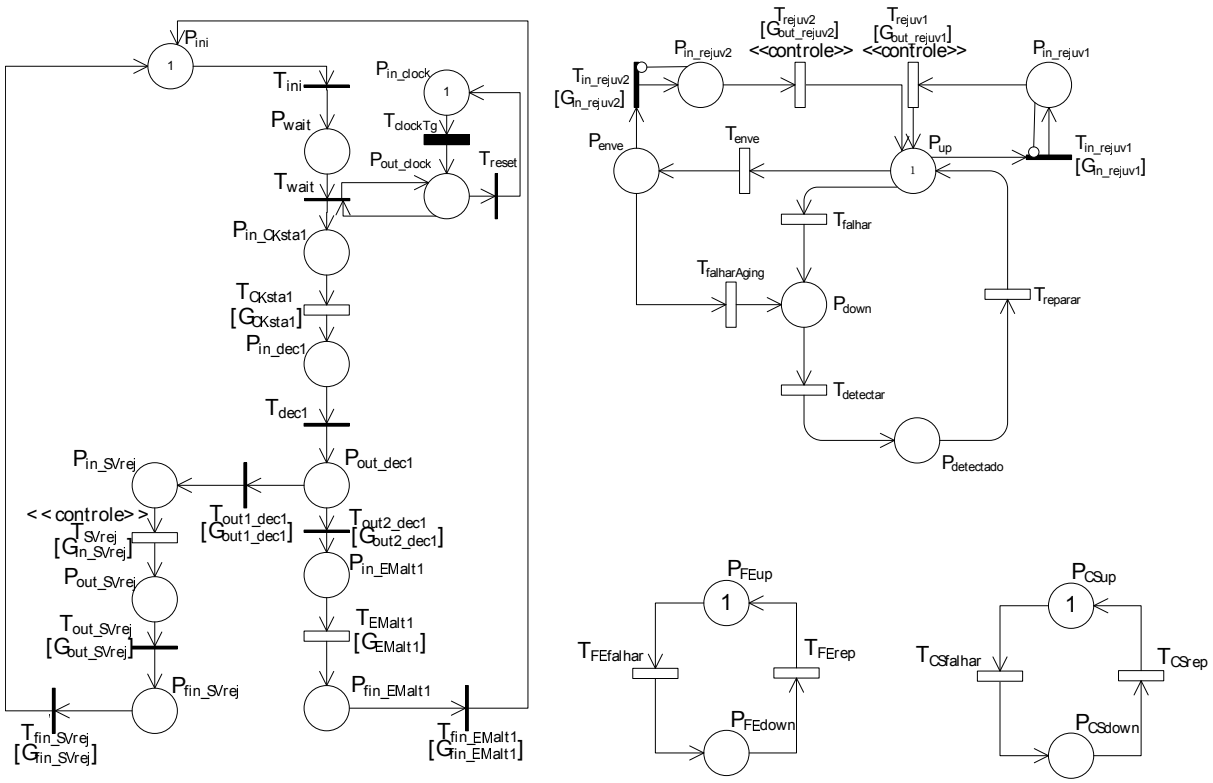


Figura 6.5: DSPNs Resultantes do Processo de Mapeamento dos Diagramas da SysML que Representam a Infraestrutura Eucalyptus.

6.1.3 Análise Numérica

Visto que o funcionamento dos componentes básicos da arquitetura adotada (*FrontEnd*, Controlador de Armazenamento e Controlador do Nó), não afeta a execução uns dos outros (subsistemas independentes), então, a abordagem hierárquica foi adotada a fim de facilitar na análise dos modelos. Os modelos DSPN obtidos pelo processo de mapeamento são separados em submodelos independentes (ver Figure 6.6) e as disponibilidades são calculadas para cada um deles. Os parâmetros usados na análise numérica dos modelos obtidos pelo processo de mapeamento são sumarizados na Tabela 6.2. Note que esses valores são provenientes das anotações atribuídas aos elementos dos diagramas através do *profile* MARTE e das propriedades dos blocos (MTTF e MTTR). Foram usados parâmetros de entrada baseados em experimentos descritos em [AMM+11, Dan13]. Neste estudo de caso, todas as transições são exponenciais com exceção da transição determinística $T_{clockTg}$. As métricas de disponibilidade usadas para cada submodelo estão descritas na Tabela 6.3. Essas métricas são definidas baseadas nas expressões booleanas associadas aos estados do diagramas do SysML-STM. A Tabela 6.4 lista os resultados para cada submodelo representado por S_1 (FrontEnd), S_2 (Controlado de Armazenamento) e S_3 (Controlado do Nó). O resultado mostra que o controlador do nó possui a menor disponibilidade de todos os componentes do sistema, sendo, portanto, um dos gargalos da disponibilidade do ambiente modelado.

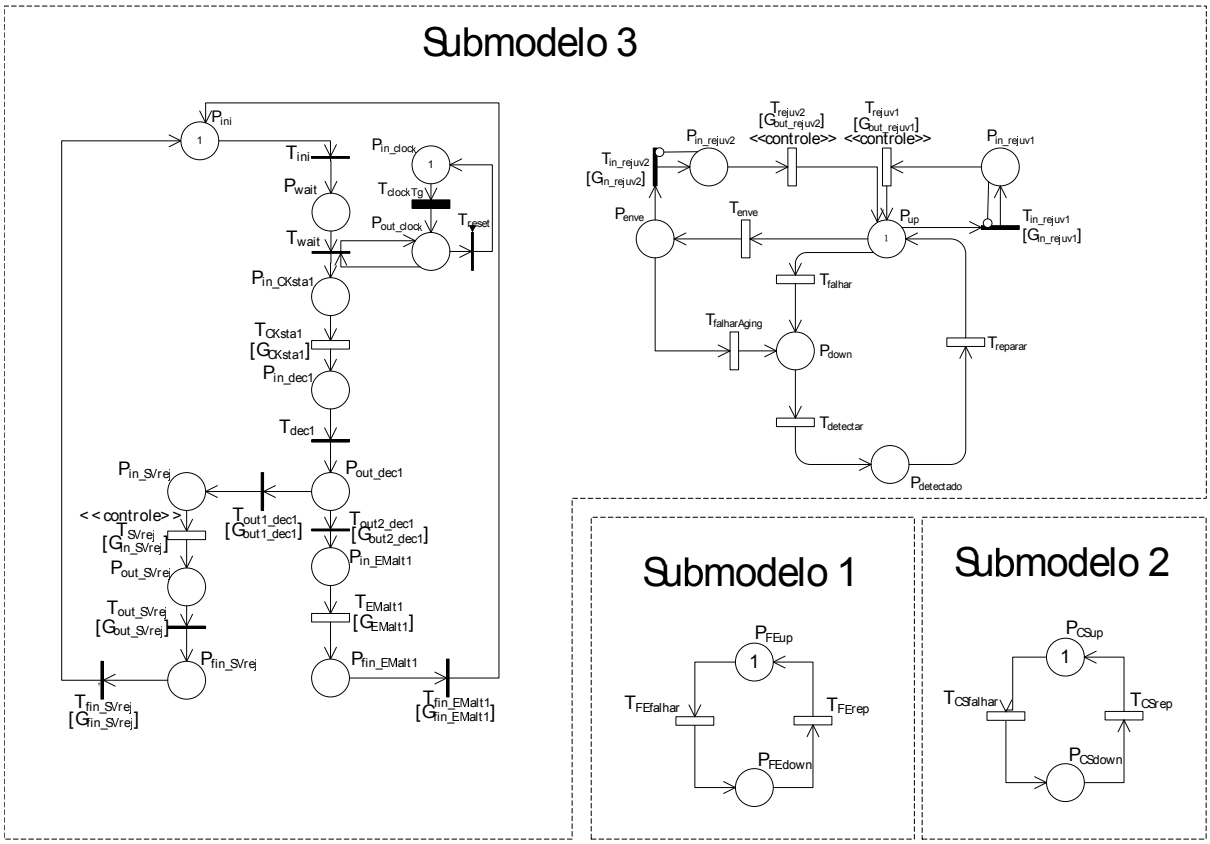


Figura 6.6: Submodelos Hierárquicos.

Tabela 6.2: Descrição dos Parâmetros de Entrada e seus Valores *Default*.

Parâmetros	Transição	Valor [horas]
Tempo Até a Falha do FrontEnd	$T_{FEfalhar}$	333,7
Tempo de Reparo do FrontEnd	T_{FErep}	0,93
Tempo Até a Falha do Controlador de Armazenamento	$T_{CSfalhar}$	333,7
Tempo de Reparo do Controlador de Armazenamento	T_{CSrep}	0,93
Tempo de Rejuvenescimento	T_{reju1}, T_{reju2}	0,0833
Tempo Até a Falha do Controlador do Nó	T_{Falhar}	481,82
Tempo de Envelhecimento do Controlador do Nó	T_{enve}	134,833
Tempo Até a Falha do Controlador do Nó	$T_{FalharAging}$	134,833
Tempo de Detecção da Falha do Controlador do Nó	$T_{detectar}$	0,0833
Tempo de Reparo do Controlador do Nó	$T_{reparar}$	2
Checar Status do Controlador do Nó	T_{CKsta1}, T_{CKsta2}	0,00027
Envio de Mensagem de Alerta	T_{EMalt1}, T_{EMalt2}	0,00027
Intervalo de Disparo do Rejuvenescimento	$T_{clockTg}$	24

Tabela 6.3: Métrica para Avaliação Estacionária.

Métrica	Função
Submodelo 1 (S_1)	P{$\sharp P_{FEup}$=1}
Submodelo 2 (S_2)	P{$\sharp P_{CSup}$=1}
Submodelo 3 (S_3)	P{($\sharp P_{up}$=1) OR ($\sharp P_{enve}$=1)}

Tabela 6.4: Resultados Obtidos.

	S_1	S_2	S_3	A_{sys}
Disponibilidade	0,99722081	0,99722081	0,99137928	0,985876475
Downtime Anual (hrs)	23,34	23,34	75,52	123,72

Após a obtenção das disponibilidades dos submodelos, as mesmas são usadas como entrada para cada bloco do modelo RBD (ver Figure 6.7). Note que cada submodelo é representado por um respectivo bloco em série, já que o sistema todo falha se um controlador falhar. Por fim, o RBD é usado para calcular a disponibilidade de todo o sistema (A_{sys}). A disponibilidade desta arquitetura não redundante é mostrada na Tabela 6.4. Observe que a abordagem hierárquica é muito importante quando se está lidando com modelos grandes (explosão do espaço de estado), pois ela evita os problemas de *largness* e *stifness*. No entanto, só pode ser usada se existirem subsistemas independentes. A fim de analisar o resultado obtido usando essa abordagem hierárquica, este resultado foi comparado com o resultado dos modelos não hierárquicos (composição exata). A diferença entre os valores foi menor que 1%, conforme apresentada na Tabela 6.5.

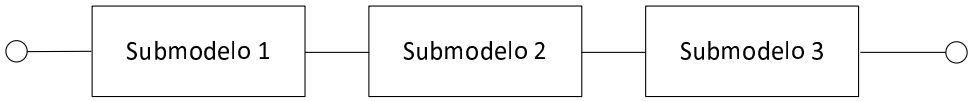


Figura 6.7: Modelo RBD da Infraestrutura Eucalyptus.

Tabela 6.5: Validação.

Abordagem	Resultados
Hierárquica	0,98587647
Não Hierárquica	0,98587648

A Tabela 6.6 apresenta um comparativo da disponibilidade da infraestrutura sob análise, em que foi levado em consideração que o controlador do nó é rejuvenescido a cada 24 horas (rejuvenescimento baseado em tempo), como também não é realizado o rejuvenescimento do controlador. A disponibilidade da infraestrutura, considerando o

rejuvenescimento baseado em tempo, foi maior do que não executar essa atividade administrativa de mitigação de *bugs* de envelhecimento. Isto é, de fato, o rejuvenescimento baseado em tempo resulta no aumento da disponibilidade e garante continuidade dos serviços providos pela a infraestrutura Eucalyptus. No entanto, a disponibilidade obtida (0,98587647) pode não ser o valor ideal. Assim, a análise de sensibilidade, que consiste no estudo do efeito da variação de parâmetros, foi conduzida com o intuito de encontrar o tempo de rejuvenescimento ótimo para a infraestrutura analisada.

Tabela 6.6: Disponibilidade Estacionária e *Downtime*

Mecanismo de Mitigação	Disponibilidade	*Downtime*
Rejuv. baseado em tempo (24 hrs)	0,98587647	123,72
Sem rejuvenescimento	0,98471306	133,91

A Figura 6.8 apresenta o resultado da análise de sensibilidade, no qual o intervalo de disparo do rejuvenescimento é variado. Se o intervalo de disparo do rejuvenescimento é próximo de zero, então o servidor é rejuvenescido mais do que o necessário e, consequentemente, resulta em baixa disponibilidade. À medida que o intervalo de rejuvenescimento aumenta, o servidor atinge um valor ótimo. O intervalo de rejuvenescimento ideal é de 60 horas, sendo o tempo *downtime* por ano de 113,88 horas. Se o intervalo de rejuvenescimento vai além do valor ótimo, a disponibilidade permanece relativamente estável, mas ela começa a cair porque as falhas do servidor têm influência maior sobre a disponibilidade do que a operação de rejuvenescimento.

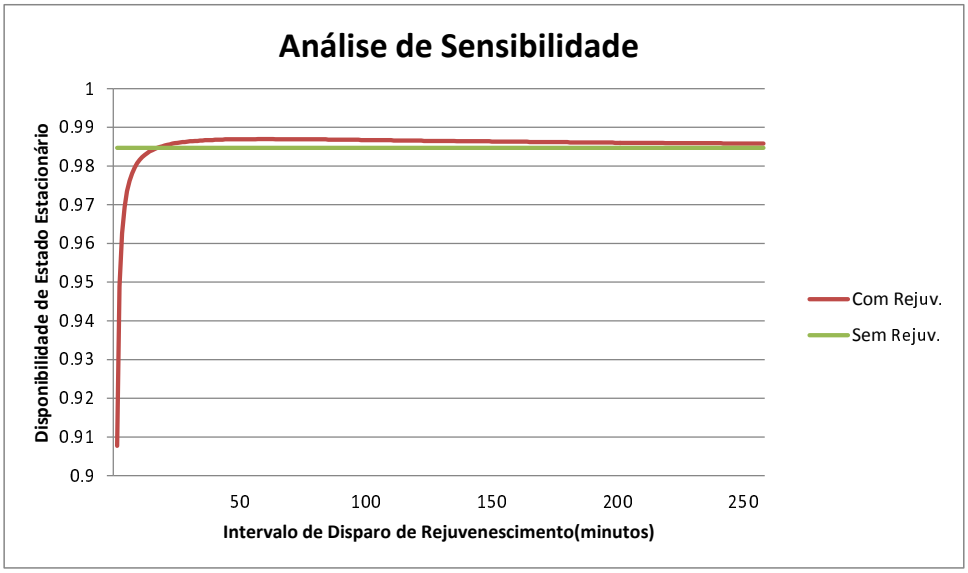


Figura 6.8: Análise de Sensibilidade.

6.2 SISTEMA WEB HOSPEDADO NUMA INFRAESTRUTURA NAS NUVENS

Neste estudo de caso, nós consideramos um sistema de aplicação *Web* hospedado numa infraestrutura nas nuvens. Mais especificamente, iremos focar em uma nuvem IaaS (Infraestrutura como Serviço) provido por um fornecedor de nuvens privadas. O termo IaaS refere-se ao fornecimento de infraestrutura computacional como um serviço. Essas infraestruturas, por sua vez, provêm um conjunto de vantagens em relação ao modelo tradicional para disponibilizar serviços de TI, tais como elasticidade e alta disponibilidade dos recursos computacionais baseado na tecnologia de virtualização dos servidores. As seguintes características são especialmente importantes em termos de tratamento de interrupções dos serviços em nuvens: (i) zona de isolamento de falha (são áreas separadas dentro de uma região que foram arquitetadas para que falhas em uma zona não afetem os serviços de outra zona), (ii) função de auto *scaling* (permite escalar o número de instâncias de uma aplicação para cima ou para baixo, de acordo com as regras definidas pelos os usuários) e (iii) balanceador de cargas (distribui a carga de trabalho entre as instâncias de uma aplicação hospedadas nas zonas de isolamento de falha). Dessa forma, o objetivo deste estudo de caso é demonstrar a eficácia desses mecanismos de tratamento de interrupções adotados em nuvens públicas (ex.: Amazon Web Services EC2).

