Rodrigo Silva Lima

Subsídios para Avaliação de Disponibilidade em Arquiteturas de Software de Sistemas-de-Sistemas com base em Simulações

Vitória da Conquista/BA

2019

Rodrigo Silva Lima

Subsídios para Avaliação de Disponibilidade em Arquiteturas de Software de Sistemas-de-Sistemas com base em Simulações

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação na Universidade Estadual do Sudoeste da Bahia – UESB.

Universidade Estadual do Sudoeste da Bahia – UESB

Curso de Bacharelado em Ciência da Computação

Departamento de Ciências Exatas e Tecnológicas

Orientador: Prof. Dr. Valdemar Vicente Graciano Neto

Coorientador: Prof. Dra. Maísa Soares dos Santos Lopes

Vitória da Conquista/BA

2019

Rodrigo Silva Lima
Subsídios para Avaliação de Disponibilidade em Arquiteturas de Software de Sistemas-de-Sistemas com base em Simulações/ Rodrigo Silva Lima. – Vitória da Conquista/BA, 2019-
63 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Valdemar Vicente Graciano Neto

Trabalho de Conclusão de Curso (Graduação)
Universidade Estadual do Sudoeste da Bahia – UESB
Curso de Bacharelado em Ciência da Computação
Departamento de Ciências Exatas e Tecnológicas, 2019.

1. Sistemas-de-Sistemas. 2. Arquitetura de software. 3. Simulação de modelos. I. Dr. Valdemar Vicente Graciano Neto. II. Universidade Estadual do Sudoeste da Bahia. III. Curso de Ciência da Computação. IV. Subsídios para Avaliação Baseada em Simulações do Atributo de Qualidade Disponibilidade em Arquiteturas de Software de Sistemas-de-Sistemas

Rodrigo Silva Lima

Subsídios para Avaliação de Disponibilidade em Arquiteturas de Software de Sistemas-de-Sistemas com base em Simulações

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação na Universidade Estadual do Sudoeste da Bahia – UESB.

Trabalho aprovado. Vitória da Conquista/BA, 30 de agosto de 2019:

Prof. Dr. Valdemar Vicente Graciano Neto
Orientador

Profa. Dra. Maísa Soares dos Santos Lopes
Coorientadora

Prof. Dr. Hélio Lopes dos Santos
Convidado 1

Vitória da Conquista/BA
2019

Este trabalho é dedicado a Bárbara, Gerffeson e Lucineide,
o maior alicerce da minha vida.

Agradecimentos

Chega ao fim um ciclo da minha vida, concluo o curso que sonhei um dia poder iniciar e saio completamente diferente do que quando entrei. Sou muito grato por tudo que tive a oportunidade de vivenciar e aprender e sei que muitas pessoas foram essenciais para esta conquista.

Agradeço ao meu Pai, Gerffeson Novaes Lima, por ter dedicado a vida a prover o melhor para mim e pelos conselhos que levarei para a vida toda, qualquer vitória minha é consequência da sua luta.

Agradeço a minha Mãe, Lucineide Silva Lima, por toda a sua preocupação, carinho e dedicação que só a senhora tem, lutarei com todas as minhas forças para retribuir o que a senhora faz por mim.

Agradeço a minha Irmã, Bárbara Silva Lima, por ter sido minha parceira nesses quase 6 anos morando em Conquista sob o mesmo teto, tudo o que passamos aqui, só a gente sabe,

Agradeço a minha namorada, Fernanda Nogueira Macena, por estar sempre ao meu lado e ser o meu amparo nos momentos mais difíceis.

Agradeço ao meu amigo/irmão Bruno Peters, por acreditar tanto em mim e por essa amizade tão verdadeira. Agradeço aos veteranos que me inseriram na vida acadêmica: Iago, Jéssica, Vinícius e Thomas. Agradeço aos paceiros que este curso me deu: João Vitor, Konai, Leandro, Raphael e Matheus Thiago.

Agradeço aos colegas de curso que se tornaram colegas de trabalho e que sei que posso contar para tudo: Carlos Hatus, Helber Henrique, Lucas Badaró, Wali Queiroz e Victor Willer. Agradeço também aos colegas do NTI, que me fizeram crescer muito como pessoa e profissional.

Agradeço ao todos os professores do curso de Ciência da Computação da UESB. Um agradecimento em especial ao prof. Dr. Hélio Lopes e a minha co-orientadora, a profa. Dra. Maísa Soares, por toda a ajuda, conselhos, ensinamentos e oportunidades que me deram. Agradeço também a profa. Dra. Alzira Ferreira e ao prof. Dr. Roque Trindade, pelos ensinamentos e apoio nos projetos de pesquisa. Agradeço também a Celina Pereira, eu não estaria formando se não fosse ela, obrigado por ser a mãezona do nosso curso.

Agradeço especialmente ao meu orientador, o prof. Dr. Valdemar Vicente Graciano Neto, por ter mudado a minha vida acadêmica. Sou muito grato por ter aceito este desafio de orientar um estudante de outra universidade e por fazê-lo de forma tão dedicada e atenciosa. Jamais esquecerei dos feriados, fins de semana e de momentos de descanso que abdicou para responder aos meus e-mails ou até mesmo para nos reunirmos.

"Those who can imagine anything, can create the impossible."

(Alan Turing)

Resumo

Sistemas de Sistemas (SoS) são um conjunto de sistemas intensivos em software independentes destinados a apoiar domínios críticos, como saúde e gerenciamento de crises e emergências. Para tanto, o SoS deve exibir comportamentos robustos e garantir que todo o sistema esteja continuamente disponível, sem falhar e, como consequência, deixando seus usuários desamparados. No entanto, a arquitetura dinâmica inerente do SoS pode afetar potencialmente a disponibilidade das funcionalidades oferecidas pelo SoS. Por isso, é importante antecipar, ainda em tempo de projeto, o grau de disponibilidade que um SoS pode oferecer e fornecer mecanismos para reforçar a disponibilidade fornecida por ele. Normalmente, os arquitetos de software usam modelos estáticos para prever atributos de qualidade em arquiteturas de software. No entanto, esses modelos não são bem-sucedidos para lidar com a natureza dinâmica das arquiteturas de SoS. As principais contribuição desta monografia são: (i) fornecer um modelo de falhas para simulações de arquitetura de software de SoS e (ii) subsídios para o estabelecimento de uma abordagem baseada em simulação para prever empiricamente e medir a disponibilidade de arquiteturas de software de SoS em tempo de projeto. Um SoS de monitoramento de enchentes foi utilizado como plataforma experimental. Resultados preliminares indicam que o modelo de simulação gerou com sucesso as falhas desejadas e a natureza da disponibilidade nas arquiteturas de SoS não é influenciada apenas pela disponibilidade dos constituintes do SoS, mas também pela topologia do SoS.

Palavras-chave: Sistemas-de-Sistemas, SoS, Simulação, Arquitetura de Software, Atributos de qualidade.

Abstract

Systems-of-Systems (SoS) are a set of independent software-intensive systems intended to support critical domains, such as emergency and crisis response systems and health. As such, SoS should exhibit robust behaviors and guarantee that the entire system is continuously available, not failing and leaving its users helpless. However, the SoS inherent dynamic architecture can potentially affect the availability of the functionalities being offered by the SoS. Hence, it is prominently important to anticipate, at design-time, the degree of availability a SoS can offer and provide mechanisms to reinforce the availability provided by it. Usually, software architects use static models to predict quality attributes in software architectures. However, these models are not well-succeeded to deal with the dynamic nature of the SoS architectures. The main contribution of this monography is thus providing a fault generator for SoS software architecture simulation models and presenting contributions to futurely support the establishment of a simulation-based approach for empirically predicting and measuring availability of SoS software architectures at design time. A Flood Monitoring SoS was used as the experimental platform. Preliminary results indicate that the simulation model successfully generated the desired faults and the nature of availability in SoS architectures is not only influenced by the availability of the SoS constituents, but also by the SoS topology.

Keywords: Systems-of-Systems, SoS, Simulation, Software Architecture, Quality Attributes.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DEVS: *Discrete Event System Specification*

EC: *Encapsulated Constituent*

ER: *Error Receiver*

FMSoS: *Flood Monitoring System-of-Systems*

FSG: *Fault Stimuli Generator*

GQM: *Goal-Question-Metric*

ISO: *International Organization for Standardization*

M&S: *Modeling & Simulating*

OCL: *Object Constraint Language*

SES: *System Entity Structure*

SoS: *Systems-of-Systems*

SySML: *Systems Modeling Language*

UML: *Unified Modeling Language*

Sumário

Parte I

Preparação da pesquisa

1 Introdução

1.1 Contextualização

O crescimento da população urbana tem sido evidenciado nos últimos anos (BHATTA, 2010). Aliado ao desenvolvimento de novas tecnologias de informação e de comunicação, os seres humanos nunca estiveram em posse de tantos softwares e hardwares distintos. Sistemas integrados a carros, celulares ou até mesmo sistemas mais complexos como os financeiros ou hospitalares são comuns e necessários em nossa sociedade atualmente. Neste contexto, como forma de extrair novas funcionalidades dos recursos já existentes, surge a necessidade de sistemas isolados passarem a interoperar. Interoperabilidade é a capacidade de dois ou mais sistemas ou componentes trocarem informações e usarem as informações que foram trocadas (IEEE, 1991). A interoperabilidade aumenta a capacidade dos sistemas, visto que estes poderão ter acesso a dados ou "[...] funcionalidades mais complexas as quais não poderiam ser providas por nenhum dos sistemas se estivessem operando individualmente"(NAKAGAWA et al., 2013). Os sistemas podem estar isolados geograficamente, serem executados em outras plataformas ou mesmo pertencerem a outra organização. Interoperabilidade de sistemas é uma característica essencial em qualquer sistema para que possa interagir com outros sistemas durante sua missão sem nenhum problema ou antecipando estes e seus efeitos quando necessário (CHAPURLAT; DACLIN, 2012).

Nesse cenário onde há necessidade de interoperabilidade mesmo em sistemas que não foram projetados para tal ou em sistemas altamente complexos e que afetam diretamente vidas humanas como em cidades inteligentes (NAM; PARDO, 2011), surge o conceito de Sistemas-de-Sistemas ou SoS (Do inglês *System-of-Systems*). SoS são um conjunto de sistemas constituintes independentes e heterogêneos que formam um sistema maior com o objetivo de alcançar um conjunto de missões (Graciano Neto, 2016). SoS possuem arquitetura dinâmica, isto é, seus constituintes podem entrar, auxiliar o SoS a atingir sua missão e sair do SoS a qualquer momento. Devido a arquitetura dinâmica, o ciclo de vida de um SoS se torna difícil de ser previsto em tempo de projeto. Visto que sistemas complexos[1], a classe de sistemas na qual SoS faz parte, requerem métodos que suportem validações e verificações adequadas (Graciano Neto, 2018). Devido ao alto custo e o risco envolvendo a implantação de Sistemas-de-Sistemas, surge a necessidade de se projetar e simular a arquitetura dos SoS no início do ciclo de vida do projeto.

[1] Sistema complexo é um sistema sociotécnico coletivo com muitos componentes independentes e interdependentes, que interagem de maneira não linear (o comportamento não pode ser expresso como somatórios da atividade de componentes individuais) e possuem interdependências que são difíceis de descrever, prever e projetar (ALAMPALLI; PARDO, 2014)

A disponibilidade é atualmente uma propriedade bem conhecida para sistemas de software isolados (ISO, 2011). No entanto, o SoS é caracterizado por ter uma arquitetura dinâmica, ou seja, os constituintes podem ingressar ou deixar a arquitetura do SoS em tempo de execução. Assim, o impacto de tal dinâmica na disponibilidade de SoS está sujeito a investigação (OQUENDO, 2016b). Portanto, assim como outros atributos de qualidade, a disponibilidade deve ter sua definição e natureza revisadas para o contexto de SoS (BIANCHI; SANTOS; FELIZARDO, 2015). As simulações são consideradas uma técnica de prototipagem de alta fidelidade para prever propriedades de vários tipos de sistemas (RUDD; STERN; ISENSEE, 1996). Além disso, também fornecem o rigor e o formalismo necessários para sustentar avaliações empíricas de sistemas baseados em software (FRANÇA; TRAVASSOS, 2012; FRANÇA; TRAVASSOS, 2016). Além disso, em relação ao SoS e suas arquiteturas dinâmicas, as simulações podem fornecer uma infraestrutura subjacente para prever, em tempo de projeto, o impacto das alterações arquiteturais no SoS em tempo de execução, avaliando empiricamente o grau de disponibilidade fornecido por todo o SoS como consequência do disponibilidade individual de seus constituintes, garantindo a operação contínua do SoS, possibilitando a previsão e evitando desastres.

1.2 Problemas

SoS são formados por sistemas constituintes. Estes, por sua vez, como são intensivos em software, possuem uma arquitetura de software, isto é, a estrutura fundamental de um sistema de software e a disciplina para criar tais estruturas e sistemas. Cada estrutura compreende elementos de software, relações entre eles e propriedades de ambos os elementos e relações (CLEMENTS et al., 2002). A partir disso, emerge um novo conceito chamado arquitetura de software de SoS, que é formado por todas as unidades de software envolvidas no SoS, suas propriedades e relações. Tal arquitetura também é influenciada por atributos de qualidade priorizados, dentre eles a disponibilidade, que é inerentemente afetada pela arquitetura dinâmica.

As arquiteturas de software podem ser documentadas usando modelos, que auxiliam o arquiteto a raciocinar sobre custo, cronograma e até mesmo para ditar a estrutura da organização. Os arquitetos de software devem ser capazes de prever, ainda em tempo de projeto, a resposta de um sistema aos seus atributos de qualidade, analisando modelos de arquitetura e avaliando sua qualidade (BASS; CLEMENTS; KAZMAN, 2012). No entanto, no contexto de SoS, a arquitetura dinâmica, presente neste tipo de sistema, complica essa previsão, uma vez que é difícil garantir propriedades de qualidade para todos os sistemas constituintes (BIANCHI; SANTOS; FELIZARDO, 2015).

A arquitetura deve ser avaliada pela sua capacidade de entregar os atributos de qualidade importantes do sistema. Isso deve acontecer no início do ciclo de vida, quando os

benefícios são maiores (BASS; CLEMENTS; KAZMAN, 2012) e antes das consequências de um erro arquitetural serem propagadas. Alguns atributos de qualidade já foram também revisados quanto à sua definição para o contexto de SoS. Embora hajam algumas definições bem estabelecidas para cada atributo de qualidade, eles não podem ser completamente aplicados para o contexto de SoS (BIANCHI; SANTOS; FELIZARDO, 2015). Por exemplo, a confiabilidade do SoS é um conceito mais amplo e flexível que deve ser levado em conta, devido a natureza flexível e dinâmica do SoS (Garro; Tundis, 2015). Como também, o comportamento emergente do SoS torna difícil capturar e avaliar a interoperabilidade no nível dos sistemas constituintes (Meilich, 2006).

Para avaliar atributos de qualidade em arquiteturas de software, utiliza-se a documentação arquitetural, geralmente especificada textualmente e com base em diagramas UML e SysML em aderência às 4+1 visões de Kruchten. UML e SysML são notações recorrentes para documentação de visões arquiteturais. Dentre estes diagramas, diagramas de sequência e atividades são muito utilizados para representar aspectos dinâmicos de sistemas. Entretanto, existem alguns problemas quando trata-se de SoS: (i) diagramas de comportamento da UML e SysML permitem a representação de apenas uma única configuração arquitetural dentre as múltiplas que um SoS pode assumir devido à arquitetura dinâmica, o que demandaria outros modelos, um para cada configuração arquitetural, (ii) a *execução* do modelo fica a cargo do arquiteto, que tem que realizá-la mentalmente e/ou manualmente, e (iii) a verificação de corretude destes modelos dinâmicos torna-se tanto mais difícil e propensa a erros quanto maior for o porte do sistema sendo modelado, isto é, um SoS com cinco constituintes permite fácil visualização, mas um de 300 constituintes pode oferecer dificuldades para execução manual. É importante salientar que existem versões executáveis de UML e SysML (Wang; Dagli, 2008; WOLNY et al., 2019; WOLNY, 2017). Entretanto, elas também não lidam com arquiteturas dinâmicas e cada modelo representa um único sistema, não múltiplos como é o caso de SoS.

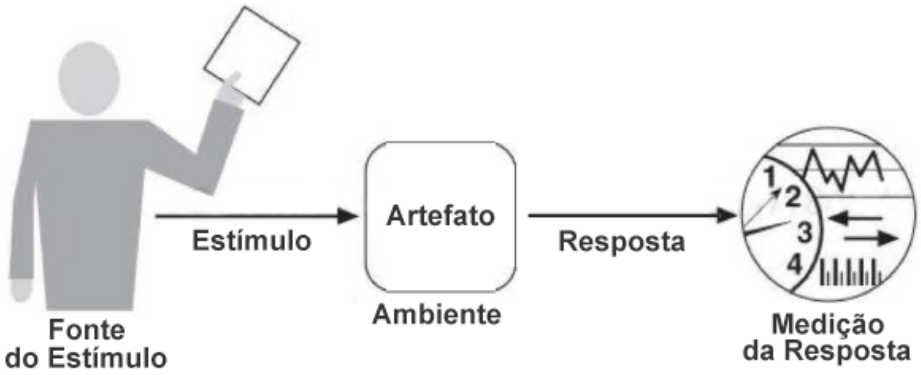
Segundo (GAGLIARDI; WOOD; KLEIN, 2009), utilizar uma abordagem que foca em atributos de qualidade de SoS pode produzir benefícios como:

- Arquitetura de SoS aperfeiçoada.
- Identificação prévia de desafios e riscos arquiteturais significantes.
- Melhor comunicação no SoS, sistema, e *stakeholders* de software.
- Integração de sistemas componentes mais previsível.
- Análise de causa base mais efetiva de áreas problemáticas.

GAPS (PONTOS EM ABERTO). Modelos de simulação convencionais possuem apenas representações do sistema, tais como suas partes e como estão ligadas. Logo, a

representação de atributos de qualidade demanda criação de outras entidades artificiais no modelo de simulação que possam reproduzir as condições que permitem visualizar cada atributo. Cada modelo de simulação de SoS representa, em essência, seus sistemas constituintes, tipos de dados, objetos e mensagens trafegadas, os links de interoperabilidade entre eles, e o arranjo arquitetural. Logo, é necessário prover uma representação artificial no modelo de simulação que seja capaz de emular as condições e estímulos que permitam prever como o SoS reagirá a adversidades; ou seja, considerando os elementos necessários para modelagem de atributos de qualidade em modelos arquiteturais expostos na Figura 1, os elementos inerentes ao modelo de simulação correspondem apenas à representação do artefato. Logo, todo o restante do aparato técnico para SoS precisa ser desenvolvido para cada AQ (Atributo de Qualidade) que se quiser representar em um modelo de simulação. Em particular, no que tange ao AQ disponibilidade, os estudos de simulação de SoS que investigam disponibilidade geralmente mantém enfoque na representação de um tipo isolado de falha (omissão). Entretanto, dada a criticidade dos domínios apoiados por SoS, é essencial que todos os tipos de falhas sejam avaliados, o que representa outro ponto em aberto que endossa e motiva a pesquisa realizada.

Figura 1 – As partes do cenário de atributos de qualidade



Fonte: (BASS; CLEMENTS; KAZMAN, 2012).

Problemas. Garantir qualidade no SoS e garantir precisão na documentação arquitetural do SoS.

Solução planejada. Especificar uma visão arquitetural da arquitetura de software do SoS usando formalismo de simulação e propor e implementar o aparato teórico necessário para o AQ disponibilidade.

1.3 Metodologia de Pesquisa

Esta seção discute a metodologia científica utilizada para investigar os problemas relatados. Optou-se por utilizar uma abordagem de ensaios experimentais com base nas diretrizes de França e Travassos (FRANÇA; TRAVASSOS, 2016). Também foi adotada a técnica Objetivo-Questão-Métrica ou GQM (Do inglês Goal-Question-Metric) (BASILI, 1992), como prescrito por França and Travassos (FRANÇA; TRAVASSOS, 2016) para elaborar o objetivo deste estudo e para derivar as questões de pesquisa, como pode ser visto abaixo.

1.3.1 Objetivos Gerais e Específicos

Para a condução desta pesquisa, o seguinte objetivo geral foi definido em conformidade com as diretrizes de (FRANÇA; TRAVASSOS, 2016).

Objetivo Geral. O objetivo deste estudo é **analisar** modelos de simulação **com a finalidade de** observar e predizer, em tempo de projeto, a influência de diferentes tipos de falhas no grau de disponibilidade fornecida por um SoS **do ponto de vista de** um arquiteto de software **no contexto de** uma simulação de uma arquitetura SoS de Monitoramento de Inundação.

Considerando o objetivo geral estabelecido, este trabalho de conclusão de curso propõe um mecanismo de representação do atributo de qualidade *disponibilidade* em modelos de simulação de arquitetura de software de SoS. Para atingir tal objetivo, foi também necessário desenvolver um modelo de simulação de falhas para SoS. Com isso, a intenção é permitir uma avaliação mais precisa de arquiteturas de software de SoS através de simulações que reproduzam de modo mais fidedigno as condições sob as quais um SoS é submetido. A partir disso, a intenção é obter um maior de grau de confiança nos resultados obtidos ainda em tempo de projeto de modo a subsidiar as decisões arquiteturais tomadas, contribuindo como consequência para uma maior qualidade do SoS.