Baseado nas características de uma nuvem IaaS, a Figura 6.9 apresenta a infraestrutura adotada neste estudo de caso. Essa infraestrutura consiste de quatro instâncias de uma aplicação *Web* e duas instâncias de um servidor de Banco de Dados (BD). Essas instâncias estão hospedadas em duas zonas de isolamento distintas. Neste estudo de caso, foi assumido que cada instância é executada na sua própria máquina virtual. O Balanceador de Carga (LB) é responsável por distribuir as requisições dos usuários entre as instâncias da aplicação *Web*. Em cada zona de isolamento de falha, o número de instâncias operacionais é mantido pelo mecanismo de tratamento de interrupções, chamado de mecanismo de auto *scaling*. Esse mecanismo é configurado para sempre manter quatro instâncias rodando nas duas zonas de isolamento, mesmo diante de falhas tanto das instâncias da aplicação *Web* quanto das zonas de isolamento. Note que as quatro instâncias são divididas igualmente entre as duas zonas de isolamento, desde que elas estejam operacionais. Se uma zona se tornar indisponível, o mecanismo de auto *scaling* incrementa automaticamente o número de instâncias para quatro na outra zona de isolamento, a fim de satisfazer a quantidade mínima de instâncias da aplicação *Web* adotadas neste estudo de caso. A instância do servidor de BD (ativo) é sincronizada com o BD *hot standby* que se encontra hospedado na outra zona de isolamento. Quando o servidor de BD ativo falhar, o BD *hot standby* assume as operações até que o BD ativo seja recuperado.

6.2.1 Modelos da SysML Anotados

Para o sistema de aplicação *Web*, representado na Figura 6.9, o projetista usa o SysML-IBD para descrever a configuração estática do sistema. A Figura 6.10 apresenta o SysML-IBD do sistema de aplicação *Web*. Cada bloco representa um elemento básico do sistema

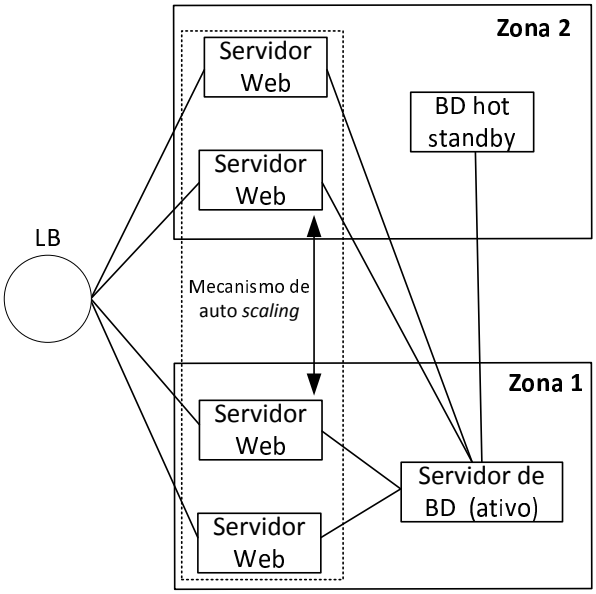


Figura 6.9: Sistema de Aplicação Web Hospedado nas Nuvens.

(ex.: balanceador de cargas, servidor *Web*, servidor de BD, zona etc.), onde os blocos na parte superior do SysML-IBD possuem alocações do tipo ≪hosted≫ aos blocos que representam as zonas de isolamento na parte inferior do diagrama. Esse tipo de relacionamento é utilizado para descrever as dependências de hospedagem entre os elementos do sistema. Isto é, se a Zona 1 se tornar indisponível, os servidores *Web* e o servidor de BD também se tornaram indisponíveis ao mesmo tempo. O relacionamento entre a instância do servidor de BD e o BD *hot standby* é representado através da alocação ≪standby≫. Adicionalmente, neste estudo de caso, os compartimentos dos blocos são usados para descrever: (i) as propriedades dos blocos (MTTF, MTTR e Max), (ii) as alocações dos blocos aos diagramas (*allocatedFrom*) e (iii) a redundância dos blocos. É importante ser destacado que o parâmetro (*Max*) define o número máximo de instâncias da aplicação *Web*. Para o sistema de aplicação *Web* adotado neste estudo de caso, o número máximo de instâncias consideradas é cinco, sendo requerido quatro para o funcionamento adequado do sistema como um todo. A alocação *allocatedFrom* define os relacionamentos entre os blocos e os diagramas da SysML (SysML-STM e SysML-AD). Os blocos *Servidor Web 1* e *Servidor Web 2* são detalhados em termos de estados pelo SysML-STM da Figura 6.11 (a), já o mecanismo de auto *scaling* que determina o número de instâncias da aplicação *Web* rodando nas zonas de isolamento é descrito na Figura 6.12.

O mecanismo de auto *scaling* permite que as instâncias da aplicação *Web* sejam automaticamente removidas ou adicionadas, conforme as definições estabelecidas pelos usuários. Esse mecanismo é extremamente importante para garantir alta disponibilidade dos sistemas em nuvem, visto que ele consiste em monitorar a infraestrutura de interesse, e em caso de falhas dos servidores ou das zonas de isolamento, ações são realizadas para manter a configuração mínima definida pelo o usuário. Neste estudo de caso, foi considerado o cenário em que a quantidade mínima de instâncias da aplicação *Web*, rodando nas

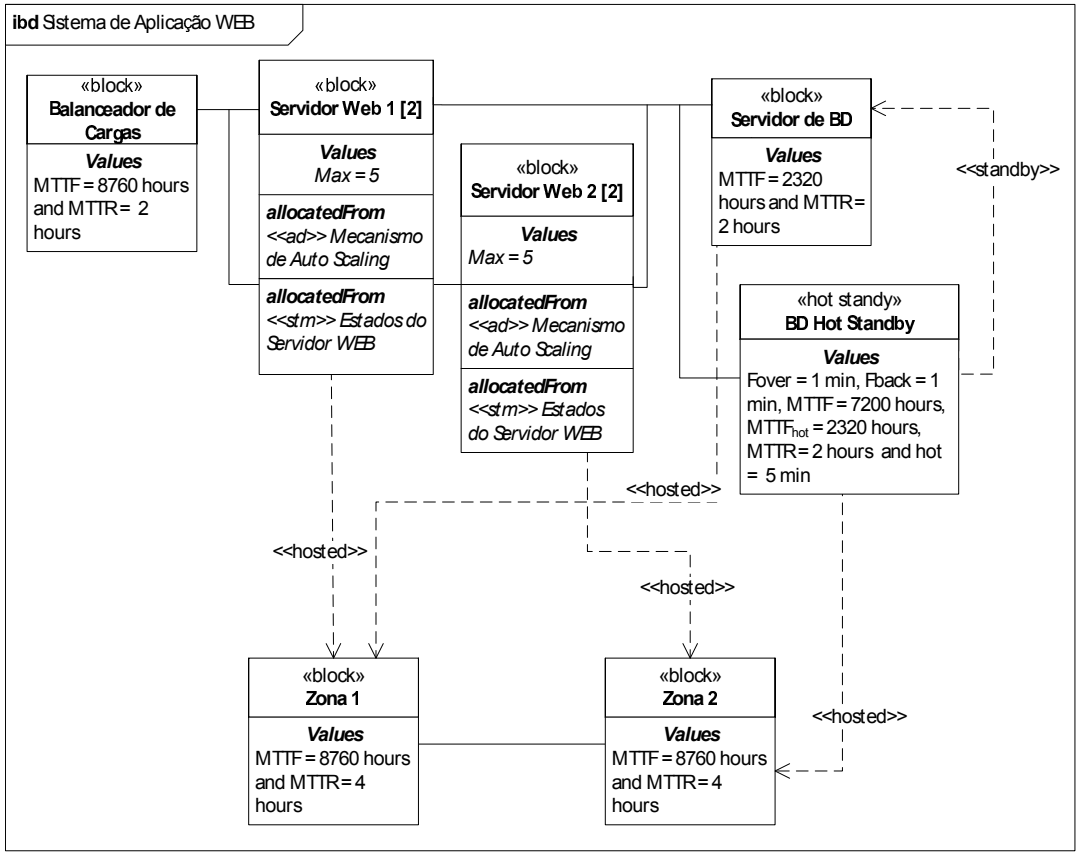


Figura 6.10: SysML-IBD para o Sistema de Aplicação *Web* Hospedado nas Nuvens.

duas zonas de isolamento, é quatro. A Figura 6.12 descreve o mecanismo de auto *scaling* adotado. Em primeiro lugar, o mecanismo de auto *scaling* é aplicado apenas às instâncias da aplicação *Web* de uma zona. Isto é, para cada zona de isolamento, é utilizado um mecanismo de auto *scaling* distinto. Esses mecanismos iniciam pela execução da atividade *Checar o status da outra zona* (ver Figura 6.12), que é realizada periodicamente a cada 5 minutos. Dependendo da disponibilidade da outra zona, o número de instâncias da aplicação *Web* é ajustado. Se a outra zona estiver operacional [*Up*], o mecanismo de auto *scaling* mantém o número de instâncias na sua própria zona para dois. Se o número de instâncias disponíveis é menor que dois, o mecanismo cria uma nova instância desde que se tenham instâncias disponíveis para serem usadas. Note que o número máximo de instâncias da aplicação *Web* é cinco (*Max=5*). Por outro lado, se o número de instâncias é maior que dois, o mecanismo para uma instância operacional. Caso a outra zona esteja indisponível [*Down*], o mecanismo de auto *scaling* tenta aumentar o número de instâncias para até quatro na sua própria zona. Como pode ser observado na Figura 6.12, as atividades *Checar Status do outra zona*, *Checar o número de instâncias disponíveis* (I_d) *e o número de instâncias usadas* (I_u), *Criar uma nova instância da aplicação Web* e *Parar uma nova instância da aplicação Web* possuem anotações de disponibilidade especificadas através de MARTE. Para essas atividades foram atribuídos tempos exponenciais de

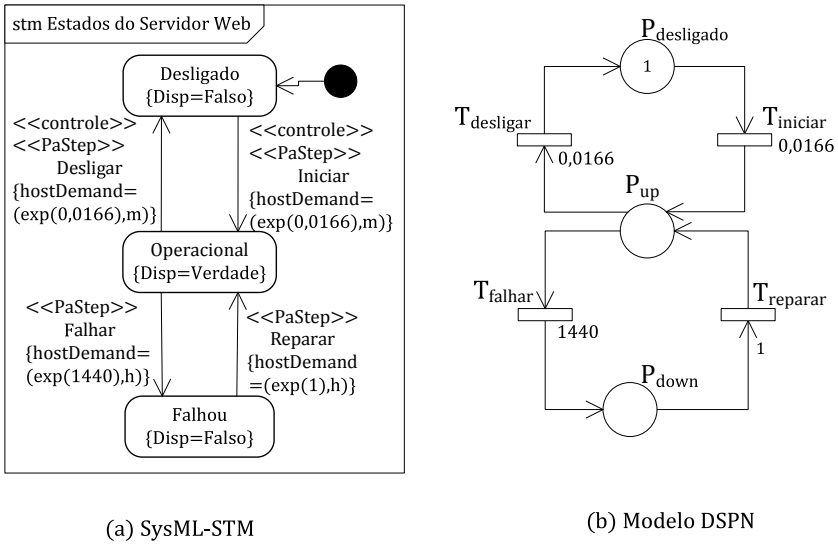


Figura 6.11: Modelos Usados para Representar as Instâncias da Aplicação *Web*.

1 segundo cada (0,00027 horas).

6.2.2 Mapeamento dos Diagramas da SysML Anotados em DSPNs

Esta seção detalha o processo de mapeamento dos diagramas da SysML, anotados de acordo com o *profile* MARTE, em uma DSPN. O processo consiste basicamente em três etapas: (i) mapeamento dos diagramas da SysML anotados, (ii) composição dos modelos, e (iii) sincronização dos modelos.

6.2.2.1 Mapeamento dos Diagramas da SysML Nesta etapa, primeiramente, os blocos do SysML-IBD são mapeados em componentes do modelo, com base nas regras de mapeamento apresentadas anteriormente. Cada bloco é mapeado no componente do modelo apresentado na Figura 4.3 (b). O componente do modelo ≪hot standby≫ é gerado para o *BD Hot Standby*, conforme o esterótipo ≪standby≫ (ver Seção 4.3.4 do Capítulo 4). Em seguida, os SysML-STMs usados para representar o comportamento de falha e reparo das instâncias da aplicação *Web* são mapeados no componente do modelo apresentado na Figura 6.11 (b). Por fim, o SysML-AD que representa o mecanismo de auto *scaling* é mapeado na *Rede de Atividades* apresentada na Figura 6.13. Note que para as aplicações *Web* das Zonas 1 e 2, duas *Redes de Atividades* são geradas.

6.2.2.2 Composição dos Modelos DSPN Uma vez gerados os componentes do modelo a partir do SysML-IBD e do SysML-STM, o *framework* compõe esses modelos, com base nas anotações de alocação. O modelo resultante dessa composição é denominado de *Rede do Sistema*. Inicialmente, os componentes do modelo dos blocos são substituídos pelos componentes do modelo gerados através do SysML-STM. Essa atividade é reali-

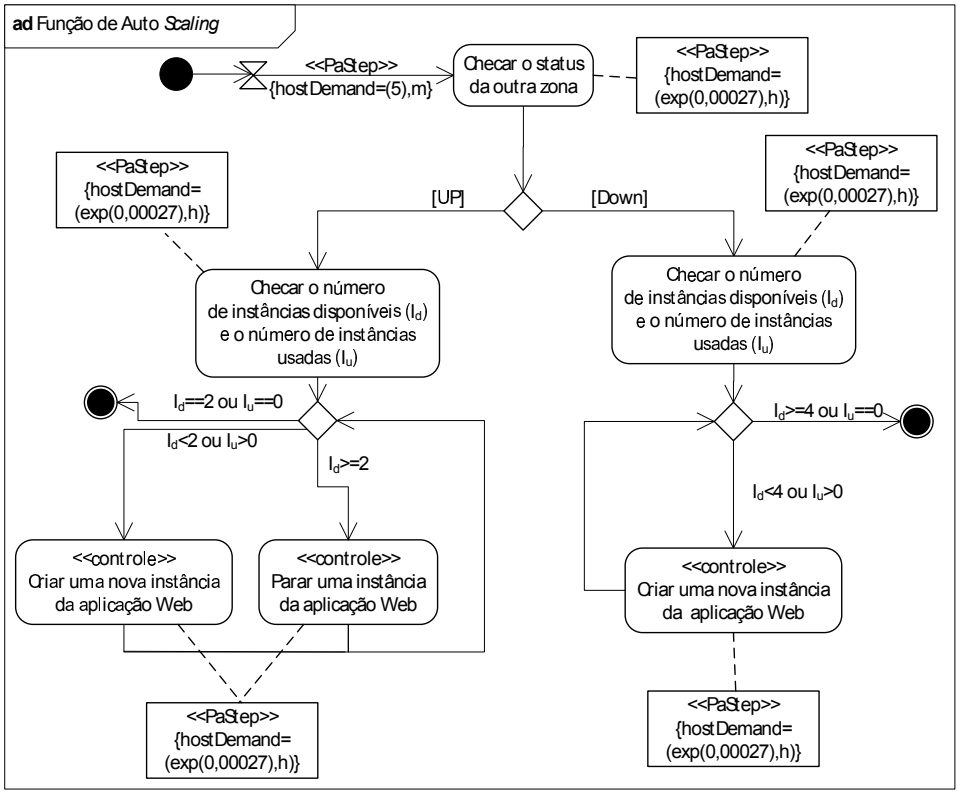


Figura 6.12: SysML-AD Usado para Representar o Mecanismo de Auto *Scaling*.

zada de acordo com definições estabelecidas para a alocação *alocatedFrom*. Após isso, as funções de guarda para o comportamento de *failover* do ≪standby≫ são geradas. Por fim, as transições imediatas e as funções de guarda para os componentes do modelo das instâncias da aplicação *Web* são geradas, de acordo com a alocação ≪hosted≫. As funções de guarda obtidas por esse processo são listadas na Tabela 6.7. Os modelos DSPN são sumarizados na Figura 6.14.

Tabela 6.7: Funções de Guarda para a *Rede do Sistema*.

Alocação	Transição	Definição
≪hosted≫	G_{DBupZ1}, $G_{WEB1upZ1}$	$\sharp P_{z1down}$=0
	G_{DHupZ2}, $G_{WEB2upZ2}$	$\sharp P_{z2down}$=0
	G_{DBdwZ1}, $G_{WEB1dwZ1}$	$\sharp P_{z1down}$>0
	G_{DHdwZ2}, $G_{WEB2dwZ2}$	$\sharp P_{z2down}$>0
≪standy≫	$G_{DHfover}$	$\sharp P_{DBdown}$>0
	G_{DHfbac}	$\sharp P_{DBup}$>0

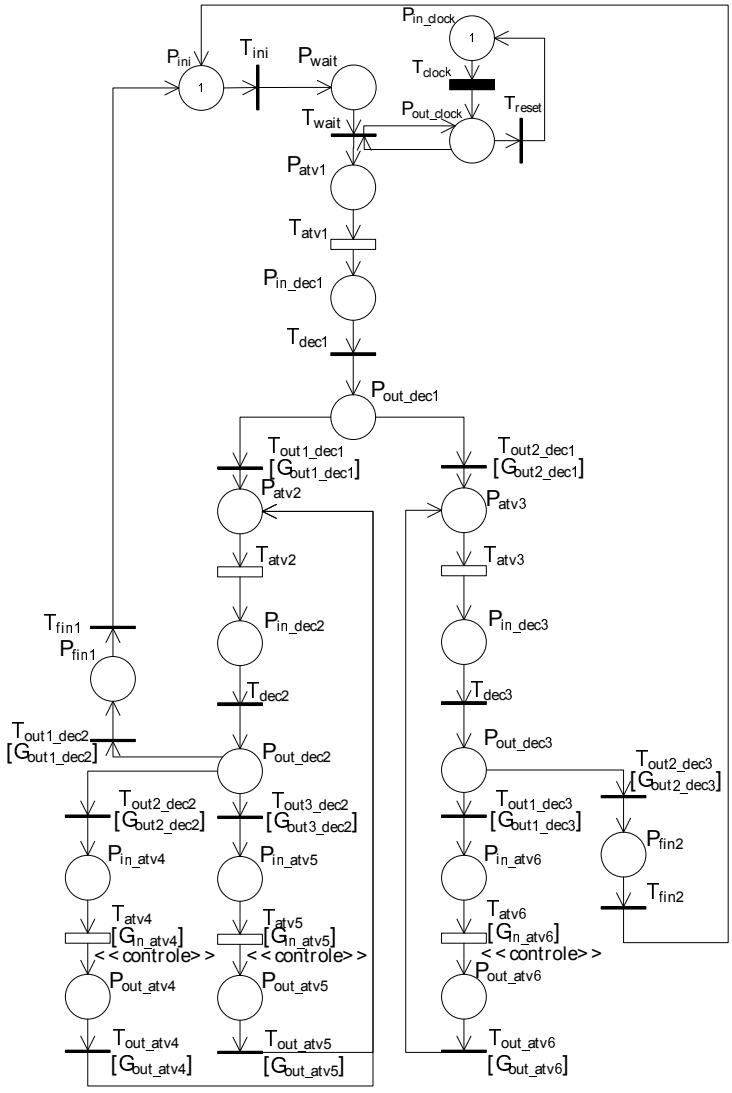


Figura 6.13: *Rede de Atividades* do Mecanismo de Auto *Scaling*.

6.2.2.3 Sincronização dos Modelos DSPN Na *Rede de Atividades*, as transições T_{atv4}, T_{atv5} e T_{atv6} são sincronizadas com as transições $T_{WEB1ini}$ e $T_{WEB1des}$ da *Rede do Sistema*, visto que as atividades *Criar uma nova instância da aplicação Web* e *Parar uma instância da aplicação Web* podem resultar no disparo das transições *Desligar* e *Iniciar* das instâncias da aplicação *Web*. A partir do nome da atividade no SysML-AD, os projetistas podem encontrar facilmente as transições correspondentes a essas atividades nos SysML-STMs, pois todas elas são identificadas através do estereótipo ≪controle≫. Como explicado na Figura 4.13, as transições $T_{WEB1ini}$ e $T_{WEB1des}$ da *Rede do Sistema* são expandidas com transições imediatas, lugares e funções de guarda. A *Rede do Sistema* atualizada para as instâncias da aplicação *Web* da Zona 1 é apresentada na Figura 6.15. Já que o número máximo de instâncias é 5, *tokens* adicionais são depositados no lugar $P_{WEB1des}$.

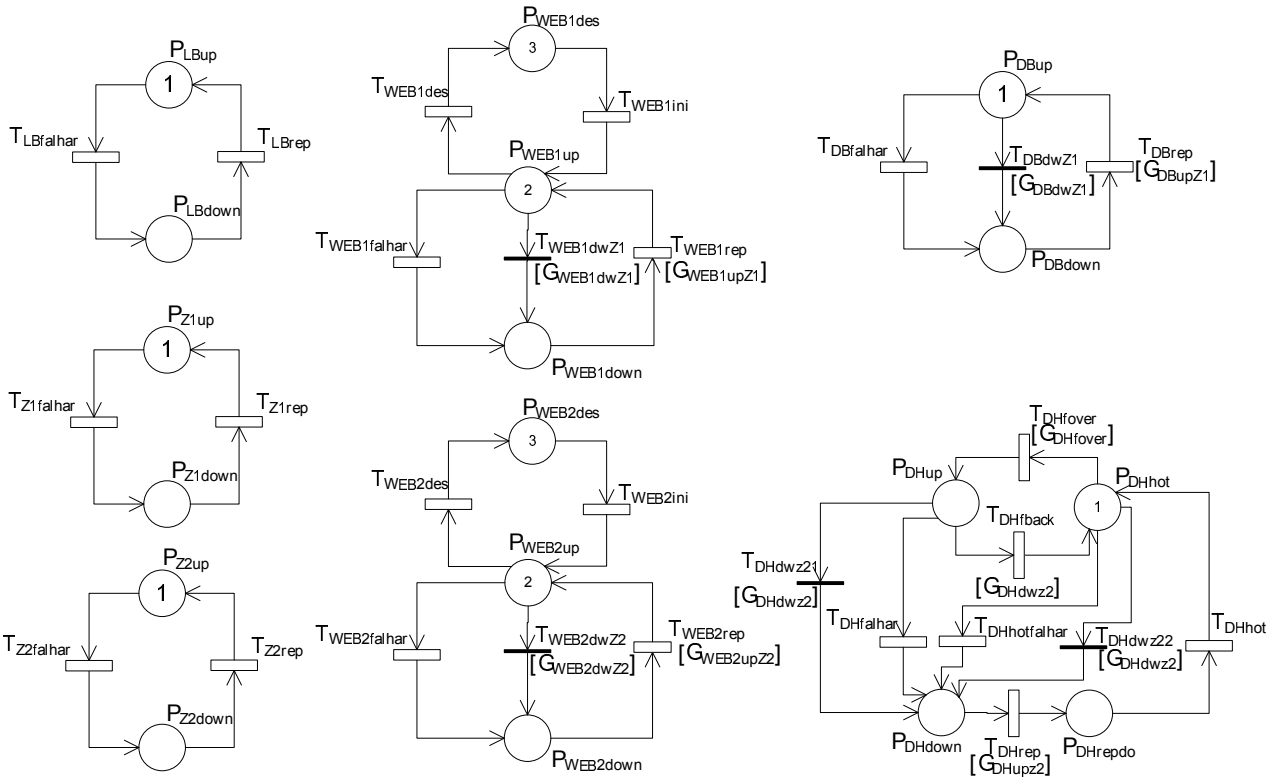


Figura 6.14: *Redes do Sistema* Obtidas pelo Processo de Composição.

As funções de guarda para os nós de decisão também precisam ser definidas durante a sincronização dos modelos DSPN, conforme descrito na Seção 4.3.5 do Capítulo 4. Uma vez que os fluxos de saída do lugar P_{out_dec1} dependem da disponibilidade da outra zona (Zona 2), o projetista define as funções de guarda G_{out1_dec1} e G_{out2_dec1} usando a marcação de P_{z2up} (ver Figura 6.14). Já os fluxos de saída dos lugares P_{out_dec2} e P_{out_dec3} dependem do número de instâncias da aplicação *Web* disponíveis e operacionais na própria zona (Zona 1). Assim, as funções de guarda G_{out1_dec2}, G_{out2_dec2}, G_{out3_dec2}, G_{out1_dec3} e G_{out2_dec3} são definidas baseadas nas marcações dos lugares P_{WEB1up} e $P_{WEB1des}$ (ver Figura 6.14). A Tabela 6.8 lista as funções de guarda obtidas através da sincronização dos modelos da Zona 1. Semelhantemente, o SysML-AD da Zona 2 é sincronizado com a *Rede do Sistema* do Servidor *Web* 2. Note que esse processo não foi apresentado nesta seção, pois o mesmo é semelhante ao apresentado anteriormente.

6.2.3 Análise Numérica

Esta seção descreve os resultados obtidos a partir dos modelos gerados pelo processo de mapeamento. As métricas adotadas são listadas na Tabela 6.9. Os valores *default*, usados na avaliação, são sumarizados na Tabela 6.10. Os tempos de falha e reparo das zonas são definidos de acordo com informações fornecidas pela *Amazon EC2* [Ama13]. Para os outros parâmetros, foram usados tempos baseados em trabalhos anteriormente

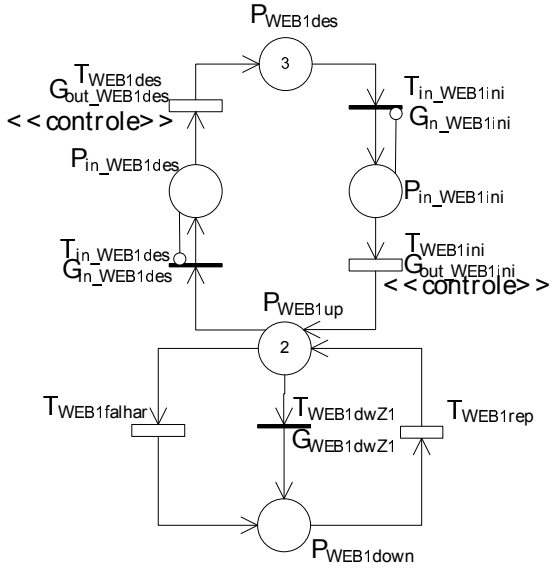


Figura 6.15: *Rede do Sistema* Atualizada.

Tabela 6.8: Funções de Guarda Geradas pelo Processo de Sincronização dos Modelos DSPN.

Guarda	Função	
Criar Inst.	$G_{in\_WEB1ini}$	($\sharp P_{in\_atv4}$=1) OR ($\sharp P_{in\_atv6}$=1)
	G_{in_atv4}, G_{in_atv6}	$\sharp P_{in\_WEB1ini}$=1
	$G_{out\_WEB1ini}$	($\sharp P_{out\_atv4}$=1) OR ($\sharp P_{out\_atv6}$=1)
	G_{out_atv4}, G_{out_atv6}	$\sharp P_{in\_WEB1ini}$=0
Parar Inst.	$G_{in\_WEB1des}$	$\sharp P_{in\_atv5}$=1
	G_{in_atv5}	$\sharp P_{in\_WEB1des}$=1
	$G_{out\_WEB1des}$	$\sharp P_{out\_atv5}$=1
	G_{out_atv5}	$\sharp P_{in\_WEB1des}$=0
Decisão	G_{out1_dec1}	$\sharp P_{z2up}$>0
	G_{out2_dec1}	$\sharp P_{z2up}$=0)
	G_{out1_dec2}	($\sharp P_{WEB1up}$=2) OR ($\sharp P_{WEB1des}$=0)
	G_{out2_dec2}	($\sharp P_{WEB1up}$<2) AND ($\sharp P_{WEB1des}$>0)
	G_{out3_dec2}	$\sharp P_{WEB1up}$>2
	G_{out1_dec3}	($\sharp P_{WEB1up}$<4) AND ($\sharp P_{WEB1des}$>0)
	G_{out2_dec3}	($\sharp P_{WEB1up}$>=4) OR ($\sharp P_{WE1Bdes}$=0)

publicados [KMT09, HGG+10, AMKT11]. Neste estudo de caso, todas as transições são exponencialmente distribuídas com exceção da transição determinística T_{clock}. A disponibilidade para cada componente do sistema (Balanceador de Carga, Servidores Web e Servidor de Banco de Dados), bem como a disponibilidade de todo o sistema é apresentada na Tabela 6.11. Note que a disponibilidade do sistema (A_{sys}) não é igual ao produto da disponibilidade de cada componente porque eles não são independentes.

Tabela 6.9: Métricas para Avaliação Estacionária.

Métrica	Função
Balanceador de Carga (R_{LB})	P{$\sharp P_{LBup}$=1}
Servidor Web (R_{WEB})	P{($\sharp P_{WEB1up}$ + $\sharp P_{WEB2up}$>=4)}
Banco de Dados (R_{DB})	P{($\sharp P_{DBup}$ + $\sharp P_{DHup}$>=1)}
Sistema (A_{sys})	P{($\sharp P_{LBup}$) AND ($\sharp P_{WEB1up}$ + $\sharp P_{WEB2up}$>=4) AND ($\sharp P_{DBup}$ + $\sharp P_{DHup}$>=1)}

Tabela 6.10: Descrição dos Parâmetros de Entrada e seus Valores *Default*.

Parâmetros	Transição	Valor [horas]
Tempo Até a Falha do LB	$T_{LBfalhar}$	8760
Tempo de Reparo do LB	T_{LBrep}	2
Tempo Até a Falha do Servidor Web	$T_{WEB1falhar}$, $T_{WEB2falhar}$	1440
Tempo de Reparo do Servidor	$T_{WEB1rep}$,$T_{WEB2rep}$	1
Tempo de Iniciar e Desligar o Servidor	$T_{WEB1ini}$,$T_{WEB2ini}$	0,0166
	$T_{WEB1des}$,$T_{WEB2des}$	
Tempo Até a Falha do BD	$T_{DBfalhar}$	2320
Tempo de Reparo do BD	T_{DBrep}	2
Tempo Até a Falha do BD *Hot Standby*	$T_{DHfalhar}$	7200
Tempo de Reparo do BD *Hot Standby*	T_{DHrep}	0,0833
Tempo de *Failover* e *Failback*	$T_{DHfover}$, T_{DHfbac}	0,0166
Tempo Até a Falha da Zona	$T_{Z1falhar}$, $T_{Z2falhar}$	8760
Tempo de Reparo do Zona	T_{Z1rep},T_{Z2rep}	4
Intervalo de Checagem da Condição	$T_{clockIn}$	0,0833
Tempo das Atividades	T_{actX}	0,00027

Tabela 6.11: Resultado da Análise Numérica.