Objetivos específicos. Os objetivos específicos definidos são:

- Refletir sobre o conceito de *disponibilidade* em SoS;
- Prover um mecanismo de falhas que permita estudar o efeito dos diferentes tipos de falhas na disponibilidade de um SoS;
- Adaptar, estender e elaborar um mecanismo de representação para o atributo *disponibilidade* em simulações de SoS colaborativo;

1.3.2 Questões de Pesquisa

Considerando o contexto e os objetivos apresentados, definiu-se as seguintes questões de pesquisa (QP) para este trabalho:

i *Como modelar e simular falhas em arquiteturas de SoS?*

ii *Como avaliar disponibilidade de SoS utilizando modelos de simulação de arquitetura de SoS?*

Para responder às duas questões de pesquisa, dois estudos diferentes foram conduzidos. No primeiro estudo, o autor desenvolveu um modelo de representação de diversas classes de falhas em modelos de simulação de arquiteturas de software de SoS. No segundo estudo, o modelo de falhas desenvolvido no primeiro foi utilizado para aferir o grau de disponibilidade entregue por um SoS de acordo com cada tipo de falha modelado no passo anterior. É importante salientar também que avaliações independentes foram conduzidas para cada um dos estudos. Assim, os estudos e seus respectivos resultados são apresentados em dois capítulos diferentes (Capítulo 4 e Capítulo 5).

Em alinhamento com a técnica GQM, este capítulo apresenta os objetivos e questões. Por sua vez, as métricas foram elaboradas para cada um dos estudos específicos conduzidos no escopo deste trabalho e serão apresentadas nos respectivos capítulos.

1.4 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma: O capítulo 2 apresenta conceitos e definições para a compreensão do escopo do trabalho; O capítulo 3 apresenta o modelo de simulação de arquitetura de SoS utilizado como base para os experimentos dos capítulos seguintes; O capítulo 4 apresenta e avalia um modelo de simulação de falhas para SoS; No capítulo 5 o modelo de falhas é utilizado para a proposta de um modelo de disponibilidade para SoS; O capítulo 6 conclui o trabalho, apresentando as contribuições e trabalhos futuros.

Parte II

Fundamentação Teórica

2 Conceitos Essenciais

Sistemas-de-Sistemas. Um Sistema-de-Sistemas é um conjunto de subsistemas existentes e heterogêneos reunidos para alcançar uma missão global que um sistema sozinho não consegue alcançar, enquanto mantém a independência operacional e gerencial (autonomia) de cada subsistema (BILLAUD; DACLIN; CHAPURLAT, 2015). SoS surgiram inicialmente no domínio militar focados na integração e interoperabilidade entre sistemas para C4I (Comando, Controle, Comunicação, Computadores e Inteligência), ISR (Inteligência, Vigilância e Reconhecimento) e campo de batalha (NIELSEN et al., 2015). Entretanto, com o advento das cidades inteligentes, estes sistemas têm migrado para o domínio civil (Graciano Neto; SANTOS; ARAUJO, 2017). Surgem sistemas como o sistema de monitoramento urbano de enchentes (Graciano Neto et al., 2016), telemedicina (PETCU; PETRESCU, 2010), negócio eletrônico (Do inglês *e-business*) (ZHU; STAPLES; JEFFERY, 2008), dentre outros sistemas de gerenciamento de crises e emergências.

No entanto, SoS diferenciam-se de outros tipos de sistemas complexos e de larga escala devido a características inerentes a esta classe de sistemas (MAIER, 1998). Tais características tornam SoS diferentes de sistemas tradicionais e torna necessária a mudança de paradigma para o projeto e desenvolvimento desses sistemas (MENDES, 2018).

Além disso, SoS estão preocupados em cumprir missões, ou seja, um conjunto de objetivos a serem realizados (i) executando tarefas baseadas em capacidades (funcionalidades) do sistema constituinte, e (ii) interações entre sistemas constituintes levando a comportamentos emergentes. Missões individuais são realizadas pelos próprios sistemas constituintes, enquanto as missões globais de um SoS são cumpridas através dos comportamentos emergentes (Graciano Neto et al., 2018). As missões globais não podem ser realizadas considerando a contribuição de apenas um único sistema constituinte.

Características: Para um sistema ser considerado um SoS, este deve se enquadrar nas características abaixo. Segundo (NAKAGAWA et al., 2013), essas características apresentadas inicialmente por (MAIER, 1998), ainda são utilizadas pela maioria da comunidade, portanto foi utilizada com algumas contribuições de outros autores.

(i) Independência Operacional dos Constituintes: Um constituinte deve estar habilitado a continuar a operar funcional e independentemente, satisfazendo suas próprias missões, mesmo quando não estiverem cooperando com o SoS ou se o SoS for desmontado. Visto que, os constituintes operam de forma autônoma, contribuindo ocasionalmente e oportunamente para cumprir a missão no contexto SoS (MAIER, 1998; NIELSEN et al., 2015; Graciano Neto; SANTOS; ARAUJO, 2017; MENDES, 2018).

(ii) Independência Gerencial dos Constituintes: Sistemas Constituintes são

tanto desenvolvidos quanto adquiridos, mantidos e integrados por uma diversidade de *stakeholders* e organizações distintas (MAIER, 1998; Graciano Neto; SANTOS; ARAUJO, 2017).

(iii) Comportamento Emergente: Através da interoperabilidade entre constituintes em um SoS, uma sinergia é alcançada, surgindo assim uma funcionalidade mais complexa, planejada para alcançar uma ou mais missões ou surgindo de modo não previsto devido à dinâmica do SoS. Tal funcionalidade não pode ser alcançada por, ou atribuída a, nenhum dos sistemas individuais (Graciano Neto; SANTOS; ARAUJO, 2017; NIELSEN et al., 2015).

(iv) Desenvolvimento evolucionário: Os constituintes do SoS podem ser adicionados, modificados ou removidos a qualquer momento. Logo seus objetivos e funcionalidades podem estar em constantes mudanças, sempre evoluindo. Se os constituintes do SoS evoluem, como resultado o SoS como um todo também evolui (NIELSEN et al., 2015; Graciano Neto; SANTOS; ARAUJO, 2017).

(v) Distribuição: Os constituintes estão distribuídos em uma grande extensão geográfica, sendo fisicamente desacoplados e trocam apenas informação entre eles, já que sua comunicação é baseada na tecnologia de rede (Graciano Neto; SANTOS; ARAUJO, 2017; MENDES, 2018; NIELSEN et al., 2015).

Tipos de SoS: Abaixo são listados os principais tipos de SoS propostos por Maier em 1998. Alguns autores contestam a existência do SoS Virtual, afirmando que este seria apenas uma variação um pouco mais desacoplada do tipo colaborativo. Ainda não há consenso, mas Maier propõe uma nova taxonomia (Graciano Neto; SANTOS; ARAUJO, 2017; MAIER, 2015). Os tipos de SoS podem ser vistos abaixo:

(i) Dirigido: Possuem uma autoridade central, sendo construídos e gerenciados de forma centralizada para objetivos específicos (MAIER, 1998). O centralizador pode ser uma entidade de software ou um grupo de gestores. As ações dos constituintes são dirigidas pela entidade centralizadora, sendo estes subordinados ao objetivo gerenciado central (Graciano Neto; SANTOS; ARAUJO, 2017). Portanto, os constituintes mantêm a capacidade de operarem de forma independente, mas o seu modo de operação normal é gerenciado para satisfazer a um propósito concreto (NIELSEN et al., 2015).

(ii) Reconhecido: Uma autoridade central existe para possibilitar a comunicação entre os constituintes. Os sistemas se comunicam de forma bidirecional com a entidade, e esta é consultada para que os constituintes descubram novos sistemas disponíveis e serviços prestados (Graciano Neto; SANTOS; ARAUJO, 2017). Os sistemas constituintes mantêm sua independência de propriedades e objetivos. Além disso, os objetivos, o gerenciamento e os recursos são reconhecidos por todas as partes do SoS e o gerenciamento deste é baseado na negociação entre estas partes (MENDES, 2018). Sendo assim, os constituintes

se comunicam para se organizarem e cumprirem uma ou mais missões como resultado.

(iii) Colaborativo: Neste tipo de SoS não há uma entidade central que possua poder coercitivo para executar o sistema. Os sistemas constituintes colaboram voluntariamente para cumprir a missão global do SoS (MAIER, 1998). A cooperação entre os sistemas constituintes acontece de forma espontânea, mas com uma intenção prevista de interação dentro de um propósito pré-definido e embutido. (Graciano Neto; SANTOS; ARAUJO, 2017). Assim sendo, os sistemas decidem coletivamente a forma como irão cumprir e manter os padrões do SoS.

(iv) Virtual: Aqui o SoS não possui um gerenciamento central e nem um propósito em comum. O que torna o seu comportamento e os objetivos altamente emergentes, como também torna as estruturas e meios que produzem as funcionalidades do sistema difíceis de discernir e distinguir (NIELSEN et al., 2015). O SoS é baseado em um mecanismo relativamente invisível para se manter (MAIER, 1998). Os subsistemas realizam as suas funcionalidades individuais e o produto final acaba sendo a soma dos resultados parciais, não há uma intenção clara ou colaboração explícita entre eles. (Graciano Neto; SANTOS; ARAUJO, 2017)

Arquitetura de Software. Existem diversas definições e abordagens diferentes para arquitetura de software. Barbosa fez uma revisão de diversas dessas definições e as apresenta em seu livro-texto (BARBOSA, 2009). Devido ao detalhamento e a utilização de modelos bem descritos, este trabalho utilizou a definição feita por (BASS; CLEMENTS; KAZMAN, 2012) e seguiu o seu trabalho também como base para atributos de qualidade mostrados em sua obra.

(BASS; CLEMENTS; KAZMAN, 2012) definem a arquitetura de um programa ou de sistemas computacionais como a estrutura ou estruturas do sistema, a qual é composta de elementos de software, as propriedades externamente visíveis desses elementos, e os relacionamentos entre eles. As estruturas são um conjunto de elementos mantidos juntos por uma relação. Sistemas são compostos por diversas estruturas, porém nenhuma delas é a arquitetura em si, mas uma representação desta sob uma perspectiva. As estruturas podem ser categorizadas em três tipos, sendo elas (i) *módulos*, estruturas estáticas que focam na forma como as funcionalidades do sistemas são divididas e passadas à equipe de desenvolvimento; (ii) *componente & conector* ou C&C (Do inglês Component-and-Connector), estrutura dinâmica, a qual foca na forma em que os elementos interagem entre si em tempo de execução para executar as funções do sistema; (iii) *alocação*, tipo de estrutura que descreve o mapeamento de estruturas de software para os ambientes de organização, desenvolvimento, instalação e execução do sistema, nesse contexto os componentes são implantados em hardware para serem executados. Além dessas estruturas, softwares tem outros incontáveis tipos de estruturas, mas nem todas são arquiteturais. Uma estrutura é arquitetural se possibilita pensar sobre o sistema ou uma propriedade deste.