	R_{LB}	R_{WEB}	R_{DB}	A_{sys}
Duas Zonas	0,999772	0,999804	0,999993	0,999571
Uma Zona	0,999772	0,999025	0,999420	0,998778

Com o intuito de estudar os efeitos das interrupções em múltiplas zonas de isolamento, foi feito um outro modelo no qual todas as instâncias da aplicação *Web* e do banco de dados pertencem à mesma zona (análise do tipo "e se"). O mecanismo de auto *scaling* também foi simplificado para considerar apenas uma zona de isolamento. O modelo de disponibilidade para os diagramas da SysML modificados foram obtido pelo mesmo processo de mapeamento apresentado anteriormente. Uma vez obtido os modelos DSPN,

realizamos análise de sensibilidade do mesmo, variando o intervalo de monitoramento da função de auto *scaling* (T_{clock}). Em seguida, comparamos os resultados da análise de sensibilidade considerando uma e duas zonas de isolamento. Essa comparação é apresentada na Figura 6.16. A partir dos resultados, foi possível observar a vantagem de usar duas zonas de isolamento numa infraestrutura nas nuvens. Também foi verificado que a efetividade do mecanismo de *scaling* depende do intervalo de monitoramento do mecanismo de tratamento de interrupções. Quanto menor o intervalo, maior é a disponibilidade do sistema. No entanto, tal monitoramento pode resultar num fenômeno conhecido como efeito de intrusão, que resulta do fato da observação de um sistema poder afetar o comportamento do mesmo. Desta forma, é importante que o mecanismo de monitoramento cause a menor intrusão possível no sistema observado.

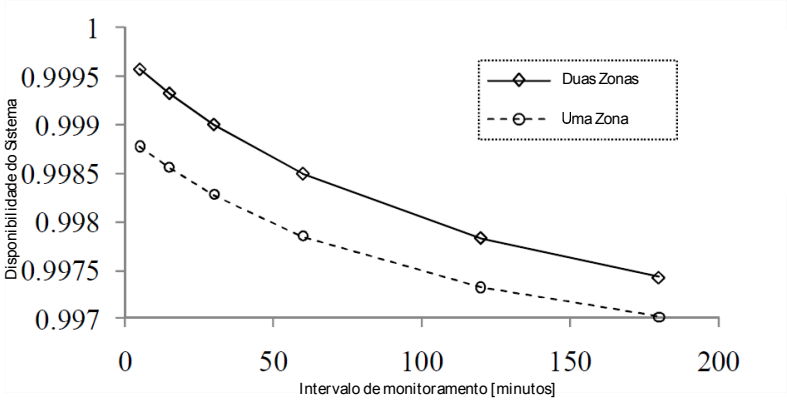


Figura 6.16: Análise de Sensibilidade.

6.3 SISTEMA DE RECUPERAÇÃO DE DESASTRES COMO UM SERVIÇO NAS NUVENS

Devido aos eventos catastróficos que aconteceram nos últimos anos (ex.: ataque terrorista do 11 de setembro, tsunami no Japão, furação em Nova York etc.) ou as falhas nos serviços de grandes corporações das comunicações, tais com Gmail, Dropbox, Facebook, entre outros, as empresas perceberam a importância de estudar os mecanismos de tratamento de interrupções, já que as interrupções, sejam elas brandas ou severas, podem acontecer em qualquer lugar, a qualquer momento, com pouco ou nenhum aviso. Nesse sentido, este estudo de caso apresenta uma sistema de recuperação de desasastes como um serviço nas nuvens, visto que as empresas com dados armazenados em nuvem têm a possibilidade de recuperá-los muito mais rapidamente do que as organizações com tecnologia baseada em *backup* de dados em dispositivos de fita ou discos removíveis. Dentre os benefícios oferecidos pela recuperação de desastres em nuvem, é possível destacar: (i) redução dos custos (a necessidade de compra de novos equipamentos e custos de gestão são eliminados), (ii) automação (*backup* e serviço de recuperação podem ser totalmente automatizado e ocorre de forma segura), (iii) recuperação rápida de dados (a recuperação

de desastres baseada em nuvem pode demorar apenas algumas poucas horas), e (iv) replicação e espelhamento *off-site* (os dados são distribuídos geograficamente).

A Figura 6.17 descreve os componentes da infraestrutura de recuperação de desastres adotada como estudo de caso. Essa infraestrutura foi montada no nosso laboratório (Centro de Informática, UFPE) e experimentos foram realizados para analisar os modelos obtidos pelo processo de mapeamento. A infraestrutura de recuperação de desastres possui um *data center* primário, no qual são consideradas quatro máquinas virtuais. Essas VMs consistem de um balanceador de cargas, dois servidores de aplicação (Apache Tomcat versão 7.0) e um servidor de banco de dados (MySQL versão 5.5). A aplicação *Web* fornecida pelos servidores de aplicação é um *mashup* (composição de serviços web) para recomendação de eventos musicais. O balanceador de carga distribui as requisições dos usuários, igualmente, entre os servidores de aplicação para evitar que uma única VM fique sobrecarregada enquanto as outras estão ociosas.

A nuvem de recuperação de desastres, que é um espelho do *data center* primário, possui réplicas do balanceador de cargas, dos servidores Web e do servidor de banco de dados. Os Volumes de Armazenamento (VA) usados pelos servidores de banco de dados primário e secundários são mantidos sincronizados, usando o *software* DRBD (*Distributed Replicated Block Device*) [Rei02]. A aplicação *mashup* é orientada a leitura, então a frequência de alterações no servidor de banco de dados é muito baixa, permitindo a sincronização completa dos volumes primário e secundário a baixo custo de comunicação. O sistema operacional usado para todas as máquinas virtuais, neste estudo de caso, é o *Ubuntu Server Linux* 11.10. Um servidor IBM x3200 M3, com processador quad-core Intel Xeon 3400 3.0 GHz e 4GB de RAM foi usado para hospedar as máquinas virtuais do *data center*. Já a nuvem de recuperação de desastres era constituída de 5 computadores com processadores Intel Core 2 Quad (2.66 GHz) e 4 GB RAM, executando o sistema operacional *Ubuntu Server Linux* 11.04, e o *framework* de computação em nuvem *Eucalyptus* versão 2.0.2.

O balanceador de cargas, os servidores Web e o servidor de banco de dados estão todos ativos na nuvem de recuperação de desastres. No entanto, eles não processam nenhuma requisição dos usuários enquanto o *data center* primário estiver operacional. O monitor de desastres, localizado fora de ambos os sites, é responsável por detectar a ocorrência de uma falha no *data center* primário, devido a um desastre. Em caso de um desastre, o monitor de desastres notifica o LB da nuvem para assumir o endereço IP (*Internet Protocol*) virtual do serviço de modo que as requisições dos usuários sejam encaminhadas para a nuvem. Esse mecanismo de *failover* é habilitado pelo *software* de gerenciamento de recursos de *cluster* chamado *Pacemaker* [PPP11], em conjunto com a ferramenta de gerenciamento de *clusters Heartbeat*. Ao utilizar este mecanismo, a transição do *data center* primário para a nuvem de recuperação de desastres não é percebida pelos os usuários da aplicação *mashup*, com exceção daqueles que enviaram requisições no período de tempo decorrido entre a falha do *data center* principal e o momento em que o *load balancer* da nuvem recebe a notificação do monitor de desastres, assumindo o endereço IP virtual do serviço. Quando ocorre o reparo do *data center* principal após o desastre,

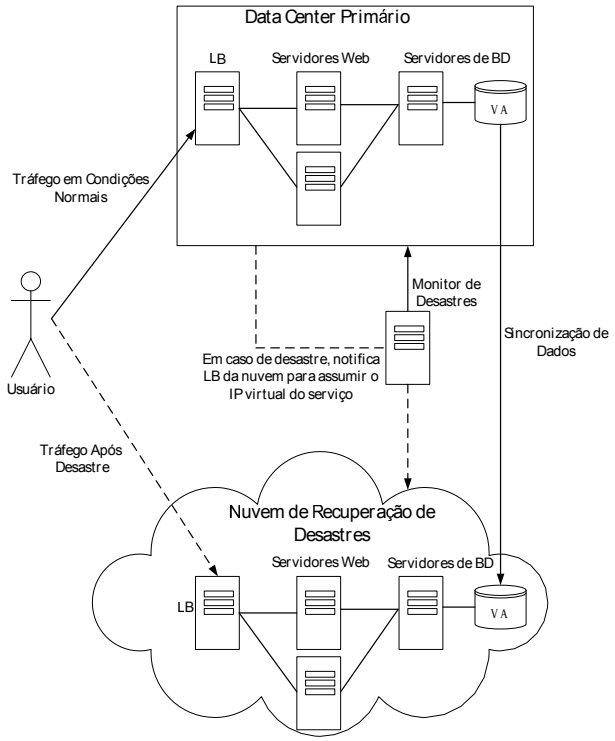


Figura 6.17: Sistema de Recuperação de Desastres.

o monitor de desastre notifica o LB da nuvem para que ele libere o endereço IP virtual do serviço, que será reassumido pelo LB do *data center*.

6.3.1 Modelos da SysML Anotados

O SysML-IBD do sistema de recuperação de desastres é apresentado na Figura 6.18. Cada bloco representa um elemento básico do sistema, tais como: balanceador de cargas, servidor *Web*, *data center*, nuvem, entre outros. O número "2"no cabeçalho dos blocos *Servidor Web* e *Servidor Web Hot Standby* representa o nível de redundância dos elementos do sistema. Os compartimentos dos blocos, por sua vez, são usados para representar as propriedades dos blocos (MTTF e MTTR), bem como as alocações deles a outros diagramas (*allocatedFrom*). Além disso, a alocação do tipo ≪hosted≫ é usada para representar as dependências de hospedagem entre os elementos do sistema. Por exemplo, se o *data center* se tornar indisponível, o balanceador de cargas, os servidores *Web* e o servidor de BD se tornam indisponíveis ao mesmo tempo.

O SysML-STM do bloco *Data Center* é descrito na Figura 6.19. O *data center* inicia no estado *Up* (operacional). Uma vez nesse estado, o servidor pode falhar. A falha pode ser em decorrência de um desastre ou em decorrência de uma falha transiente[1]. Após

[1]As falhas transientes são aquelas de duração limitada, causadas por mal funcionamento temporário ou por alguma interferência externa.

isso, o *data center* é reparado. Note que o tempo de reparo de um desastre é maior que o tempo de reparo de uma falha transiente. Os blocos *Servidor Web Hot Standby* e o *Servidor de BD Hot Standby* são detalhados em termos de estados pelos SysML-STMs da Figura 6.20. O servidor *hot standby* inicia no estado *Hot Standby*, que representa um estado de prontidão do servidor. Uma vez nesse estado, o servidor *hot standby* pode falhar ou assumir a operação do servidor ativo. Se o mesmo falhar, o servidor *hot standby* assume o estado *Falhou*. Em seguida, o servidor *hot standby* é reparado e retorna para o estado *Operacional*. Se o servidor *hot standby* assumir a operação do servidor ativo (operação de *failover*), o mesmo pode falhar ou retornar para o estado *Hot Standby*, após o servidor ativo ser reparado. Note que as transições *Failover* e *Failback* são estereotipadas com ≪controle≫, indicando que essas transições podem ser afetados pela execução das atividades do SysML-AD. Ademais, os tempos associados à transição de estados dos diagramas foram especificados através do *profile* MARTE e são detalhados abaixo.

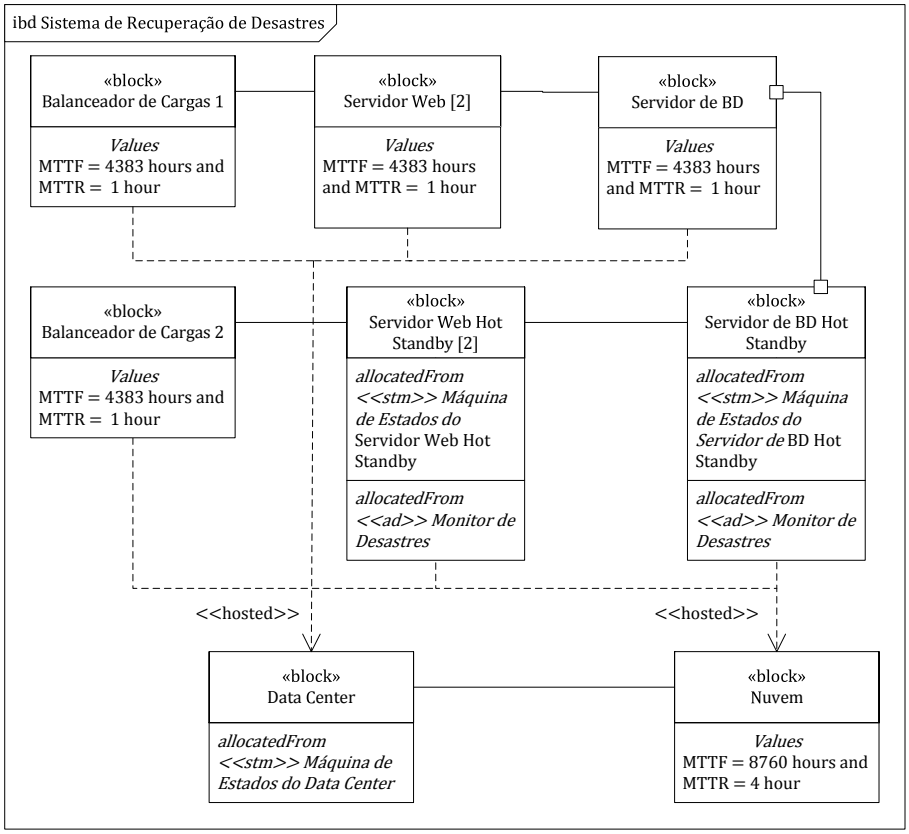


Figura 6.18: SysML-IBD do Sistema de Recuperação de Desastres.

O SysML-AD usado para representar o mecanismo de monitoramento de desastres é detalhado na Figura 6.21. A cada cinco minutos o status do *data center* é analisado. Se um desastre não tiver sido detectado [Não], então o mecanismo espera 5 minutos para analisar novamente. Caso contrário [Sim], o status da nuvem é verificado e caso ela esteja operacional, é executado o *failover* de todos os servidores *hot standby*. Isto é, quando ocorre um desastre, a nuvem de recuperação de desastres assume a execução

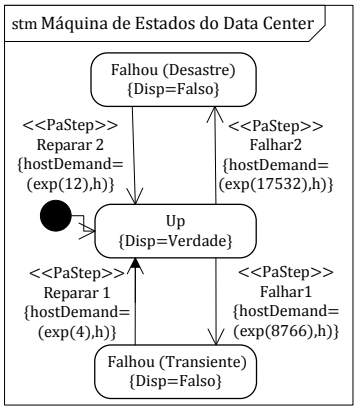
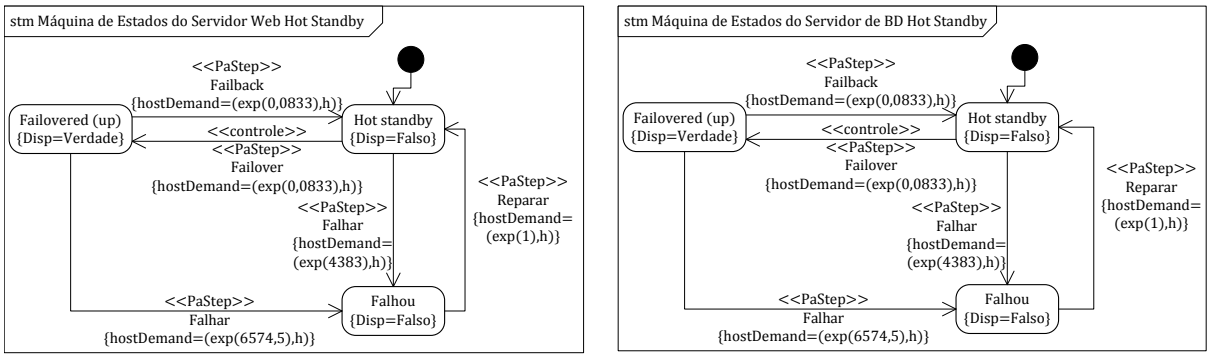


Figura 6.19: SysML-STM para o *Data Center*.