Este raciocínio deve ser sobre um atributo do sistema que é importante ao *stakeholder*.

Ademais, sistemas modernos são frequentemente muito complexos para serem compreendidos de uma vez só. As diferentes perspectivas de uma arquitetura auxiliam neste processo. Restringimos nossa atenção para um grupo menor de estruturas de software do sistema, deixando bem claro qual estrutura ou estruturas estão sendo discutidas no momento. A abstração da arquitetura nos permite olhar para o sistema em termos de seus elementos, como eles estão organizados, como eles interagem, como são compostos, quais propriedades do sistema são as que dão suporte a pensar sobre este, dentre outras. A abstração é essencial para controlar a complexidade de um sistema, uma vez que é difícil para o ser humano lidar com toda a complexidade de uma só vez (BASS; CLEMENTS; KAZMAN, 2012).

Consequentemente, a arquitetura de software executa um papel importante na determinação da qualidade do sistema, visto que ela forma a base para qualquer sistema de software-intensivo bem sucedido. Como também a arquitetura de software além de trazer qualidade ao sistema, também o faz de forma previsível. Para o contexto deste trabalho, Nakagawa et. al. afirmam que dar atenção a arquitetura de software de SoS também é com certeza fundamental para o sucesso de tal sistema (NAKAGAWA et al., 2013). Se soubermos que certos tipos de decisões arquiteturais levam a certos atributos de qualidade em um sistema, então podemos fazer essas decisões e esperar que sejamos recompensados com os atributos de qualidade associados. Além disso, quando examinamos uma arquitetura, podemos observar se as decisões foram tomadas e prever de forma precisa se a arquitetura exibirá as qualidades associadas (BASS; CLEMENTS; KAZMAN, 2012).

No entanto, a arquitetura por si só não consegue garantir uma funcionalidade ou atributo de qualidade, as decisões tomadas durante todo o ciclo de vida do software afetam a qualidade do sistema e determinam se o sistema terá uma arquitetura adequada. Nenhum atributo de qualidade é totalmente dependente do design e nem da implementação ou implantação (BASS; CLEMENTS; KAZMAN, 2012).

Atributos de qualidade. Para (BASS; CLEMENTS; KAZMAN, 2012), um atributo de qualidade ou AQ (Do inglês Quality Attribute) é uma propriedade mensurável e testável de um sistema que é utilizada para indicar o quão bem o sistema satisfaz as necessidades dos seus *stakeholders*. Os atributos de qualidade clássicos da literatura são (i) disponibilidade, (ii) interoperabilidade, (iii) modificabilidade, (iv) performance, (v) segurança, (vi) testabilidade e (vii) usabilidade (ISO, 2011). Além destes, para cada área de software existem atributos de qualidade específicos. (NAKAGAWA et al., 2013) em sua revisão bibliográfica de SoS, citam os atributos de qualidade mais abordados em publicações voltadas para SoS. Os atributos de qualidade mais citados nas publicações revisadas foram (i) interoperabilidade, aparecendo em 65% delas, (ii) adaptabilidade/flexibilidade, com 30%, (iii) segurança/confidencialidade, 28%. O atributo Disponibilidade, que será abordado

mais profundamente neste trabalho, aparece apenas na 10ª posição, com 8% de aparição nas publicações, evidenciando a carência de publicações e a necessidade de um estudo mais detalhado deste atributo de qualidade. Além disso, (BIANCHI; SANTOS; FELIZARDO, 2015) realizaram uma análise sobre a cobertura ISO/IEC 25010 dos atributos de qualidade de SoS, o estudo indicou que 48% dos atributos de qualidade comumente considerados no SoS não são abordados por este modelo de qualidade.

A especificação de um atributo de qualidade deve ser feita de forma não-ambígua e testável. Para tal, (BASS; CLEMENTS; KAZMAN, 2012) utilizam as seguintes partes como base dos atributos de qualidade:

i **Fonte do estímulo**. É um tipo de entidade (Um humano, um sistema computacional ou qualquer outro atuador) que gera o estímulo.

ii **Estímulo**. O estímulo é uma condição que necessita de uma resposta quando chega no sistema.

iii **Ambiente**. O estímulo ocorre sob certas condições. O sistema pode estar em condição de sobrecarga ou em operação normal, ou em algum outro estado relevante. Para vários sistemas, a operação "normal" pode se referir a um de vários modos diferentes. Para estes tipos de sistema, o ambiente deve especificar em qual modo o sistema está executando.

iv **Artefato**. Algum artefato é estimulado. Pode ser uma coleção de sistemas, o sistema inteiro, alguma parte ou partes dele.

v **Resposta**. A resposta é a atividade tida como o resultado da chegada de um estímulo.

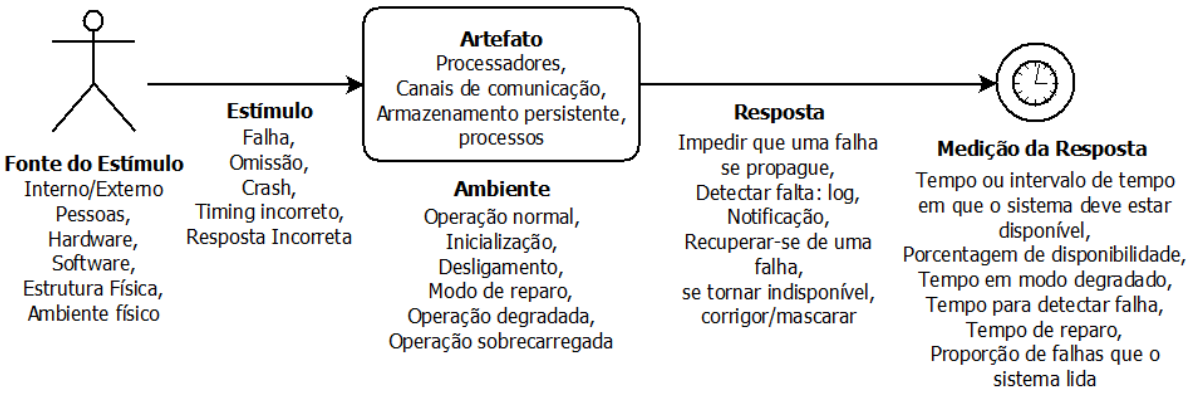
vi **Medição da resposta**. Quando a resposta acontece, ela deve ser medida para que o requisito possa ser testado.

As partes do cenário de atributo de qualidade também podem ser vistas na Figura 1, além disso é possível ver uma instância para o atributo Disponibilidade na Figura 2.

Disponibilidade A disponibilidade pode ser influenciada por quatro classes de falhas: (i) omissão, na qual um componente é solicitado e não responde, (ii) *crash*, que acontece quando um componente sofre múltiplas omissões repetidamente, (iii) *timing*, quando um componente responde muito cedo ou muito tarde e (iv) (BASS; CLEMENTS; KAZMAN, 2012). Apesar da definição e realização da disponibilidade ser amplamente conhecida para sistemas individuais, ela pode ser significativamente impactada no contexto de um SoS, uma vez que a entrega de um determinado conjunto de comportamentos depende da presença dos constituintes que são capazes de oferecer as funcionalidades necessárias quando o SoS

é necessário para entregá-los. Portanto, é necessário estudar como a presença ou ausência de um ou mais constituintes em um SoS pode afetar a disponibilidade de todo o SoS. E, considerando que a indisponibilidade de um SoS em um domínio crítico pode significar mortes e danos para usuários que dependem de seus serviços, é importante prever, ainda em tempo de projeto, o grau com que um SoS arquitetado pode estar disponível quando requerido/solicitado. No contexto de SoS, *disponibilidade* é particularmente importante. A ISO define disponibilidade como sendo "o grau em que um sistema, produto ou componente está operacional e acessível quando necessário para uso" (ISO, 2011).

Figura 2 – Cenário geral para Disponibilidade.



Fonte: Autoria própria, baseado em (BASS; CLEMENTS; KAZMAN, 2012).

Para o cenário geral de *Disponibilidade*, visualizado na Figura 2, a *Fonte do Estímulo* são todos os elementos internos e externos que podem ser as origens da falha.; Os *Estímulos* são as quatro classes de falha: (i) omissão, (ii) *crash*, (iii) *timing* e (iv) *response*; O *Artefato* é o recurso que deve estar altamente disponível; O *Ambiente* é o estado em que o ambiente se encontra no momento da falha, visto que este pode influenciar na resposta; A *Resposta* pode se dar de diversas formas, as principais são (i) prevenir que a falha se propague, (ii) detectar a falha e (iii) recuperar-se da falha; A *Medição da Resposta* pode especificar (i) uma porcentagem de disponibilidade; (ii) pode especificar um tempo para detectar a falha; (iii) tempo para reparar a falha, (iv) tempos ou intervalos de tempo durante os quais o sistema deve estar disponível ou (v) a duração pela qual o sistema deve estar disponível (BASS; CLEMENTS; KAZMAN, 2012).

Arquitetura de SoS. Enquanto a definição clássica de uma arquitetura de software compreende uma tupla *elementos de software, relações entre eles, e propriedades de ambos* (PERRY; WOLF, 1992; BASS; CLEMENTS; KAZMAN, 2012), a arquitetura de software de um SoS inclui seus constituintes, seus links de interoperabilidade, e as decisões e fundamentos compartilhados que acionam o seu edifício (NIELSEN et al., 2015). O estudo

da arquitetura de software do SoS aborda, então, o estudo das propriedades funcionais e não funcionais percebidas a partir do software dos constituintes. Neste contexto, modelos dinâmicos são então necessários para prever, em tempo de projeto, como vários constituintes independentes podem afetar a disponibilidade de SoS em tempo de execução.

Simulações. As simulações compreendem uma imitação aproximada da operação de um sistema que pode representar as principais características do sistema, incluindo seu comportamento, funções e propriedades físicas ou abstratas. Simulações requerem modelos, que representam abstratamente o próprio sistema, enquanto a simulação representa sua operação ao longo do tempo (BANKS et al., 2000; GRAY; RUMPE, 2016). Modelagem e Simulação (M&S) é na verdade um padrão *de-facto* para outras engenharias como meio de prototipar e prever propriedades de sistemas, como engenharia civil e de sistemas. Particularmente, o M&S já é considerado uma plataforma adequada para obter evidências empíricas em estudos de engenharia de software (FRANÇA; TRAVASSOS, 2016). Entre os possíveis formalismos para simulação (Arquitetura de Alto Nível, modelagem baseada em agentes, cadeias de Markov), o DEVS foi adotado com sucesso para simular SoS (ZEIGLER; SARJOUGHIAN, 2013; ZEIGLER; SARJOUGHIAN, 2017).

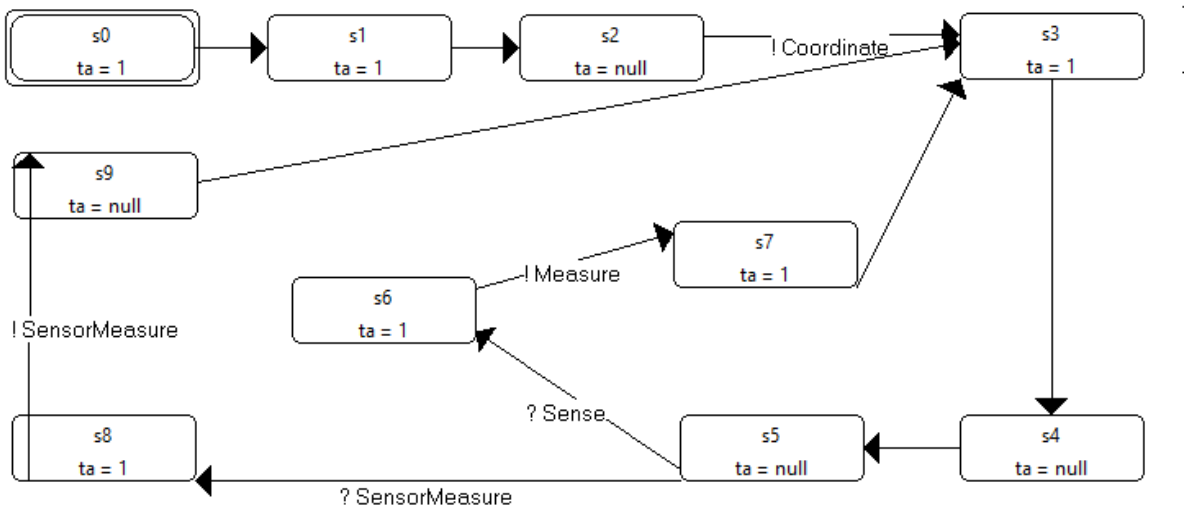
DEVS. *Discrete-Event Systems Specification* (DEVS) é um formalismo de modelagem de simulação que fornece a base para a simulação de SoS em ambiente virtual (ZEIGLER; SARJOUGHIAN, 2017). DEVS é estruturado através de modelos atômicos, utilizados para representar o comportamento dos sistemas, e de modelos acoplados, estes para representar a estrutura arquitetural do sistema. Além da linguagem DEVS, a plataforma de simulação MS4 Me[1] também utiliza da SES para definir as conexões entre os modelos atômicos e construir o modelo acoplado (TENDELOO; VANGHELUWE, 2017). A ferramenta MS4 Me é constituída de um ambiente de modelagem e simulação DEVS escrito em Java e baseado no framework Eclipse e é gratuita por um ano após a instalação.

Os modelos atômicos DEVS são constituídos basicamente de (i) *Estados* e (ii) *Transições*. Os estados podem ser (i) *Estados Passivos*, (ii) *Estados de Espera* ou (iii) *Estados Iniciais*. Já as transições são divididas em (i) *Transições Internas* ou (ii) *Transições Externas*. Além disso, as transições internas podem produzir uma *Saída*. Os estados podem ter uma ou mais transações externas, e o tipo de *entrada* definirá a transição que será efetivada. Um exemplo de um diagrama de estados do modelo atômico de um sensor pode ser visto na Figura 3. No diagrama de estados do MS4 Me as portas de entrada são representadas por "?"enquanto as portas saídas por "!". As portas de entrada e saída acompanham os símbolos "?"e "!", portanto "! Coordinate"representa uma saída na porta "Coordinate". O estado inicial possui uma borda duplicada. Ademais, as transições internas possuem um tempo de permanência no estado, na figura representados por "ta", enquanto as transições externas acontecem de acordo com a mensagem recebida. O MS4 Me permite

[1] http://www.ms4systems.com/pages/ms4me.php

ainda a inserção de código Java nas transições, tornando este tipo de simulação ainda mais poderosa, visto que traz todo o poder de uma linguagem de programação como forma de auxílio para a simulação.

Figura 3 – Diagrama de estados do modelo atômico de um sensor no Editor do MS4 Me.

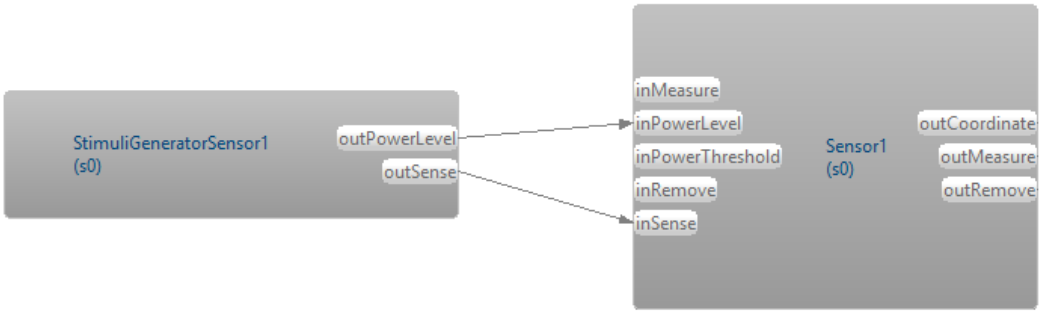


Fonte: Autoria própria.

SES. *System Entity Structure* (SES) é um framework de ontologia de alto nível voltado para modelagem, simulação, projeto de sistemas e engenharia (ZEIGLER; SARJOUGHIAN, 2017). No SES são definidos: (i) *aspectos*, as estruturas hierárquicas do sistema; (ii) *especializações* que possam ocorrer; (iii) *acoplagens* conexões das portas de entrada e saída de cada sistema e como estes interagem na simulação; (iv) *similaridades* que podem ser utilizadas para indicar que um sistema é similar a outro de alguma forma; e (v) *variáveis* que podem afetar o comportamento do sistema. Durante a execução da simulação, as mensagens são enviadas de um sistema a outro baseado no que foi definido na SES da simulação. Uma saída gerada por um sistema será recebida por todos os sistemas que estão conectados via SES. Um exemplo de modelo acoplado simples pode ser visto na Figura 4, em que há um gerador de estímulos provendo dados a um sensor.

Para simular disponibilidade em SoS, é necessário simular falhas. Entretanto, durante uma busca na literatura, não encontramos trabalhos que proveem soluções extensivas para falhas, lidando com tipos específicos de falhas, que podem trazer ameaças às conclusões obtidas via simulação. Logo, antes de aferir disponibilidade de uma arquitetura de SoS, foi necessário elaborar um modelo de malhas robusto e abrangente, que será discutido no Capítulo 4. A próxima seção discute os trabalhos correlatos

Figura 4 – Exemplo de modelo acoplado na simulação do MS4 Me.



Fonte: Autoria própria.

2.1 Trabalhos Correlatos

A ISO 25010 inclui 'disponibilidade' como um subatributo de *confiabilidade*. Estudos na literatura lidam com confiabilidade e/ou seus atributos correlatos no contexto de SoS. Contudo, ainda não existem muitos estudos investigando a fundo este tópico. Uma pesquisa piloto foi realizada no ACM DL[2] buscando no título, abstract e palavras-chave, e IEEE Xplore[3] utilizando as palavras-chave `Availability E Systems-of-Systems`. O primeiro retornou apenas 11 estudos, e o último retornou 23. Portanto, isso mostra que a área ainda precisa de investigação[4]. Entre os estudos retornados, alguns deles mostram resultados em campos correlatos. Por exemplo, Eusgeld e Dietz usam uma abordagem de SoS para investigar interdependências entre sistemas de controle industrial e suas infraestruturas críticas (CI, Do inglês Critical Infrastructures) subjacentes (EUSGELD; NAN; DIETZ, 2011). Eles adotam a Arquitetura de Alto Nível (HLA, Do inglês High Level Architecture) para modelar a arquitetura para modelagem e simulação de seu sistema. Eles também investigam a propagação de falhas em sua arquitetura. No entanto, não há menção à arquitetura de software e eles investigam o impacto de falhas na arquitetura, mas não há um foco claro em um atributo de qualidade específico. Por sua vez, Johnson e colegas introduzem o metamodelo MAPL (Multi-Attribute Prediction Language), o qual auxilia os arquitetos a analisar as qualidades não-funcionais de SoS (Johnson et al., 2016). Eles trabalham em um conjunto de pontos de vista arquiteturais e analisam quantitativamente um SoS usando o padrão ArchiMate associado aos formalismos UML e OCL. No entanto, a arquitetura dinâmica não é abordada, o que pode afetar o grau de disponibilidade entregue.

(Walker; Tummala; McEachen, 2010) analisaram a disponibilidade usando uma

[2] http://dl.acm.org/
[3] https://ieeexplore.ieee.org/Xplore/home.jsp
[4] Pesquisa realizada em Junho de 2019

abordagem de SoS para sistemas espaciais. Eles analisam o impacto de falhas e de sistemas ficando *offline*, mas não há uma análise da disponibilidade de todo o sistema, mas dos constituintes individualmente. Ford et al. também investigam a disponibilidade, mas em um conjunto de sistemas distribuídos (não são sistemas independentes) (FRANÇA; TRAVASSOS, 2012). Portanto, não há um foco em como uma falha de componentes individuais afeta os comportamentos globais que estão sendo entregues de forma coordenada. Durante a pesquisa piloto, não foram encontrados estudos que abordassem a disponibilidade em arquiteturas de software de SoS, investigando o impacto da arquitetura dinâmica no comportamento global que está sendo entregue.

Por fim, (BOGADO; GONNET; LEONE, 2014) apresenta um modelo de simulação reutilizável e escalável, desenvolvido com DEVS para testar atributos de qualidade. Este trabalho foca em performance, confiabilidade e disponibilidade. Contudo, o modelo foi desenvolvido para sistemas individuais que não interoperam e não dão suporte a SoS. Além disso, a disponibilidade é avaliada apenas por omissão, enquanto existem 3 outros tipos de falha em disponibilidade.

Parte III

Contexto de Motivação da Proposta

3 SoS de Monitoramento de Enchentes

Para ilustrar as necessidades que motivam a elaboração da proposta apresentada, neste capítulo é apresentado um exemplo concreto de domínio que foi utilizado nesta pesquisa (o SoS de Monitoramento de Enchentes), evidenciando porque são necessários (i) o estabelecimento de um modelo de falhas específico e adaptado para SoS, e (ii) o uso de modelos de simulação e de modelagem de atributos de qualidade em tempo de projeto para subsidiar no futuro uma abordagem de predição de disponibilidade e seu impacto no comportamento de um SoS.