(a) Máquina de Estados do Servidor *Web Hot Standby*

(b) Máquina de Estados do Servidor de BD *Hot Standby*

Figura 6.20: SysML-STMs para os Servidores Web *Hot Standby* e o Servidor de BD *Hot Standby*.

das atividades do *data center* primário. Note que é possível que a nuvem também esteja indisponível; nesse caso, é enviada uma mensagem de alerta ao administrador que é responsável por intervenções manuais. As atividades *Checar Status do Data Center*, *Checar Status do Nuvem*, *Enviar Mensagem de Alerta* e *Executar Failover de Todos os Servidores* possuem anotações de disponibilidade especificadas através de MARTE. Para essas atividades foram atribuídos tempos exponenciais de 1 segundo cada (0,00027 horas).

6.3.2 Mapeamento dos Diagramas da SysML Anotados em DSPN

Uma vez modelado o sistema de recuperação de desastres usando os diagramas da SysML e as anotações de MARTE, então os mesmos são mapeados em modelos DSPN. Conforme apresentado anteriormente, o método proposto consiste, primeiramente, no mapeamento dos diagramas da SysML, anotados através de MARTE, em modelos DSPN. Em seguida, as DSPN obtidas são compostas, de acordo com as anotações de alocações *alocatedFrom* e ≪hosted≫. Por fim, os modelos são sincronizados para formar uma DSPN completa representando todo o sistema modelado. A Tabela 6.12 lista as funções de guarda geradas

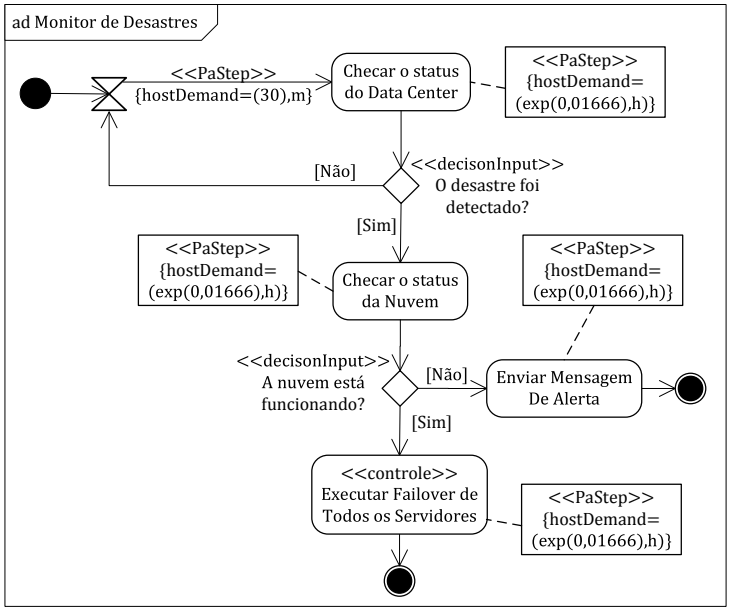


Figura 6.21: Mecanismo de Monitoramento de Desastres.

pelo processo de mapeamento. A Figura 6.22 apresenta os modelos DSPN resultantes de todo o processo de mapeamento.

6.3.3 Análise Numérica

Esta seção descreve os resultados preliminares da análise numérica do sistema de recuperação de desastres. Os valores *default* usados na avaliação são sumarizados na Tabela 6.13. Esses valores foram baseados parâmetros descritos em [KMT09, HGG+10, MAKT11, DMAM12]. Neste estudo de caso, todas as transições são exponencialmente distribuídas com exerção da transição determinística T_{clock}. A disponibilidade e o *downtime* do sistema com e sem recuperação de desastres são apresentados na Tabela 6.14. A métrica utilizada para calcular a disponibilidade do sistema (A_{sys}) é apresentada na Tabela 6.15. Como pode ser observada, a disponibilidade do sistema com recuperação de desastres é bem maior que o sistema sem recuperação de desastres. Essa disparidade também pode ser notada comparando o *downtime* anual de ambos os cenários. O *downtime* do sistema diminuiu mais de 50% devido a introdução da infraestrutura de recuperação de desastres. Isso deve-se ao fato de que a nuvem de recuperação de desastres é um elemento de redundância do *data center* primário. Assim, em caso de desastres, a nuvem assume a execução da aplicação *mashup*, resultando em alta disponibilidade e continuidade dos serviços oferecidos pela a aplicação.

O custo do *downtime* para uma empresa de TI de pequeno para médio porte também foi calculado, baseado em uma calculadora de custo de *downtime* disponível em [Rac]. O custo obtido foi de 913 dólares por hora. Os parâmetros utilizados para esse cálculo são: $1.000.000 de receita anual, 100% da receita de atividades online e 3 horas de receita

Tabela 6.12: Funções de Guarda Geradas pelo processo de Sincronização dos modelos DSPN.

Guarda	Função	
Composição	$G_{WEB1dwDC}$	($\sharp P_{DCdis}$=1) OR ($\sharp P_{DCdown}$=1)
	$G_{DB1dwDC}$	($\sharp P_{DCdis}$=1) OR ($\sharp P_{DCdown}$=1)
	$G_{LB1dwDC}$	($\sharp P_{DCdis}$=1) OR ($\sharp P_{DCdown}$=1)
	$G_{WEB1rep}$, G_{DB1rep}	$\sharp P_{DCup}$=1
	G_{LB1rep}	$\sharp P_{DCup}$=1
	$G_{DB2dwNV}$	$\sharp P_{NVdown}$=1
	$G_{WEB2dwNV}$	$\sharp P_{NVdown}$=1
	$G_{LB2dwNV}$	$\sharp P_{NVdown}$=1
	$G_{WEB2rep}$, G_{DB2rep}	$\sharp P_{NVup}$=1
	G_{LB2rep}	$\sharp P_{NVup}$=1
Sincronização	$G_{in\_DB2fover}$	$\sharp P_{in\_atv3}$=1
	$G_{in\_WEB2fover}$	$\sharp P_{in\_atv3}$=1
	G_{in_atv3}	($\sharp P_{WEB2fover}$>=1) OR ($\sharp P_{DB2fover}$=1)
	$G_{out\_WEB2fover}$	$\sharp P_{out\_atv3}$=1)
	$G_{out\_DB2fover}$	$\sharp P_{out\_atv3}$=1
	G_{out_atv3}	($\sharp P_{in\_WEB2fover}$=0) AND ($\sharp P_{in\_DB2fover}$=0)
Decisão	G_{out1_dec1}	se ($\sharp$('P_{DCup}=1
	G_{out2_dec1}	$\sharp P_{DCdis}$=1
	G_{out1_dec2}	$\sharp P_{NVup}$=1
	G_{out2_dec2}	$\sharp P_{NVdown}$=1
Failback	$G_{DB2fback}$	($\sharp P_{DCup}$=1) AND ($\sharp P_{DB1up}$=1) AND ($\sharp P_{WEB1up}$>=1)
	$G_{WEB2fback}$	($\sharp P_{DCup}$=1) AND ($\sharp P_{DB1up}$=1) AND ($\sharp P_{WEB1up}$>=1)

muito elevada. Assim, considerando o valor do custo obtido, foi calculado o custo do *downtime* por ano das infraestruturas estudadas (ver Tabela 6.14). Como já era de se esperar, o custo do *downtime* do sistema sem um solução de recuperação de desastre é bem maior que o custo do *downtime* do sistema com recuperação de desastre.

Quando o sistema falha devido a um desastre, todas as transações dos usuários são perdidas enquanto o sistema não é restabelecido em outra localidade/site. Dessa forma, também é importante considerar as transações perdidas durante esse processo. A taxa de dados perdidos refere-se ao número de transações perdidas por um período de interesse. Neste estudo de caso, foi considerado que as transações dos usuários seguem um processo *Poisson* com taxa λ_a. A taxa de dados perdidos é dada pela Equação 6.1. O parâmetro "T"representa o período de interesse. A Tabela 6.14 apresenta a taxa de dados perdidos (trans. por hora) para os sistemas com e sem recuperação de desastres. Esses resultados demonstram que o mecanismo de recuperação de desastres reduz pela metade a quantidade de transações perdidas pelos usuários. Note que interrupções no fornecimento de

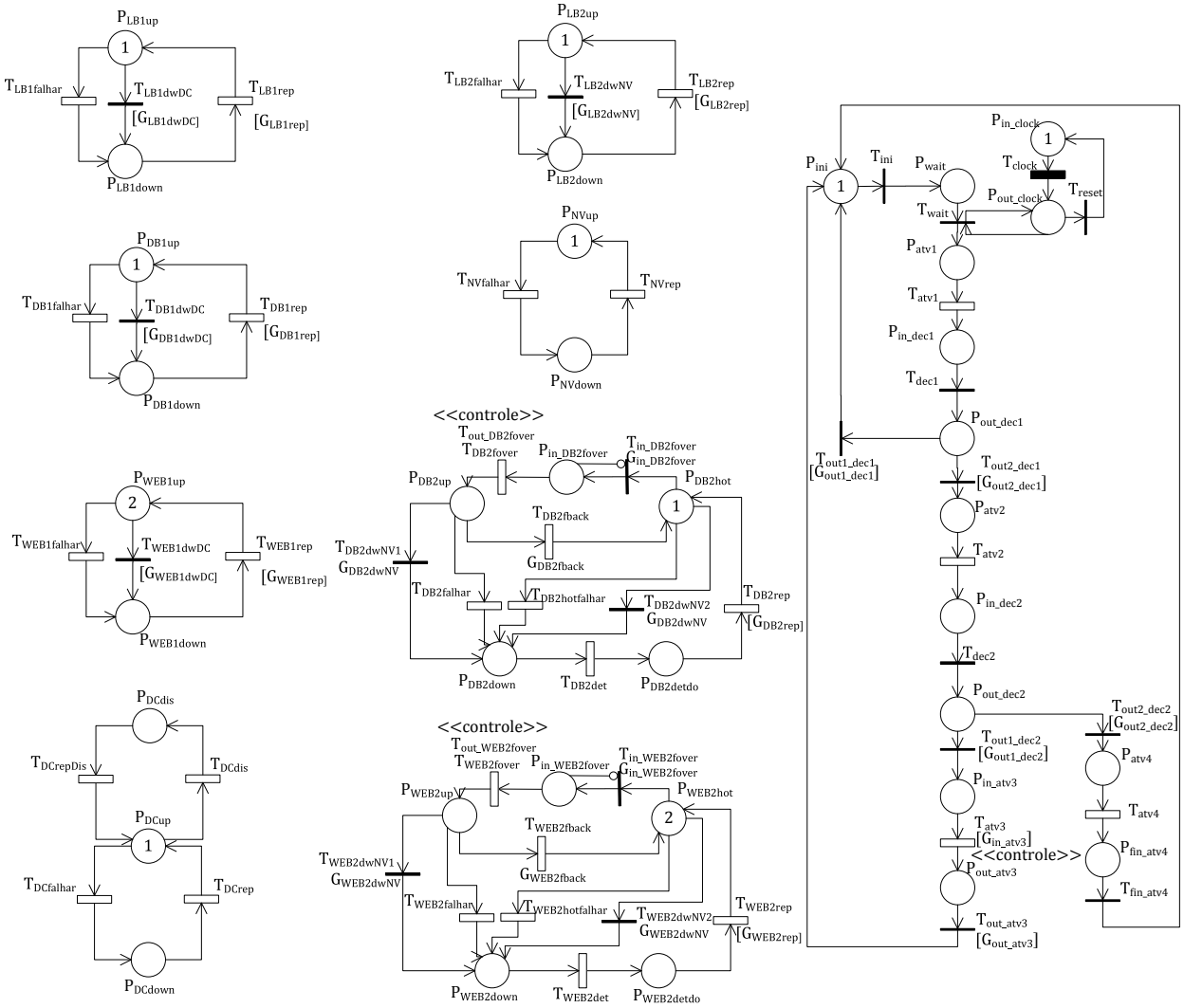


Figura 6.22: DSPN Resultantes do Processo de Mapeamento.

serviços prestados podem levar à má reputação da empresa e, consequentemente, diminuir a credibilidade dela no mercado.

$$\gamma = (1 - A_{sys}) * \lambda_a * T \tag{6.1}$$

A avaliação dos modelo DSPN do sistema de recuperação de desastres permite estudar o impacto de alguns fatores na disponibilidade do sistema. Um fator que é esperado ter grande impacto é o *delay* para a detecção do desastre, representado no modelo pelo intervalo de monitoramento. A Figura 6.23 apresenta a análise de sensibilidade da disponibilidade do sistema, em que o intervalo de monitoramento do sistema de recuperação de desastres foi variado (T_{clock}). Como pode ser observado a partir da gráfico, se o intervalo de monitoramento é próximo de zero, o sistema é checado constantemente evitando o *downtime* e, assim, resulta numa maior disponibilidade do sistema. Por outro lado, à

Tabela 6.13: Descrição dos Parâmetros de Entrada e seus Valores *Default*.

Parâmetros	Transição	Valor [horas]
Tempo Até a Falha do LB	$T_{LB1falhar}$, $T_{LB2falhar}$	4383
Tempo de Reparo do LB	T_{LBrep1}, T_{LBrep2}	1
Tempo Até a Falha do Servidor *Web*	$T_{WEB1falhar}$, $T_{WEB2falhar}$	4383
Tempo de Reparo do Servidor *Web*	$T_{WEB1rep}$, $T_{WEB2rep}$	1
Tempo Até a Falha do BD	$T_{DBfalhar}$, $T_{DB2falhar}$	4383
Tempo de Reparo do BD	T_{DB1rep}	1
Tempo de *Failover* e *Failback*	$T_{WEB2fover}$, $T_{WEB2fback}$, $T_{DB2fover}$, $T_{DB2fback}$	0,08333
Tempo Até a Falha do DC (Desastre)	T_{DCdis}	17532
Tempo Até a Falha do DC (Transiente)	$T_{DCfalhar}$	8766
Tempo de Reparo do DC (Desastre)	T_{DCdis}	12
Tempo de Reparo do DC (transiente)	T_{DCrep}	4
Tempo Até a Falha da Nuvem	$T_{NVfalhar}$	8766
Tempo de Reparo da Nuvem	T_{NVrep}	4
Intervalo de Checagem do Desastre	$T_{clockIn}$	0.5
Tempo das Atividades	T_{actX}	0,016667
Taxa de Transações de Entrada	λ_a	0,001

Tabela 6.14: Resultados Obtidos.

	Disponibilidade	Downtime	Custo	Taxa de dados perdidos
Sistema c/ DR	0,9983669	14,30	\$13.061,34	21,16
Sistema s/ DR	0,9963723	31,78	\$29.013,91	47,01

Tabela 6.15: Métricas para Avaliação Estacionária.

Métrica	Função
A_{sys}	P{((♯P_{LB1up}=1) AND (♯P_{WEB1up}>=2) AND (♯P_{DB1up}=1)) OR ((♯P_{LB2up}=1) AND (♯P_{WEB2up}>=2) AND (♯P_{DB2up}=1))}

medida que intervalo de monitoramento aumente, a disponibilidade do sistema diminui, visto que o desastre pode levar mais tempo para ser detectado. Portanto, o intervalo de monitoramento é, de fato, um parâmetro importante no modelo analisado, visto que ele possui um impacto significante na disponibilidade do sistema.