Contextualização. Os estudos foram realizados utilizando um SoS de monitoramento de enchentes (FMSoS, do inglês Flood Monitoring SoS) como domínio. O FMSoS emite alertas aos cidadãos e as autoridades sobre inundações urbanas que podem ameaçar sua integridade, e ao fazê-lo, eles são capazes de agir e tomar decisões para minimizar as consequências da inundação. O SoS é composto por sensores inteligentes, ou seja, sistemas independentes que interoperam para fornecer o alerta de inundação, abrangendo uma ampla área geográfica que não poderia ser coberta usando um único sistema. Os sensores capturam o nível da água e enviam os dados das medidas para os *gateways*, que possuem alarmes que são acionados quando a água atinge um certo limite. O FMSoS foi inicialmente descrito por (OQUENDO, 2016a) e foi modificado neste trabalho para gerar e reagir às falhas de disponibilidade. Uma representação do FMSoS pode ser visualizada na Figura 5, onde é mostrado o posicionamento dos sensores e o local crítico da enchente.

O FMSoS é considerado um Sistema-de-Sistemas (OQUENDO, 2016a; NETO et al., 2017), uma vez que exibe as cinco características apresentadas por (MAIER, 1998):

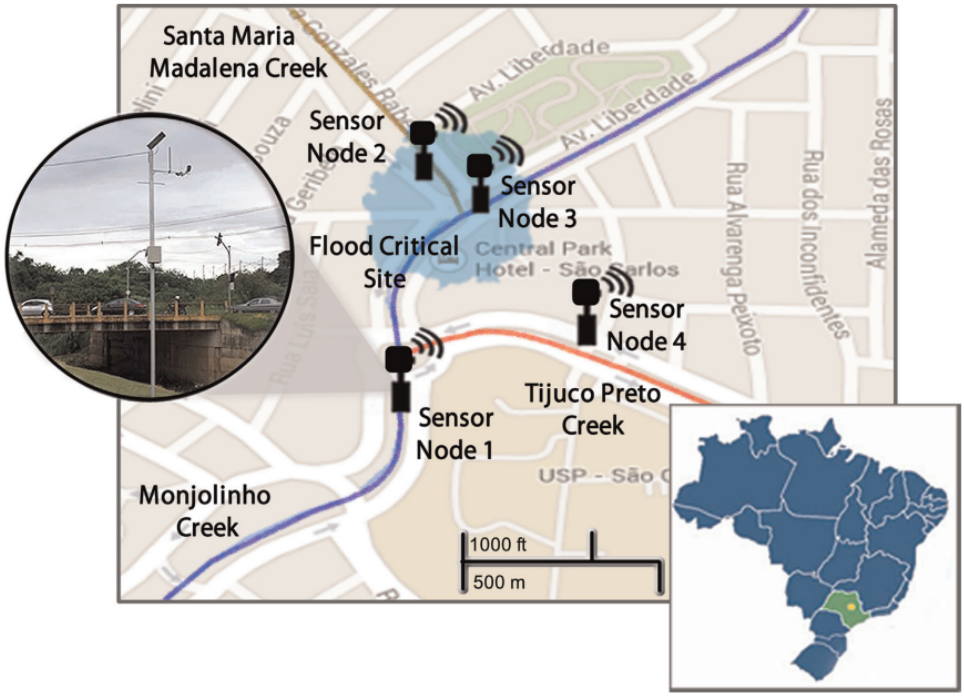
(i) Independência Operacional dos Constituintes: Cada constituinte (sensor, gateway, drone, sistema de *crowdsourcing*[1]) opera de uma forma que é independente dos outros constituintes, já que pertencem a diferentes câmaras municipais e possuem missões diferentes na região de São Carlos.

(ii) Independência Gerencial dos Constituintes: Uma diversidade de *stakeholders* e empresas podem possuir, entregar e gerenciar de maneira independente diferentes constituintes que compõem o FMSoS. Além disso, cada constituinte tem sua própria estratégia de gestão para transmissão x consumo de energia e atuará sob a autoridade dos diferentes conselhos municipais.

(iii) Comportamento Emergente: Um único constituinte não poderia fornecer

[1] O crowdsourcing é um tipo de atividade participativa on-line em que um indivíduo, uma instituição, uma organização sem fins lucrativos ou empresa propõe a um grupo de indivíduos de conhecimento, heterogeneidade e número variáveis, por meio de uma chamada aberta flexível, o empreendimento voluntário de uma tarefa.

Figura 5 – A área de estudo em São Carlos/SP.



Fonte: (HORITA et al., 2015).

um alerta de inundação de forma independente. Por exemplo, se apenas um sensor, ou sistema de crowdsourcing ou drone executa suas atividades em uma área urbana, ele pode não notificar uma inundação a tempo, não sendo eficaz. Pode emitir um falso alerta, já que a inundação poderia ser limitada a outro lugar. Assim, o alerta de inundação é resultado da interoperabilidade entre uma diversidade de constituintes que trabalham em cooperação, que se espalham ao longo da margem do rio.

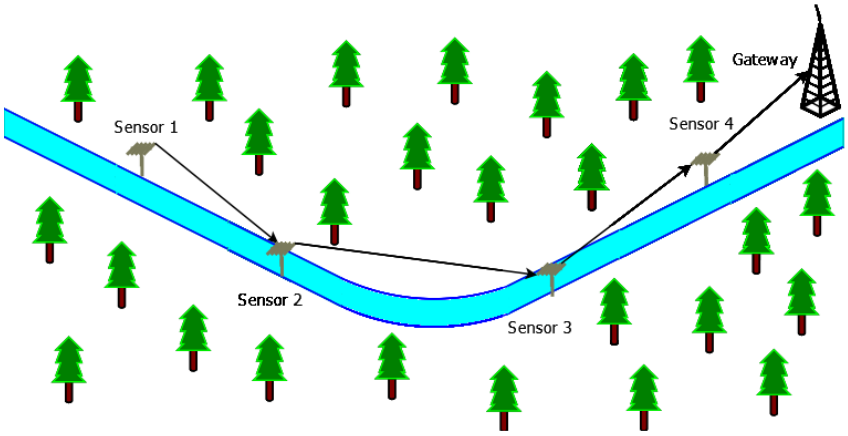
(iv) Desenvolvimento Evolucionário: O SoS evolui como consequência de alterações na configuração ou funcionalidade dos constituintes.

(v) Distribuição: Todos os constituintes interoperam através de uma rede de comunicação.

Para o escopo desta investigação preliminar, pretendeu-se entender como as falhas dos constituintes podem impactar todo o SoS. Em seguida, foi adotado um modelo no qual os sistemas constituintes são sensores que se comunicam com um *gateway*. A topologia da linha é ilustrada na Figura 6, enquanto a Figura 7 mostra a topologia de árvore. Os componentes são representados por modelos atômicos e o ambiente também é representado por modelos atômicos que fornecem os dados para alimentar a simulação.

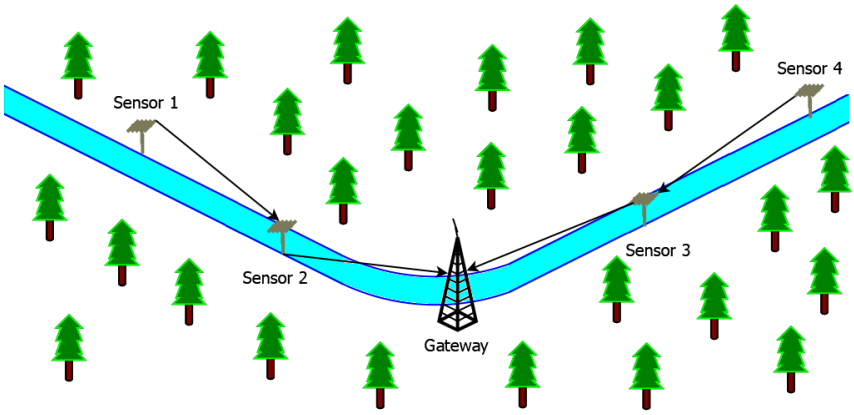
Preparação de dados. Os dados foram coletados de sensores reais atualmente implantados rio Monjolinho em São Carlos, SP, foram coletados de 23 a 25 de novembro de 2015. Foram utilizadas para os experimentos 1000 amostras desses dados, coletadas a cada 5

Figura 6 – FMSoS com a topologia linha.



Fonte: Autoria própria.

Figura 7 – FMSoS com a topologia árvore.



Fonte: Autoria própria.

minutos, correspondendo a 3,5 dias. Durante este período, ocorreu uma inundação real, o que permitirá realizar a análise de como o SoS se comportará em caso de falhas constituintes. As medições recebidas pelo *gateway* foram então armazenadas em uma planilha para posterior análise. Os dados entregues aos constituintes que serão simulados compreenderam (i) o identificador do constituinte que gerou o nível de água coletado (fonte), (ii) o tempo de simulação quando os dados foram gerados e (iii) o nível de água coletado. Os dados são providos durante a simulação através do gerador de estímulos *Stimuli-SoS* proposto por (NETO et al., 2017). Os sensores recebem as leituras dos *Stimuli-SoS*, os quais são responsáveis por simular o ambiente onde o SoS está inserido.

O próximo capítulo apresenta o modelo de falhas elaborado para SoS bem como os resultados do estudo conduzido.

Parte IV

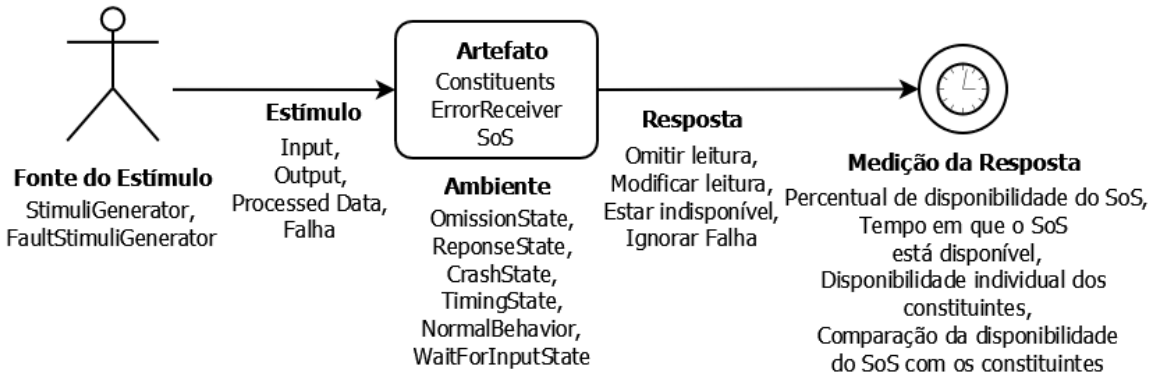
Um Modelo de Falhas para Arquiteturas de Software de Sistemas-de-Sistemas

4 Modelagem de Falhas em Sistemas-de-Sistemas

De acordo com (BASS; CLEMENTS; KAZMAN, 2012), o cenário geral para testar atributos de qualidade e suas falhas, nesse caso Disponibilidade, consiste em um *Estímulo*, uma entrada que requer uma *Resposta* do *Artefato*, no cenário de SoS são os constituintes, sendo o estímulo provido por uma *Fonte de Estímulo*, dentro de um *Ambiente*, sendo medido como uma *Medição da Resposta*. Como resultado desses requisitos, a Figura 8 foi produzida para ilustrar nosso cenário de simulação.

Portanto, para que seja possível avaliar a Disponibilidade de um sistema em uma simulação, inicialmente é necessário que a falha seja também representada no modelo de simulação. O modelo de falhas proposto aqui possibilitará a futura avaliação de Disponibilidade. Este capítulo engloba a concepção, construção e avaliação deste modelo de falhas e no cenário geral irá até a *Resposta*, considerando os elementos apresentados na Figura 8. Por sua vez, a Medição da Resposta é objeto de investigação da solução apresentada no Capítulo 5.

Figura 8 – Modelo Conceitual para Representação do Atributo de Qualidade Disponibilidade para Simulações de SoS.



Fonte: Autoria própria, adaptado de (BASS; CLEMENTS; KAZMAN, 2012).

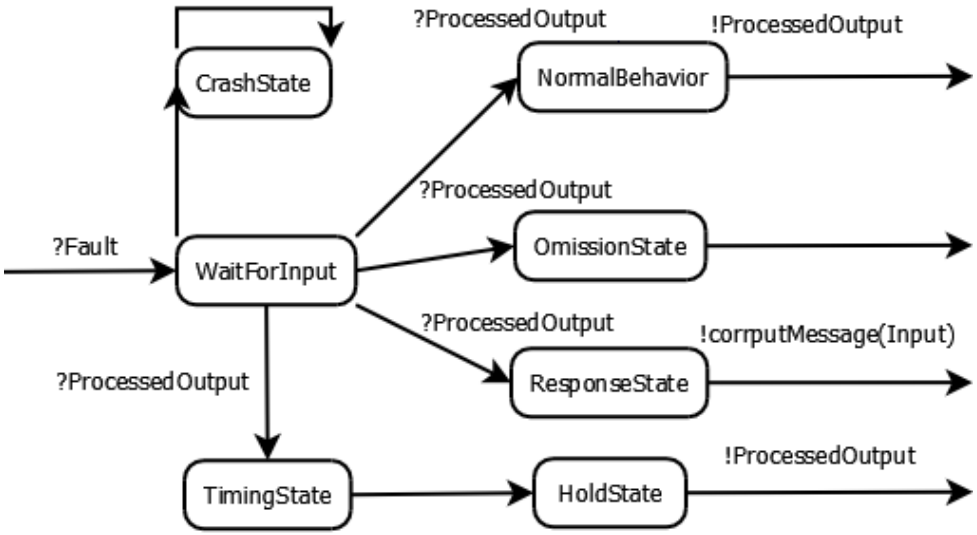
Neste projeto, um modelo de simulação que abrange quatro tipos de falhas de disponibilidade foi desenvolvido. Inicialmente foi utilizado como base o FMSoS com entidades atômicas e acopladas de simulação proposto por (OQUENDO, 2016a), sem nenhuma entidade hierárquica, a simulação reproduzia apenas o comportamento do SoS em perfeito funcionamento. A partir dessa base, foram inseridos os modelos de geração, simulação e medição de falhas. O comportamento normal do modelo de simulação

começou a sofrer de falhas de disponibilidade e, a partir dessas falhas, será possível medir a disponibilidade e perceber o impacto dessa disponibilidade ou indisponibilidade no Sistema-de-Sistemas como um todo.

4.1 Modelo Proposto

No modelo original (OQUENDO, 2016a) havia apenas constituintes (*Gateways*, *Sensors*, *Drones*, *Crowd*) como *Entidades do Mundo Real* e geradores de estímulos, que são *Entidades Artificiais*, representando o ambiente. No novo modelo, os constituintes originais foram encapsulados dentro de uma nova entidade de simulação, tornando-se uma entidade hierárquica denominada Constituinte Encapsulado (EC, do inglês *Encapsulated Constituent*). As portas de entrada e saída originais foram mantidas para permitir uma fácil modularização e reutilização do modelo DEVS.

Figura 9 – Uma parte simplificada da máquina de estados DEVS do receptor de erro (ER).

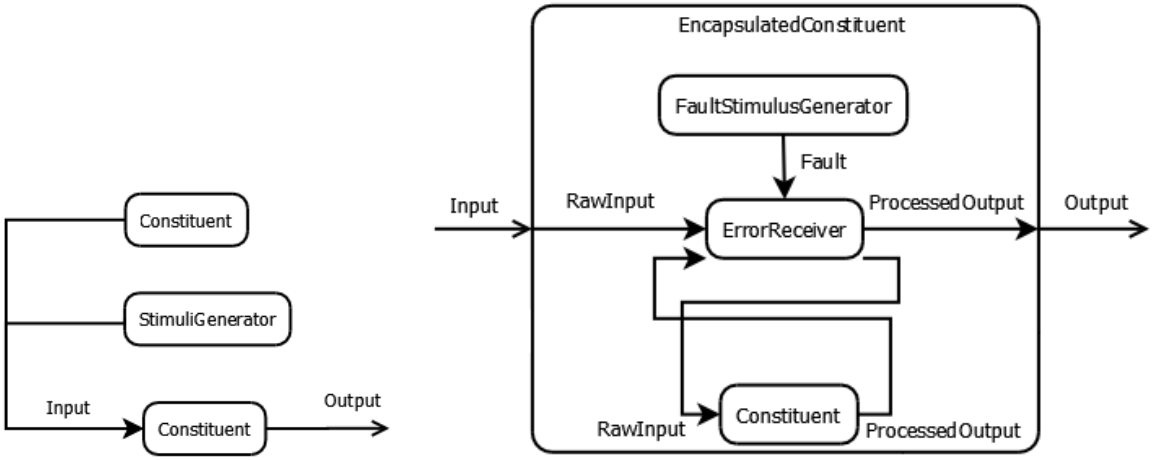


Fonte: Autoria própria.

Além disso, novas entidades artificiais foram introduzidas para gerar a falha pretendida e simular seu comportamento, como pode ser visto na Figura 10. O gerador de estímulo de falha (FSG, correspondente ao termo em inglês *Fault Stimuli Generator* visto na Figura 10) gera a falha com base na probabilidade pré-definida, e envia para o Receptor de Erro (ER, do inglês *Error Receiver*), que reproduzirá o comportamento de ocorrência daquele tipo específico de falha. Além disso, o ER é responsável pelo encapsulamento do constituinte, uma vez que recebe a entrada e controla a saída do EC. A entrada da EC pode ser produzida por um gerador de estímulo ou enviada por outra EC. Todas as entradas são enviadas para o constituinte, para serem processadas, pelo ER. O ER controlará a saída processada (Em inglês ProcessedOutput) pelo constituinte com base na falha gerada pelo

FSG. É importante ressaltar que o FSG foi baseado no gerador de estímulos Stimuli-SoS proposto por (NETO et al., 2017), simulando o ambiente onde aquele constituinte está inserido, porém, ao invés de realizar a leitura de dados coletados no mundo real, o FSG gera o estímulo da falha apenas em tempo de execução e baseado numa probabilidade pré-estabelecida.

Figura 10 – Constituinte antes e depois do encapsulamento na perspectiva SES.

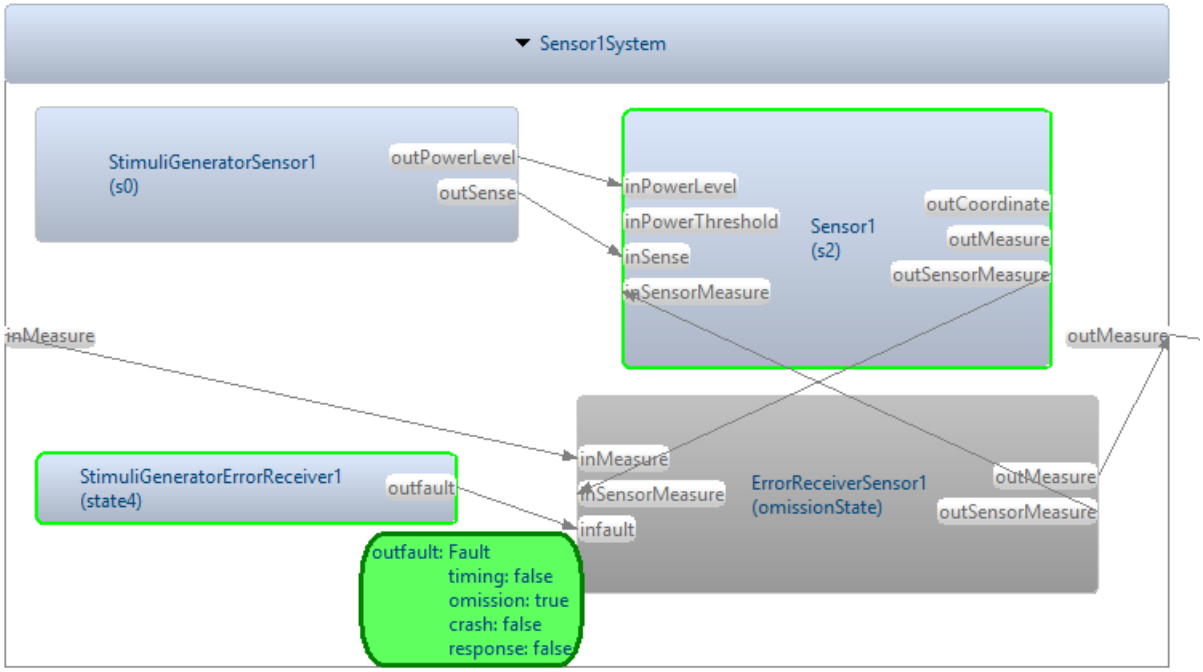


Fonte: Autoria própria.

Além disso, para reproduzir os comportamentos com falhas de disponibilidade, o ER possui estados específicos para cada classe de falha. Note que os estados na Figura 9 foram simplificados para facilitar a explicação. Para simular (i) Omissão, o ER ignora a entrada. Quando uma falha de (ii) Crash é recebida, o Receptor de Erro entra em estado morto. Em seguida, a falha de (iii) Timing é imitada entrando em um estado de espera por um período aleatório (0 a n tempo de simulação) e liberando então os dados. Finalmente, para reproduzir uma falha de (iv) Resposta, o ER gera uma mensagem corrompida gerada aleatoriamente com base na entrada.

Na Figura 11 é possível visualizar o *Sensor* encapsulado como uma entidade hierárquica (Sensor1System), sendo mantidas as suas portas de entrada (inMeasure) e de saída (outMeasure). Também é possível visualizar o seu ER (ErrorReceiverSensor1), responsável por reproduzir o comportamento de acordo com a falha recebida. O ER controla o que entra e o que sai do constituinte, estando ligado as portas de entrada e saída do Sensor1. Também é possível visualizar seu FSG (StimuliGeneratorErrorReceiver1) ligado ao ER através da porta *fault*. Por fim, ainda é possível observar que neste momento da execução o FSG gerou um estímulo de omissão, podendo ser visualizado na mensagem Fault o atributo *omission* definido como *true* e no estado atual do ER (omissionState).