O RTO e o RPO são outras duas métricas muito importantes de serem avaliadas quando se está estudando a recuperação de desastres. O RTO compreende o tempo

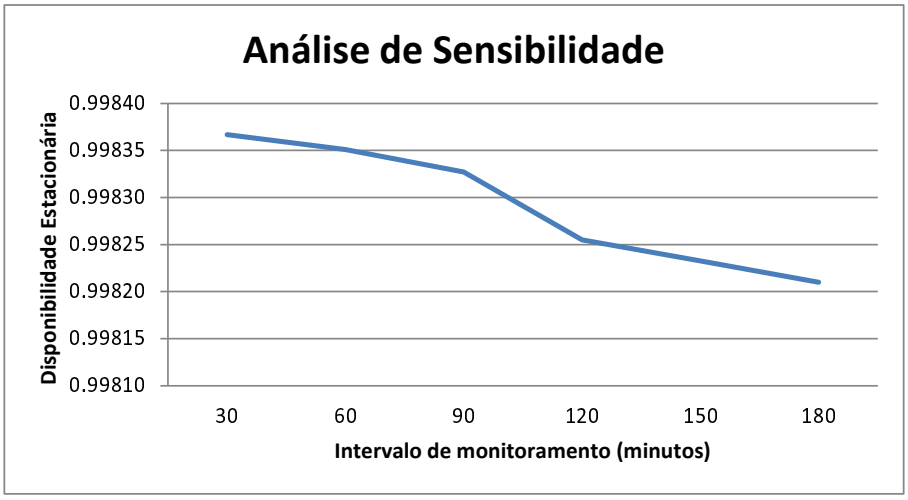


Figura 6.23: Análise de Sensibilidade.

máximo que o negócio pode suportar sem a solução tecnológica, enquanto o RPO compreende a quantidade máxima de dados que podem ser perdidos no processo de recuperação. O RPO, neste trabalho, é zero, já que os volumes primários e secundários são totalmente sincronizados (ver descrição acima). Por outro lado, o intervalo de monitoramento foi usado para calcular o valor máximo aceitável para o RTO de modo que esse intervalo satisfaça a expressão 6.2. Isto é, deve-se encontrar um intervalo de monitoramento, em que o custo anual do *downtime* da infraestrutura com recuperação de desastres ($Custo_{c/DR}$) e o custo da infraestrutura da nuvem ($Custo_{infra}$) devem ser menor que o custo anual do *downtime* da infraestrutura sem recuperação de desastres ($Custo_{s/DR}$).

$$Custo_{c/DR} + Custo_{infra} < Custo_{s/DR} \tag{6.2}$$

O custo da infraestrutura da nuvem é composto pelos custos de aquisição e custos energéticos, conforme apresentados na Equação 6.3. O custo total do *hardware* obtido foi de US$ 13.390,00 [DMAM12]. Por outro lado, não houve custos relacionados ao *software*, já que, neste estudo de caso, consideramos a nuvem *open-source* Eucalyptus, que é um *software* gratuito. Por fim, o custo energético no período de um ano ("T") foi de US$1.967,93 [oE13]. A Figura 6.24 apresenta uma comparação entre ambos os cenários (com e sem recuperação de desastres), em que o intervalo de monitoramento foi variado a fim de obter valores aceitáveis para o RTO. O gráfico mostra que o ponto máximo, em que o custo do *downtime* da infraestrutura com recuperação de desastre ($Custo_{c/DR}+Custo_{infra}$) é igual ao custo do *downtime* da infraestrutura sem recuperação de desastres ($Custo_{s/DR}$), é aproximadamente 100 minutos. Portanto, para garantir o RTO definido pela expressão 6.2, o intervalo de monitoramento deve ser mantido abaixo dos 100 minutos, aproximadamente.

$$Custo_{infra} = H_{ware} + S_{ware} + (Custo_{energetico} * T) \tag{6.3}$$

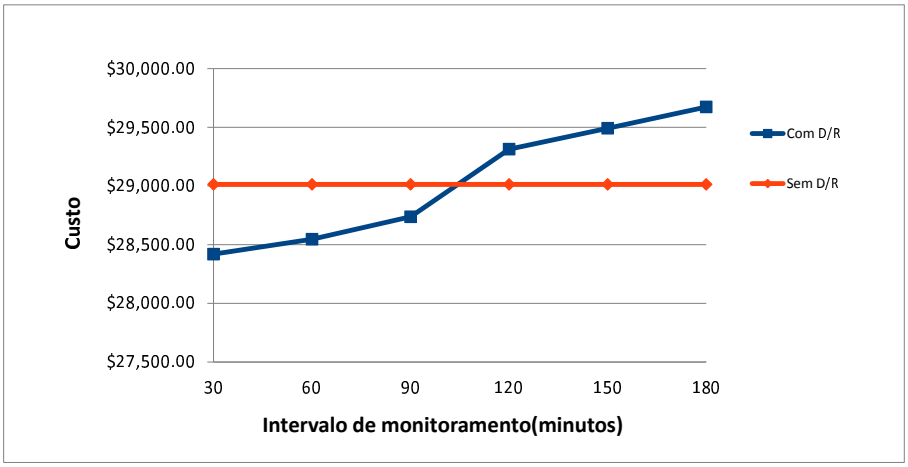


Figura 6.24: Comparativo dos Custo do Sistema com e sem Recuperação de Desastres.

6.3.4 Avaliação dos Modelos Através de Experimentos

Com o intuito de avaliar o modelo do sistema de recuperação de desastres, a disponibilidade estacionária do modelo foi comparado com a disponibilidade estacionária do sistema de recuperação de desastres (descrita acima). A disponibilidade da infraestrutura física do serviço é medida através de um *script shell* de monitoramento que a cada 10 segundos executa a aplicação cliente do *mashup*. Caso a aplicação cliente retorne alguma resposta num limite de tempo de 10 segundos, é registrado o estado *Up* (ativo) do serviço, caso contrário, é registrado o estado *Down*. Os eventos de falha e reparo são gerados através do núcleo da ferramenta Eucabomber [SMA+13], com os tempos médios entre falhas e tempos médios entre reparos seguindo a distribuição exponencial. A disponibilidade foi calculada como $A = \frac{uptime}{tempototal}$, em que o *uptime* é a soma de todos os intervalos de tempo em que o sistema de recuperação de desastre está disponível, i.e., todos os componentes de *hardware* e *software* estão funcionando.

A falha individual de cada máquina virtual, seja ela hospedada no *data center* ou na nuvem, é induzida pelo desligamento da mesma. A falha transiente do *data center* (aquela que não envolve um desastre) é representada pelo desligamento de todas as VMs simultaneamente. De forma similar, a falha da nuvem é representada pelo encerramento simultâneo de todas as VMs que executam nela. A ocorrência de um desastre no *data center* é emulada pelo desligamento do servidor físico onde as VMs estão hospedadas. O reparo de máquinas virtuais é realizado através da execução de comandos para executar as VMs novamente no *data center* ou reinstanciá-las na nuvem, dependendo do caso. O reparo do *data center* após o desastre é efetuado com o religamento do servidor físico que hospeda as VMs. Esse religamento é executado de forma automática e remota pelo núcleo da ferramenta Eucabomber, através do recurso Wake-on-LAN [Lie10] que está habilitado na placa-mãe do servidor utilizado. Todas as outras operações de falha e reparo previamente mencionadas são disparadas pela ferramenta Eucabomber, que remotamente executa *scripts shell* específicos para cada operação. Estes *scripts shell* estão instalados

no servidor do *data center* e na máquina controladora (*Cloud Controller*) da nuvem Eucalyptus.

Os experimentos para calcular a disponibilidade da infraestrutura física do sistema de recuperação de desastres foram executados por um período de 168 horas (1 semana). A disponibilidade e o intervalo de confiança obtidos a partir dos experimentos são apresentados na Tabela 6.16. A Tabela 6.16 também apresenta os resultados obtidos para os modelos DSPN. A métrica utilizada para calcular a disponibilidade dos modelos é apresentada na Tabela 6.15. Os parâmetros usados tanto para os modelos quanto para emular as falhas e os reparos da infraestrutura física são descritos na Tabela 6.17. Note que esses parâmetros foram reduzidos por um fator de 3000 para as falhas e um fator de 100 para os reparos, a fim de acelerar a ocorrência dos mesmos. A aplicação do fator de redução permite executar os experimentos mais rapidamente, visto que, se os valores reais obtidos fossem adotados, levariam meses até que uma falha ocorresse e horas para que falhas fossem corrigidas, demandando muito tempo para execução de cada experimento. A disponibilidade obtida através dos experimentos foi 0,7822950, enquanto a disponibilidade dos modelos foi 0,8041546. Existe uma diferença mínima de aproximadamente 2% entre os resultados, que é uma diferença pequena considerando os eventos envolvidos. Adicionalmente, o intervalo de confiança dos experimentos e dos modelos se sobrepõem, os quais são (0,7543693; 0,8050101) e (0,7820540; 0,8262552). Dessa forma, não há evidências estatísticas para refutar que os dados são equivalentes, considerando o nível de confiança de 95%.

Tabela 6.16: Validação dos Modelos DSPN Obtidos pelo Processo de Mapeamento com Nível de Confiança de 95%.

	Média	Limite Inferior do Intervalo	Limite Superior do Intervalo
Experimentos	0,78229	0,75436	0,80501
Modelos	0,80415	0,78205	0,82625

Tabela 6.17: Descrição dos Parâmetros de Simulação dos Experimentos.

Parâmetros	Simulado[horas]
Tempo Até a Falha do DC (desastre)	16,66666
Tempo de Reparo do DC (desastre)	0,48
Tempo Até a Falha do DC (transiente)	3,33333
Tempo de Reparo do DC (transiente)	0,04
Tempo Até a Falha das VMs do DC	0,33333
Tempo de Reparo das VMs do DC	0,01
Tempo Até a Falha das VMs da Nuvem	0,33333
Tempo de Reparo das VMs da Nuvem	0,01

6.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os resultados obtidos na realização de três estudos de caso. O primeiro trata da nuvem *open-source* Eucalyptus, enquanto o segundo consiste de um sistema de aplicação *Web* hospedado numa infraestrutura em nuvem. O último estudo lida com um sistema de recuperação de desastres como um serviço em nuvem. Além disso, experimentos foram montados e executados para demonstrar a corretude e aplicabilidade do *framework* proposto. Assim, através dos estudos aqui apresentados, foi possível demonstrar que os mecanismos de tratamento de interrupções são efetivos para garantir alta disponibilidade e continuidade dos serviços para os SD.

CAPÍTULO 7

CONCLUSÕES

I've missed more than 9000 shots in my career. I've lost almost 300 games. 26 times, I've been trusted to take the game winning shot and missed. I've failed over and over and over again in my life. And that is why I succeed.

—MICHAEL JORDAN

Com o aumento cada vez maior da dependência da nossa sociedade na continuidade dos serviços providos pelos sistemas distribuídos (ex.: computação nas nuvens e sistemas virtualizados), a alta disponibilidade dos mesmos se tornou algo extremamente importante de ser garantido, visto que as empresas (pequenas, médias ou grandes) estão susceptíveis às interrupções brandas e severas. Dessa forma, a análise de mecanismos de tratamento de interrupções é importante, já que pode prover medidas preventivas, estratégia de recuperação e considerações técnicas a fim de garantir alta disponibilidade e continuidade dos serviços para os SD.

Este trabalho apresentou um *framework*, baseado em métricas, modelos e ferramentas que auxilia os projetistas, os quais não possuem (ou possuem pouca) expertise em modelagem estocástica, a projetar e estudar as infraestruturas dos sistemas distribuídos e os mecanismos de tratamento de interrupções a partir de especificações descritas em SysML e MARTE. O método proposto consiste, primeiramente, no mapeamento dos diagramas da SysML, anotados através de MARTE, em modelos DSPN. Em seguida, as DSPN resultantes são compostas e sincronizadas para formar uma DSPN completa representando todo o sistema modelado. Por fim, as métricas de interesse são calculadas.

Este *framework* visa prover métricas de disponibilidade referentes as infraestruturas distribuídas e os mecanismos de tratamento de interrupções. Esse tipo de informação pode ser extremamente importante em um processo de tomada de decisão, a fim de maximizar a utilização dos recursos ao menor custo possível. A obtenção de um modelo analítico, como é o caso das DSPN, a partir de um modelo de alto nível procura garantir que os resultados obtidos na avaliação representem o sistema avaliado de maneira fidedigna. Contudo, para que as métricas possam ser obtidas no regime estacionário, é necessário que os modelos DSPN obtidos pelo processo de mapeamento possuam algumas propriedades fundamentais, tais como conservação e repetividade. Neste tese, foi demonstrado que os modelos gerados pelo processo de mapeamento possuem tais propriedades.

Experimentos também foram montados e executados a fim de analisar o *framework* proposto. Isto é, a infraestrutura do sistema real foi montada no nosso laboratório e o resultado da métrica de interesse foi comparada com o resultado obtido a partir dos

modelos DSPN. A diferença dos resultados foi miníma, demonstrando a eficiência do *framework* proposto. Além disso, a ferramenta OpenMADS foi desenvolvida para auxiliar todas as etapas do *framework* proposto. Essa ferramenta pode ser utilizada para: modelar os mecanismos de tratamento de interrupções e as infraestruturas dos sistemas distribuídos usando os diagramas da SysML e as anotações de MARTE, gerar os modelos DSPN através do processo de mapeamento automático, e estudar os mecanismos de tratamento de interrupções implantados nos sistemas distribuídos.

Com o intuito de demonstrar a aplicabilidade do *framework* proposto, como também demostrar que os mecanismos de recuperação/mitigação de interrupções são efetivos para melhorar a disponibilidade dos sistemas distribuídos, três estudos de casos foram apresentados. O primeiro deles consiste do estudo da nuvem *open-source* Eucalyptus, no qual uma abordagem hierárquica para os modelos obtidos foi usada. Nesse estudo de caso, também foram levados em consideração aspectos de envelhecimento de *software* e seus respectivos mecanismos de mitigação de interrupções. Já o segundo estudo de caso consiste de um sistema de aplicação *Web* hospedado numa infraestrutura em nuvem. Nesse estudo de caso, características especialmente importantes em termos de alta disponibilidade dos serviços em nuvem são consideradas, tais como zona de isolamento de falha, função de auto *scaling* e balanceador de cargas. O último estudo de caso abordou um sistema de recuperação de interrupções severas, utilizando uma infraestrutura de nuvem privada.

7.1 CONTRIBUIÇÕES

As contribuições deste trabalho são listadas abaixo. Estas estão organizadas por ordem decrescente (1- mais relevante e complexo, 8 - menos relevante e complexo).

1. ***Framework.*** O *framework per si* é uma contribuição. Ele auxilia os projetistas, os quais não possuem conhecimento em modelagem estocástica, a estudarem mecanismos de tratamento de interrupções a partir de especificações de alto nível descritas em SysML e MARTE.

2. **Mapeamento.** Embora existam vários trabalhos que lidem com o mapeamento das linguagens semiformais para os modelos formais, este trabalho é o primeiro que trata da sincronização e composição de diferentes tipos de diagramas da SysML.

3. **Análise Qualitativa de Propriedades.** Este trabalho mostrou que os modelos gerados pelo processo de mapeamento possuem um conjunto de propriedades, as quais permitem que eles possam ser adotados para o cálculo numérico das métricas no regime estacionário.

4. **Modelos DSPN.** Este trabalho apresenta um conjunto de modelos analíticos que representam características específicas de sistemas distribuídos (i.e.: zona de isolamento, balanceador de cargas etc.), bem como mecanismos de recuperação de

interrupções, tais como função de auto *scaling* e *failover*. A utilização combinada desses modelos até então não tinha sido endereçada por nenhum trabalho.

5. **Análises.** Devido à complexidade dos modelos DSPN obtidos pelo processo de mapeamento, uma abordagem hierárquica foi proposta. Nessa abordagem, o modelo de disponibilidade é composto por modelos de alto nível (ex.: diagrama de bloco) combinado com modelos baixo nível (ex.: Redes de Petri).

6. **Ferramenta.** As infraestruturas dos sistemas distribuídos podem ser complexas e o seu respectivo mapeamento é uma tarefa susceptível a erros. Assim, este trabalho desenvolveu um protótipo de uma ferramenta que suporta o trabalho proposto. Isto é, a ferramenta auxiliará os projetista na modelagem dos diagramas da SysML, no mapeamento desses diagramas e na análises dos modelos DSPN obtidos.

7. **SysML e MARTE.** Os diagramas da SysML, em conjunto com as anotações de MARTE, são usados para representar estruturas complexas dos sistemas distribuídos, levando em consideração as restrições de disponibilidade. Esses diagramas são usados tanto para representar os estados dinâmicos dos sistemas quanto as atividades administrativas, e tais representações não tinham sido endereçadas até então na literatura.

7.2 LIMITAÇÕES

As limitações deste trabalho são as seguintes:

- A geração de funções de guarda pelo *framework* proposto é realizada de forma manual. Em outras palavras, durante a sincronização dos modelos, é necessária a intervenção do projetista para atribuir as funções de guarda ao modelo gerado. Devido a esse aspecto, erros involuntários podem ser inseridos nos modelos, podendo resultar em resultados inesperados.

- Alguns elementos dos diagramas da SysML adotados neste trabalho não foram levados em consideração durante o processo de mapeamento, tais como: estado de história, junção, entrada, saída, sincronização, entre outros. Isso ocorreu devido a dois fatores principais: (i) complexidade do modelo gerado pelo processo de mapeamento e (ii) alguns desses elementos são pouco usados para modelar as infraestruturas dos sistemas distribuídos.

- Os estudos de casos apresentados neste trabalho focaram em sistemas distribuídos implantados em companhias de pequeno e médio porte. Todavia, existe uma crescente demanda por sistemas distribuídos de grande porte compostos por centenas ou até mesmo milhares de servidores. Esse tipo de cenário não foi considerado no trabalho proposto.

- O trabalho proposto limita-se a estudar métricas relacionadas à disponibilidade dos sistemas distribuídos. Isto é, questões relacionadas ao desempenho ou à combinação

das análises de desempenho e disponibilidade (performabilidade) não foram levadas em consideração.

7.3 TRABALHOS FUTUROS

Como trabalho futuro, pode-se incluir a aplicação do *framework* proposto em outros tipos de nuvens, tais como: nuvens híbridas ou nuvens móveis. Essas nuvens têm crescido significantemente ao longo dos últimos anos, sendo uma das principais tendências do mercado atual. Outro aspecto a ser considerado como extensão do trabalho proposto seria levar em consideração *datacenters* compostos por centenas ou milhares de servidores. Considerando as redes de Petri, tal cenário, certamente, resultaria na explosão do espaço de estados dos modelos gerados pelo processo de mapeamento. Para lidar com esse problema, possivelmente, seria necessário desenvolver técnicas de análise eficientes.

Embora o *framework* tenha sido concebido para auxiliar os projetistas na análise de métricas relacionadas a disponibilidade dos sistemas distribuídos, ele pode ser estendido para permitir outros tipos de análises dos sistemas. Um ponto interessante de extensão do *framework* seria a inclusão de aspectos de performabilidade. Com isso, podem ser necessárias extensões tanto no processo de mapeamento, quanto na modelagem do sistema, através da definição de novos componentes e regras de mapeamento. Além disso, as análises podem requerer que novas métricas sejam aferidas e novos parâmetros sejam informados.

A ferramenta, bem como a análise de propriedades propostas também podem ser melhorados em trabalhos futuros. A ferramenta pode ser estendida incluindo-se novas funcionalidades, bem como realizando a integração com outras ferramentas SysML existentes como o *Papyrus*, o *ARTiSAN Software Tool* e o *ParaMagic*. Tal integração pode ser feita através do desenvolvimento de um módulo que adapta o arquivo XML dos diagramas da SysML para o mesmo formato do arquivo de entrada da ferramenta que se deseja integrar. Também é possível estender a análise qualitativa considerando invariantes, bem como novos modelos.

REFERÊNCIAS BIBLIOGRÁFICAS

[AAM+13] Ermeson Andrade, Marcelo Alves, Rubens Matos, Bruno Silva, and Paulo Maciel. Openmads: An open source tool for modeling and analysis of distributed systems. In *The 32nd International Conference on Computer Safety, Reliability and Security*. Springer, 2013.

[AANM12] Ermeson Andrade, Marcelo Alves, Bruno Nogueira, and Paulo Maciel. Calau: An environment for modeling and analyzing embedded real-time systems. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 3135–3140. IEEE, 2012.

[AAPZ10] Bernardetta Addis, Danilo Ardagna, Barbara Panicucci, and Li Zhang. Autonomic management of cloud service centers with availability guarantees. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 220–227. IEEE, 2010.

[AFG+10] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[ALRL04] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

[Ama13] Amazon. Amazon EC2 service level agreement. http://aws. amazon. com/ec2-sla., 2013.

[AMC+08] Ermeson Andrade, Paulo Maciel, Gustavo Callou, Eduardo Tavares, and Bruno Nogueira. Mapping sysml state machine diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints. In *Advances in Electronics and Micro-electronics, 2008. ENICS'08. International Conference on*, pages 1–6. IEEE, 2008.

[AMC+09] Ermeson Andrade, Paulo Maciel, Gustavo Callou, Bruno Nogueira, and Carlos Araújo. Mapping uml sequence diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 377–381. ACM, 2009.

[AMC+10] Ermeson Andrade, Paulo Maciel, Gustavo Callou, Bruno Nogueira, and Carlos Araujo. An approach based in petri net for requirement analysis. *Pawel Pawlewski.(Org.). Petri Nets Applications.: INTECH*, pages 1–20, 2010.

[AMCB84] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, 2(2):93–122, 1984.

[AMCN09] Ermeson Andrade, Paulo Maciel, Gustavo Callou, and Bruno Nogueira. A methodology for mapping sysml activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints. In *Digital Society, 2009. ICDS'09. Third International Conference on*, pages 266–271. IEEE, 2009.

[AMF+10] Ermeson Andrade, Paulo Maciel, Tiago Falcão, Bruno Nogueira, Carlos Araujo, and Gustavo Callou. Performance and energy consumption estimation for commercial off-the-shelf component system design. *Innovations in Systems and Software Engineering*, 6(1):107–114, 2010.

[AMKT11] Ermeson Andrade, Fumio Machida, Dong-Seong Kim, and Kishor S Trivedi. Modeling and analyzing server system with rejuvenation through sysml and stochastic reward nets. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 161–168. IEEE, 2011.

[AMM+11] Jean Araujo, Rubens Matos, Paulo Maciel, Francisco Vieira, Rivalino Matias, and Kishor S Trivedi. Software rejuvenation in eucalyptus cloud computing infrastructure: a method based on time series forecasting and multiple thresholds. In *Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on*, pages 38–43. IEEE, 2011.

[AMN+05] Leonardo Amorim, P Maciel, M Nogueira, R Barreto, and E Tavares. A methodology for mapping live sequence chart to coloured petri net. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 4, pages 2999–3004. IEEE, 2005.

[AMN+06] Leonardo Amorim, Paulo Maciel, Meuse Nogueira, Raimundo Barreto, and Eduardo Tavares. Mapping live sequence chart to coloured petri nets for analysis and verification of embedded systems. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–25, 2006.

[AMN+10] Ermeson Andrade, Paulo Maciel, Bruno Nogueira, Carlos Araújo, and Gustavo Callou. A cots-based approach for estimating performance and energy consumption of embedded real-time systems. *Information Processing Letters*, 110(14):525–534, 2010.

[And09] Ermeson Andrade. Modelagem e análise de especificações de sistemas embarcados de tempo-real críticos com restrições de energia. *Dissertação de Mestrado, Centro de Informática, Universidade Federal de Pernambuco*, 2009.

[ASM+09] Carlos Araújo, Erica Sousa, Paulo Maciel, Fábio Chicout, and Ermeson Andrade. Performance modeling for evaluation and planning of electronic funds transfer systems with bursty arrival traffic. In *Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on*, pages 65–70. IEEE, 2009.

[AW97] Alberto Avritzer and Elaine J Weyuker. Monitoring smoothly degrading systems for increased dependability. *Empirical Software Engineering*, 2(1):59–77, 1997.

[BA12] Eric Bauer and Randee Adams. *Reliability and availability of cloud computing*. John Wiley & Sons, 2012.

[BB99] Mark Bakery and Rajkumar Buyyaz. Cluster computing at a glance. *High Performance Cluster Computing: Architectures and System. Upper Saddle River, NJ: Prentice-Hall*, pages 3–47, 1999.

[BDM02] Simona Bernardi, Susanna Donatelli, and José Merseguer. From uml sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the 3rd international workshop on Software and performance*, pages 35–45. ACM, 2002.

[BGdMT06] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor Shridharbhai Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience, 2006.

[BGdT98] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, 1998.

[Bha13] Subir Bhaumik. India floods: Thousands flee homes in assam, 2013.

[Bla11] Richard Black. *Japan earthquake: Explosion at Fukushima nuclear plant*. BBC, 2011.

[BP01a] Luciano Baresi and Mauro Pezzè. Improving uml with petri nets. *Electronic Notes in Theoretical Computer Science*, 44(4):107–119, 2001.

[BP01b] Luciano Baresi and Mauro Pezze. On formalizing uml with high-level petri nets. In *Concurrent Object-Oriented Programming and Petri Nets*, pages 276–304. Springer, 2001.

[Buy99] R. Buyya. High performance cluster computing: Systems and architectures, 1999.

[CA11] Massimo Cafaro and Giovanni Aloisio. *Grids, Clouds, and Virtualization*. Springer, 2011.

[CGH+09] Roy Campbell, Indranil Gupta, Michael Heath, Steven Y Ko, Michael Kozuch, Marcel Kunze, Thomas Kwan, Kevin Lai, Hing Yan Lee, Martha Lyons, et al. Open cirrustm cloud computing testbed: federated data centers for open source systems and services research. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 1–1. USENIX Association, 2009.

[CM06] Javier Campos and José Merseguer. On the integration of uml and petri nets in software development. In *Petri Nets and Other Models of Concurrency-ICATPN 2006*, pages 19–36. Springer, 2006.

[CMT89] G. Ciardo, J. Muppala, and K. Trivedi. Spnp: stochastic petri net package. In *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*, pages 142–151. IEEE, 1989.

[CMTA12] G. Callou, P. Maciel, D. Tutsch, and J. Araujo. Models for dependability and sustainability analysis of data center cooling architectures. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6. IEEE, 2012.

[CST00] Y Cao, HR Sun, and KS Trivedi. Performability analysis of tdma cellular systems. In *International Conference on the Performance and QoS of Next Generation Networking, PQNet2000, Nagoya, Japan*, 2000.

[Dan13] Jamilson Dantas. Modelos para análise de dependabilidade de arquiteturas de computação em nuvem. In *Universidade Federal de Pernambuco*. Dissertação de Mestrado, 2013.

[DdSS04] Luciano Mathias Döll, João Umberto Furquim de Souza, and Paulo Cesar Stadzisz. Verificação e validação de sistemas orientados a objetos usando redes de petri. In *I Workshop de Computação-WORKCOMP-SUL. Palhoça, SC, Brasil*, 2004.

[Der95] Kurt W Derr. *Applying OMT: A Practical step-by-step guide to using the Object Modeling Technique*. Cambridge University Press, 1995.

[DGPT00] Tadashi Dohi, Katerina Goseva-Popstojanova, and Kishor S Trivedi. Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. In *Dependable Computing, 2000. Proceedings. 2000 Pacific Rim International Symposium on*, pages 77–84. IEEE, 2000.

[DMAM12] Jamilson Dantas, Rubens Matos, Jean Araujo, and Paulo Maciel. An availability model for eucalyptus platform: An analysis of warm-standy replication mechanism. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 1664–1669. IEEE, 2012.

[DPP+05] S Distefano, D Paci, A Puliafito, M Scarpa, Contrada Papardo, and S Sperone. Design and implementation of a performance plug-in for the argouml tool. In *Proceedings of The International Conference of Software Engineering (SE2005)*, 2005.

[Dro13] Dropbox. Dropbox suffers global outage: Communication handled well, 2013.

[DSLT05] Vincent Debruyne, Françoise Simonot-Lion, and Yvon Trinquet. East adl an architecture description language. In *Architecture Description Languages*, pages 181–195. Springer, 2005.

[Eck10] Wayne W Eckerson. *Performance dashboards: measuring, monitoring, and managing your business*. Wiley, 2010.

[FBDSG07] M. Faugere, T. Bourbeau, R. De Simone, and S. Gerard. Marte: Also an uml profile for modeling aadl applications. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 359–364. IEEE, 2007.

[FGC+06] Patrick Farail, Pierre GOUTILLET, Agusti Canals, Christophe Le Camus, David Sciamma, Pierre Michel, Xavier Crégut, and Marc Pantel. The topcased project: a toolkit in open source for critical aeronautic systems design. *Ingenieurs de l'Automobile*, (781):54–59, 2006.

[FKT03] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid. *Berman et al.[2]*, pages 171–197, 2003.

[FLV03] Peter Feiler, Bruce Lewis, and Steve Vestal. The sae aadl standard: A basis for model-based architecture-driven embedded systems engineering. In *Workshop on Model-Driven Embedded Systems*, volume 5, 2003.

[FMS11] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2011.

[Gal95] Bill Gallmeister. *POSIX. 4 Programmers Guide: Programming for the real world*. O'Reilly Media, Inc., 1995.

[Gär99] F.C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys (CSUR)*, 31(1):1–26, 1999.

[GB11] Henson Graves and Yvonne Bijan. Using formal methods with sysml in aerospace design and engineering. *Annals of Mathematics and Artificial Intelligence*, 63(1):53–102, 2011.

[GDTS11] Sébastien Gérard, Cédric Dumoulin, Patrick Tessier, and Bran Selic. Papyrus: a uml2 tool for domain-specific language modeling. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 361–368. Springer, 2011.

[Ger00] Reinhard German. *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. Wiley New York, 2000.

[Gho12] Rahul Ghosh. *Scalable Stochastic Models for Cloud Services*. PhD thesis, Duke University, 2012.

[GLT04] Swapna S Gokhale, Michael R Lyu, and Kishor S Trivedi. Analysis of software fault removal policies using a non-homogeneous continuous time markov chain. *Software Quality Journal*, 12(3):211–230, 2004.

[GMM06] Elena Gómez-Martínez and José Merseguer. Argospe: Model-based software performance engineering. In *Petri Nets and Other Models of Concurrency-ICATPN 2006*, pages 401–410. Springer, 2006.

[GMT08] Michael Grottke, R Matias, and Kishor S Trivedi. The fundamentals of software aging. In *Software Reliability Engineering Workshops, 2008. ISSRE Wksp 2008. IEEE International Conference on*, pages 1–6. IEEE, 2008.

[GPTT95] Sachin Garg, Antonio Puliafito, M Telek, and Kishor S Trivedi. Analysis of software rejuvenation using markov regenerative stochastic petri net. In *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, pages 180–187. IEEE, 1995.

[GT05] MICHAEL Grottke and KISHOR S Trivedi. Software faults, software aging and software rejuvenation. *Journal of the Reliability Engineering Association of Japan*, 27(7):425–438, 2005.

[GT07] Michael Grottke and Kishor S Trivedi. Fighting bugs: Remove, retry, replicate, and rejuvenate. *Computer*, 40(2):107–109, 2007.

[GTNK10] Rahul Ghosh, Kishor S Trivedi, Vijay K Naik, and Dong Seong Kim. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach. In *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on*, pages 125–132. IEEE, 2010.

[Gue08] Gilleanes TA Guedes. *UML: uma abordagem prática*. Novatec Editora, 2008.

[GV02] Claude Girault and Rüdiger Valk. *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer, 2002.

[GYJ+10] M. Gallet, N. Yigitbasi, B. Javadi, D. Kondo, A. Iosup, and D. Epema. A model for space-correlated failures in large-scale distributed systems. *Euro-Par 2010-Parallel Processing*, pages 88–100, 2010.

[H+06] M. Hause et al. The sysml modelling language. In *Fifteenth European Systems Engineering Conference*, volume 9. Citeseer, 2006.

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[HGG+10] Tao Hu, Minyi Guo, Song Guo, Hirokazu Ozaki, Long Zheng, Kaoru Ota, and Mianxiong Dong. Mttf of composite web services. In *Parallel and Distributed Processing with Applications (ISPA), 2010 International Symposium on*, pages 130–137. IEEE, 2010.

[HKKF95] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 381–390. IEEE, 1995.

[IAK+07] M Ioualalen, J Asavanant, N Kaewbanjak, ST Grilli, JT Kirby, and P Watts. Modeling the 26 december 2004 indian ocean tsunami: Case study of impact in thailand. *Journal of Geophysical Research: Oceans (1978–2012)*, 112(C7), 2007.

[Jan98] V Janoušek. *Modelling Objects by Petri Nets*. PhD thesis, PhD. thesis, Brno University of Technology, Brno, Czech Republic, 1998.

[JC11] Thienne Johnson and Mauro Coutinho. *Avaliação de Desempenho de Sistemas Computacionais*. LTC, 2011.

[Jen91] Kurt Jensen. *Coloured Petri nets: A high level language for system design and analysis*. Springer, 1991.

[JJH+10] Gueyoung Jung, Kaustubh R Joshi, Matti A Hiltunen, Richard D Schlichting, and Calton Pu. Performance and availability aware regeneration for cloud based multitier applications. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 497–506. IEEE, 2010.

[JKW07] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.

[KF02] Todd Kimberlain and James Franklin. Hurricane sandy tropical cyclone update?(report). national hurricane center. Technical report, Retrieved 2012-10-24, 2002.

[KMT09] D.S. Kim, F. Machida, and K.S. Trivedi. Availability modeling and analysis of a virtualized system. In *Dependable Computing, 2009. PRDC'09. 15th IEEE Pacific Rim International Symposium on*, pages 365–371. IEEE, 2009.

[Kos12] Anthony Wing Kosner. *Amazon Cloud Goes Down Friday Night, Taking Netflix, Instagram And Pinterest With It*. Forbes, 2012.

[KP99] Peter King and Rob Pooley. Using uml to derive stochastic petri net models. In *Department of Computer Science, University of Bristol*. Citeseer, 1999.

[KP00] Peter King and Rob Pooley. Derivation of petri net performance models from uml specifications of communications software. In *Computer Performance Evaluation. Modelling Techniques and Tools*, pages 262–276. Springer, 2000.

[Lap92] Jean-Claude Laprie. Dependability: Basic concepts and terminology, volume 5 of dependable computing and fault-tolerant systems. *Springer-V erlag*, 1:992, 1992.

[LGMC04] Juan Pablo López-Grao, José Merseguer, and Javier Campos. From uml activity diagrams to stochastic petri nets: application to software performance engineering. *ACM SIGSOFT software engineering notes*, 29(1):25–36, 2004.

[Lie10] Philip Lieberman. Wake-on-lan technology, 2010.

[LKD+03] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. Web service level agreement (wsla) language specification. *IBM Corporation*, pages 815–824, 2003.

[LML+06] C.J.P. Lucena, C.M.B. Medeiros, C.L. Lucchesi, J.C. Maldonado, and V.A.F. Almeida. Grandes desafios da pesquisa em computação no brasil–2006–2016. *Relatório técnico, Sociedade Brasileira de Computação*, 2006.

[LPK+00] Jonathan Lee, Jiann-I Pan, Jong-Yih Kuo, Yong-Yi Fanjiang, and Stephen Yang. Towards the verification of scenarios with time petri-nets. In *Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International*, pages 503–508. IEEE, 2000.

[LTP95] Dimitris Logothetis, Kishor S Trivedi, and Antonio Puliafito. Markov regenerative models. In *Computer Performance and Dependability Symposium, 1995. Proceedings., International*, pages 134–142. IEEE, 1995.

[LTY09] TT Lwin, T. Thein, and M. Yangon. High availability cluster system for local disaster recovery with markov modeling approach. *International Journal of Computer Science Issues(IJCSI)*, 6(2):25, 2009.

[LYT03] M. Lanus, L. Yin, and K.S. Trivedi. Hierarchical composition and aggregation of state-based availability and performability models. *Reliability, IEEE Transactions on*, 52(1):44–52, 2003.

[MAKT11] Fumio Machida, Ermeson Andrade, Dong Seong Kim, and Kishor S Trivedi. Candy: Component-based availability modeling framework for cloud service management using sysml. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 209–218. IEEE, 2011.

[Mar90] M Ajmone Marsan. Stochastic petri nets: an elementary introduction. In *Advances in Petri Nets 1989*, pages 1–29. Springer, 1990.

[MBC+95] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[MC87] M. Marsan and G. Chiola. On petri nets with deterministic and exponentially distributed firing times. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of *Lecture Notes in Computer Science*, pages 132–145. Springer Berlin / Heidelberg, 1987.

[MCBD02] José Merseguer, Javier Campos, Simona Bernardi, and Susanna Donatelli. A compositional semantics for uml state machines aimed at performance evaluation. In *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, pages 295–302. IEEE, 2002.

[MF76] Philip Merlin and David Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, 1976.

[MG11] Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[MKT10] F. Machida, D.S. Kim, and K.S. Trivedi. Modeling and analysis of software rejuvenation in a server virtualized system. In *Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second International Workshop on*, pages 1–6. IEEE, 2010.

[MLC96] Paulo RM Maciel, Rafael D Lins, and Paulo RF Cunha. *Introdução às redes de Petri e aplicações*. UNICAMP-Instituto de Computacao, 1996.

[MMT94] Manish Malhotra, Jogesh K Muppala, and Kishor S Trivedi. Stiffness-tolerant methods for transient analysis of stiff markov chains. *Microelectronics Reliability*, 34(11):1825–1841, 1994.

[Moo02] Alan Moore. Extending the rt profile to support the osek infrastructure. In *Object-Oriented Real-Time Distributed Computing, 2002.(ISORC 2002). Proceedings. Fifth IEEE International Symposium on*, pages 341–347. IEEE, 2002.

[MTMK11] Paulo RM Maciel, Kishor S Trivedi, Rivalino Matias, and Dong Seong Kim. Dependability modeling. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, page 53, 2011.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[NN73] Jerre D Noe and Gary J Nutt. Macro e-nets for representation of parallel systems. *Computers, IEEE Transactions on*, 100(8):718–727, 1973.

[NWG+09] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*, pages 124–131. IEEE, 2009.

[oE13] U.S. Department of Energy. *City of Palo Alto utilities.* http://energy.gov/savings/city-palo-alto-utilities-palo-alto-clean-clean-local-energy-accessible-now, 2013.

[OMG03] UML OMG. Profile for schedulability, performance, and time specification. *Object Management Group*, 2003.

[OMG11] UML OMG. Profile for marte: Modeling and analysis of real-time embedded systems, version 1.1, 2011.

[Par10] Visual Paradigm. Visual paradigm for uml. *Hong Kong: Visual Paradigm International. Available at: http://www. visual-paradigm. com/product/vpuml/. Accessed April*, 15:2010, 2010.

[Pet66] C.A. Petri. *Communication with automata.* Technical report, Princeton, 1966.

[Pet77] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.

[PP05] D. Pilone and N. Pitman. *UML 2.0 in a Nutshell.* O'Reilly Media, Incorporated, 2005.

[PPP11] Luka Perkov, Nikola Pavkovic, and Juraj Petrovic. High-availability using open source software. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 167–170. IEEE, 2011.

[Pre11] C. Preimesberger. *Unplanned IT Downtime Can Cost $5K Per Minute: Report*. http://www.eweek.com/c/a/IT-Infrastructure/Unplanned-IT-Downtime-Can-Cost-5K-Per-Minute-Report-549007/, 2011.

[Pri08] Paul J Prisaznuk. Arinc 653 role in integrated modular avionics (ima). In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 1–E. IEEE, 2008.

[Rac] Rackspace. Disaster recovery planning. Available on `http://www.rackspace.com/disaster-recovery-planning`.

[Ram74] Chander Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. 1974.

[RBL+09] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio Martín Llorente, Rubén Montero, Yaron Wolfsthal, Erik Elmroth, Juan Caceres, et al. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4–1, 2009.

[RCOW12] Shriram Rajagopalan, Brendan Cully, Ryan O'Connor, and Andrew Warfield. Secondsite: disaster tolerance as a service. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, VEE '12, pages 97–108, New York, NY, USA, 2012. ACM.

[Rec94] ITUT Recommendation. 800: Terms and definitions related to quality of service and network performance including dependability. *ITU-T August 1994*, 1994.

[Rei85] Wolfgang Reisig. *Petri nets: an introduction*. Springer-Verlag New York, Inc., 1985.

[Rei02] Philipp Reisner. Distributed replicated block device. In *Int. Linux System Tech. Conf*, 2002.

[RKK08] A-E Rugina, Karama Kanoun, and Mohamed Kaâniche. The adapt tool: From aadl architectural models to stochastic petri nets through model transformation. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 85–90. IEEE, 2008.

[RWL+03] Anne Vinter Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. Cpn tools for editing, simulating, and analysing coloured petri nets. In *Applications and Theory of Petri Nets 2003*, pages 450–462. Springer, 2003.

[S+06] Alexandre José da Silva et al. Aspectos da modelagem em sysml ligados à seleção de processador para sistema embutido. 2006.

[S+10] M Swanson et al. Contingency planning guide for federal information systems, special publication 800-34 rev. 1. *National Institute of Standards and Technology: Gaithersburg, MD*, 2010.

[SCT+13] B. Silva, G. Callou, E. Tavares, P. Maciel, J. Figueiredo, E. Sousa, C. Araujo, F. Magnani, and F. Neves. Astro: An integrated environment for dependability and sustainability evaluation. *Sustainable Computing: Informatics and Systems*, 3(1):1 – 17, 2013.

[Sil11] Bruno Silva. Dissertação de mestrado em ciência da computação: Astro - uma ferramenta para avaliação de dependabilidade e sustentabilidade em sistemas data center. *Centro de Informática, UFPE*, 2011.

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[SMA+13] Debora Souza, Rubens Matos, Jean Araujo, Vandi Alves, and Paulo Maciel. A tool for automatic dependability test in eucalyptus cloud computing infrastructures. *Computer and Information Science*, 6(3):p57, 2013.

[Smi03] David M Smith. The cost of lost data. *Journal of Contemporary Business Practice*, 6(3), 2003.

[SMT+10] B. Silva, P. Maciel, E. Tavares, C. Araujo, G. Callou, E. Sousa, N. Rosa, M. Marwah, R. Sharma, A. Shah, et al. Astro: A tool for dependability evaluation of data center infrastructures. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 783–790. IEEE, 2010.

[SRT00] Archana Sathaye, Srinivasan Ramani, and Kishor S Trivedi. Availability models in practice. In *FTCC-1. Proceedings of the Int. Workshop on Fault-Tolerant Control and Computing, Shoel, Korea*, pages 823–829, 2000.

[ST01] Ronald Simon and Sheldon Teperman. The world trade center attack: lessons for disaster management. *Critical Care*, 5(6):318, 2001.

[STP96] R.A. Sahner, K.S. Trivedi, and A. Puliafito. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Kluwer Academic Publishers, 1996.

[TAM12] K. Trivedi, E. Andrade, and F. Machida. Combining performance and availability analysis in practice. *Advances in Computers: Dependable and Secure Systems Engineering*, 84:1, 2012.

[Tea12] SysML Merge Team. Systems modeling language (sysml) specification. *OMG document: formal/2012-06-01*, 2012.

[Tea13a] Amazon Team. *Amazon EC2 SLA*. http://aws.amazon.com/ec2-sla/, 2013.

[Tea13b] ArgoUML Team. The argouml tool, 2013.

[TGA10] Kishor S Trivedi, Michael Grottke, and Ermeson Andrade. Software fault mitigation and availability assurance techniques. *International Journal of System Assurance Engineering and Management*, 1(4):340–350, 2010.

[TJZ07] Jan Trowitzsch, Dan Jerzynek, and Armin Zimmermann. A toolkit for performability evaluation based on stochastic uml state machines. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, page 30. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[Tri82] K.S. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*, volume 3982. Prentice-hall Englewood Cliffs, 1982.

[TVS02] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*, volume 2. Prentice Hall, 2002.

[TZ05] J Trowitzsch and A Zimmermann. Real-time uml state machines: an analysis approach. *Object oriented software design for real time and embedded computer systems*, 2005.

[TZ06] Jan Trowitzsch and Armin Zimmermann. Using uml state machines and petri nets for the quantitative investigation of etcs. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, page 34. ACM, 2006.

[TZH05] Jan Trowitzsch, Armin Zimmermann, and Günter Hommel. Towards quantitative analysis of real-time uml using stochastic petri nets. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 139b–139b. IEEE, 2005.

[Uml04] OMG Uml. 2.0 superstructure specification. *OMG, Needham*, 2004.

[VCF+06] Nandamudi Lankalapalli Vijaykumar, SV Carvalho, Carlos Renato Lisboa Francês, Vakulathil Abdurahiman, and ASM Amaral. Performance evaluation from statecharts representation of complex systems: Markov approach. *IV WPerformance, SBC*, pages 183–202, 2006.

[VDAVHR00] WMP Van Der Aalst, KM Van Hee, and HA Reijers. Analysis of discrete-time stochastic petri nets. *Statistica Neerlandica*, 54(2):237–255, 2000.

[VHHT01] Kalyanaraman Vaidyanathan, Richard E Harper, Steven W Hunter, and Kishor S Trivedi. Analysis and implementation of software rejuvenation in cluster systems. In *ACM SIGMETRICS Performance Evaluation Review*, volume 29, pages 62–71. ACM, 2001.

[VMw12] VMware. *A Guide to Modern IT Disaster Recovery*. http://www.neorhino.com/wp-content/uploads/2012/03/Guide-to-Modern-IT-Disaster-Recovery.pdf, 2012.

[War12] Matt Warman. *Gmail goes down worldwide*. The Telegraph, 2012.

[WCR+10] T. Wood, E. Cecchet, KK Ramakrishnan, P. Shenoy, J. Van der Merwe, and A. Venkataramani. Disaster recovery as a cloud service: Economic benefits & deployment challenges. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 8–8. USENIX Association, 2010.

[Wei11] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann, 2011.

[XDP04] Min Xie, Yuan-Shun Dai, and Kim-Leng Poh. *Computing system reliability: models and analysis*. Springer, 2004.

[YAM+12] Xiaoyan Yin, Javier Alonso, Fumio Machida, Ermeson Andrade, and Kishor S Trivedi. Availability modeling and analysis for data backup and restore operations. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 141–150. IEEE, 2012.

[ZKHH06] A. Zimmermann, M. Knoke, A. Huck, and G. Hommel. Towards version 4.0 of timenet. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2006 13th GI/ITG Conference*, pages 1–4. VDE, 2006.

[ZWST03] Xinyu Zang, Dazhi Wang, Hairong Sun, and Kishor S Trivedi. A bdd-based algorithm for analysis of multistate systems with multistate components. *Computers, IEEE Transactions on*, 52(12):1608–1618, 2003.

[ZZLK10] Zibin Zheng, Tom Chao Zhou, Michael R Lyu, and Irwin King. Ftcloud: A component ranking framework for fault-tolerant cloud applications. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 398–407. IEEE, 2010.