Figura 11 – Falha sendo gerada em um constituinte encapsulado durante uma simulação no MS4 Me.



Fonte: Autoria própria.

4.2 Avaliação

Para fins de avaliação, um ensaio experimental foi planejado e realizado para avaliar a viabilidade e validade do modelo. Além disso, o objetivo desta avaliação foi ser uma prova de conceito sobre a viabilidade do modelo para representar falhas no SoS e prever o impacto delas sobre os comportamentos do SoS. O experimento é um estudo *in silico*, pois todos os elementos, sujeitos e ambiente, são representados por modelos computadorizados. Para elaborar o objetivo e a pergunta de pesquisa, foi aplicada a GQM (Objetivo-Pergunta-Métrica, do inglês Goal-Question-Metric) (BASILI, 1992), também prescrita por (FRANÇA; TRAVASSOS, 2012), o resultado pode ser visto abaixo.

Métricas. Para este ensaio experimental, foram definidas as seguintes métricas: (m_1) *percentual de dados recebidos*, é realizada a contagem dos dados que chegam no gateway; e (m_2) *efeitos de falha observáveis*, cada tipo de falha produz um efeito diferente no SoS e destaca como este é afetado por ela. Para fins de demonstrar a validade dos modelos produzidos, essas métricas foram consideradas suficientes.

Variáveis. Variáveis independentes são (v_1) *classe de falha*, (v_2) *probabilidade de falhas*. Nenhuma variável dependente foi considerada neste ensaio experimental.

Cenários. Vinte e quatro cenários foram planejados para serem executados, seis para cada classe de falha, variando as probabilidades de falha dos seus constituintes. Para resumir

Tabela 1 – Cenários da Simulação do Modelo de Falhas.

Cenário	Falha	Probabilidade de Falha	Cenário	Falha	Probabilidade de Falha
1	Omissão	75%	13	Timing	75%
2	Omissão	50%	14	Timing	50%
3	Omissão	25%	15	Timing	25%
4	Omissão	10%	16	Timing	10%
5	Omissão	5%	17	Timing	5%
6	Omissão	1%	18	Timing	1%
7	Crash	10%	19	Response	75%
8	Crash	5%	20	Response	50%
9	Crash	1%	21	Response	25%
10	Crash	0.1%	22	Response	10%
11	Crash	0.05%	23	Response	5%
12	Crash	0.01%	24	Response	1%

os resultados apenas uma execução para cada cenário executado foi apresentada aqui. Números distintos de probabilidade de falhas foram escolhidos para permitir uma visão mais ampla do impacto de falhas no comportamento do SoS, desde um valor pequeno até uma enorme quantidade de estímulos de falha. Os cenários foram listados na Tabela 1. Todos os cenários foram executados utilizando a topologia árvore, com 4 sensores e 1 gateway. Uma representação da arquitetura pode ser visualizada na Figura 7. É importante notar que, para os cenários de *Crash*, foram escolhidas probabilidades menores, já que as chances acima de 1% produziram resultados semelhantes: todos os constituintes falharam e o SoS tornou-se totalmente indisponível. Como evidência desde problema, os cenários 7, 8 e 9 foram mantidos nos cenários apresentados neste trabalho.

Verificação do experimento. Para testar a corretude dos modelos, foram realizadas para cada classe de falha, execuções com 0% e 100% de probabilidade de ocorrência de falhas. Como resultado, com 0% de probabilidade, obteve-se para: (i) omissão, todos os dados chegando no gateway; (ii) crash não ocorrendo e todos os constituintes ficando totalmente disponíveis durante toda a simulação; (iii) timing, todos dados sendo recebidos e no tempo certo; (iv) response, nenhum dado sendo modificado. Enquanto com 100% de probabilidade, os resultados para: (i) omissão foram que nenhum dado foi recebido no gateway; (ii) crash ocorreu em todos os constituintes desde o primeiro ciclo; (iii) timing, nenhum dado chegou no tempo esperado; (iv) response, todos os dados foram modificados.

Preparação de dados. Todas as execuções foram alimentadas com o mesmo conjunto de dados descrito no capítulo 3, removendo essa variação e permitindo que o foco seja mantido no comportamento de SoS e na sua resposta a falhas e sua probabilidade.

O ambiente de simulação utilizado para executar os experimentos foi o MS4 Me e todo o experimento foi executado em um computador com as seguintes configurações: i7

7700k 4-Core 8-Thread a 4.5GHz, GTX 1080 8GB GDDR5 e 8GB de RAM. Demorou cerca de 5 minutos e 20 segundos para executar cada cenário. Toda a simulação foi concluída após cerca de 128 minutos (2,13 horas).

4.2.1 Relato dos Resultados

A seguir, todas as classes de falhas serão analisadas individualmente e seu respectivo impacto na disponibilidade do SoS será verificado. Os gráficos apresentados possuem no eixo X o tempo de simulação e no eixo Y o nível da água, em centímetros. Além disso, no gráfico também está representado, quando relevante, a quantidade de dados (Do inglês, Data Count) que chegaram ao *gateway*. Em todos os casos, o ideal é que o número de medidas recebidas seja de 4000. A equivalência do tempo de simulação com o tempo real se dá na proporção de 4 unidades de tempo de simulação para 5 minutos. Visto que, a cada 5 minutos os dados são lidos na realidade e na simulação demora-se 4 tempos para um novo dado ser lido. Todas as simulações demoraram 8000 unidades de tempo de simulação para serem concluídas, o que equivale a 3,5 dias. Para omissão, *timing* e falhas de *response*, será analisado o comportamento do tempo de simulação de 1200 a 1800, uma vez que ocorreu uma inundação neste período. Já para *crash*, será analisado o comportamento durante toda a execução, que se dá do tempo 0 até o tempo 8000 de simulação.

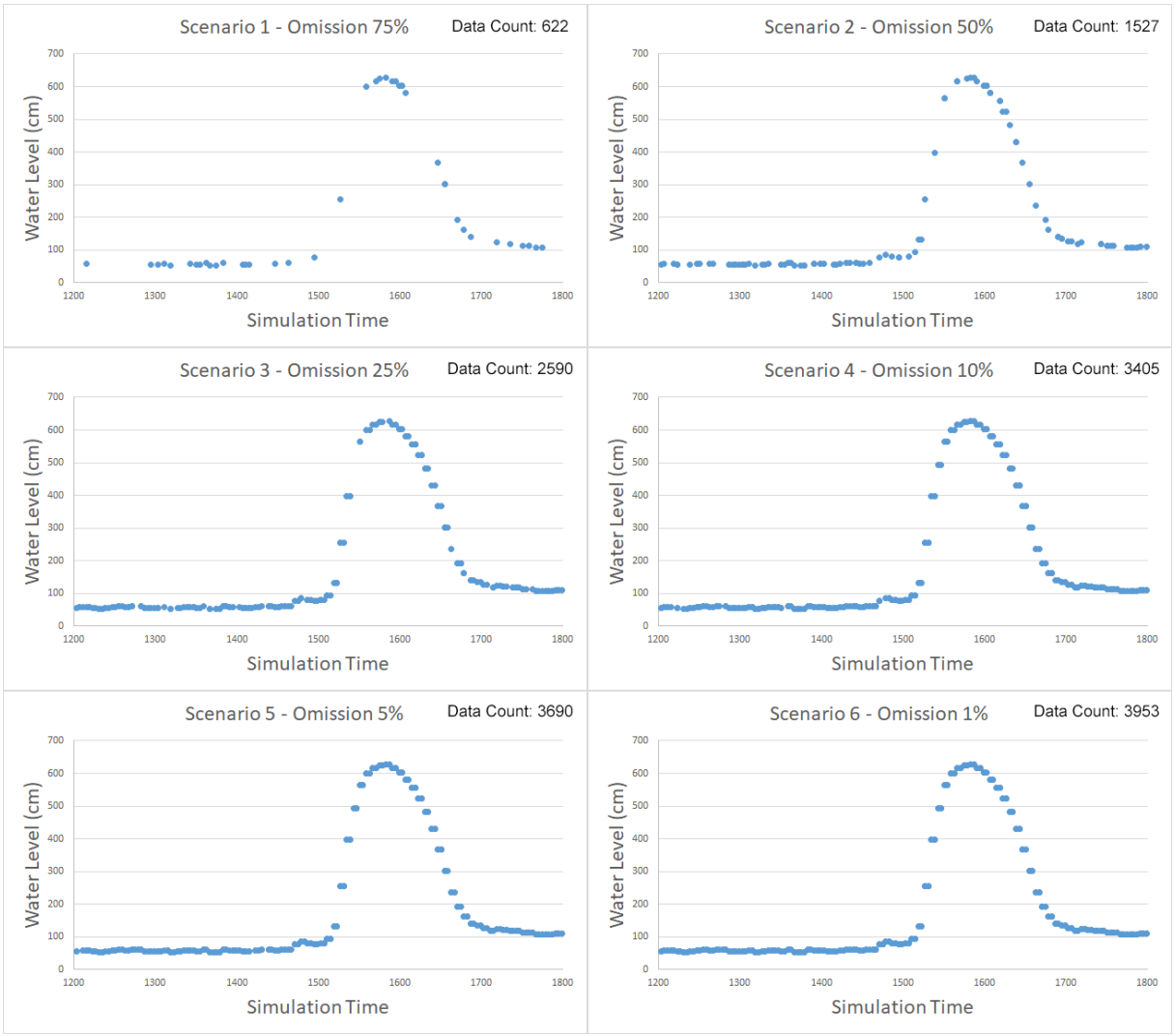
4.2.1.1 Falha de Omissão

Conforme apresentado na Figura 12, é possível observar que conforme a probabilidade de falhas aumenta, os dados recebidos no gateway diminuem. Os dados continuam chegando no tempo de simulação correto e os constituintes continuam entregando os dados quando a falha de omissão chega ao fim, isto é, os constituintes não estão em crash. Portanto o modelo de simulação está funcionando como o esperado. Além disso, durante os testes foi possível perceber que omissão é a classe de falha mais previsível, já que o número de dados recebidos pelo *gateway* é muito semelhante se fosse executado o mesmo cenário várias vezes. No entanto, utilizamos da simulação para analisar casos em que falhas ocorram em momentos específicos e que possam comprometer a missão do SoS, podendo causar dano a pessoas ou danificar equipamentos.

4.2.1.2 Falha de Crash

Nas execuções com 10%, 5% e 1% de probabilidade de falha, todos os constituintes se tornaram indisponíveis logo no início da simulação, como pode ser observado nos gráficos da Figura 13. Ao reduzir a probabilidade de falhas, é que foi possível analisar melhor o impacto da falha no SoS. Foi possível visualizar no cenário 11 que um constituinte se tornou indisponível no final da execução, afetando a quantidade de dados totais que chegaram ao *gateway*.

Figura 12 – Cenários de Omissão.



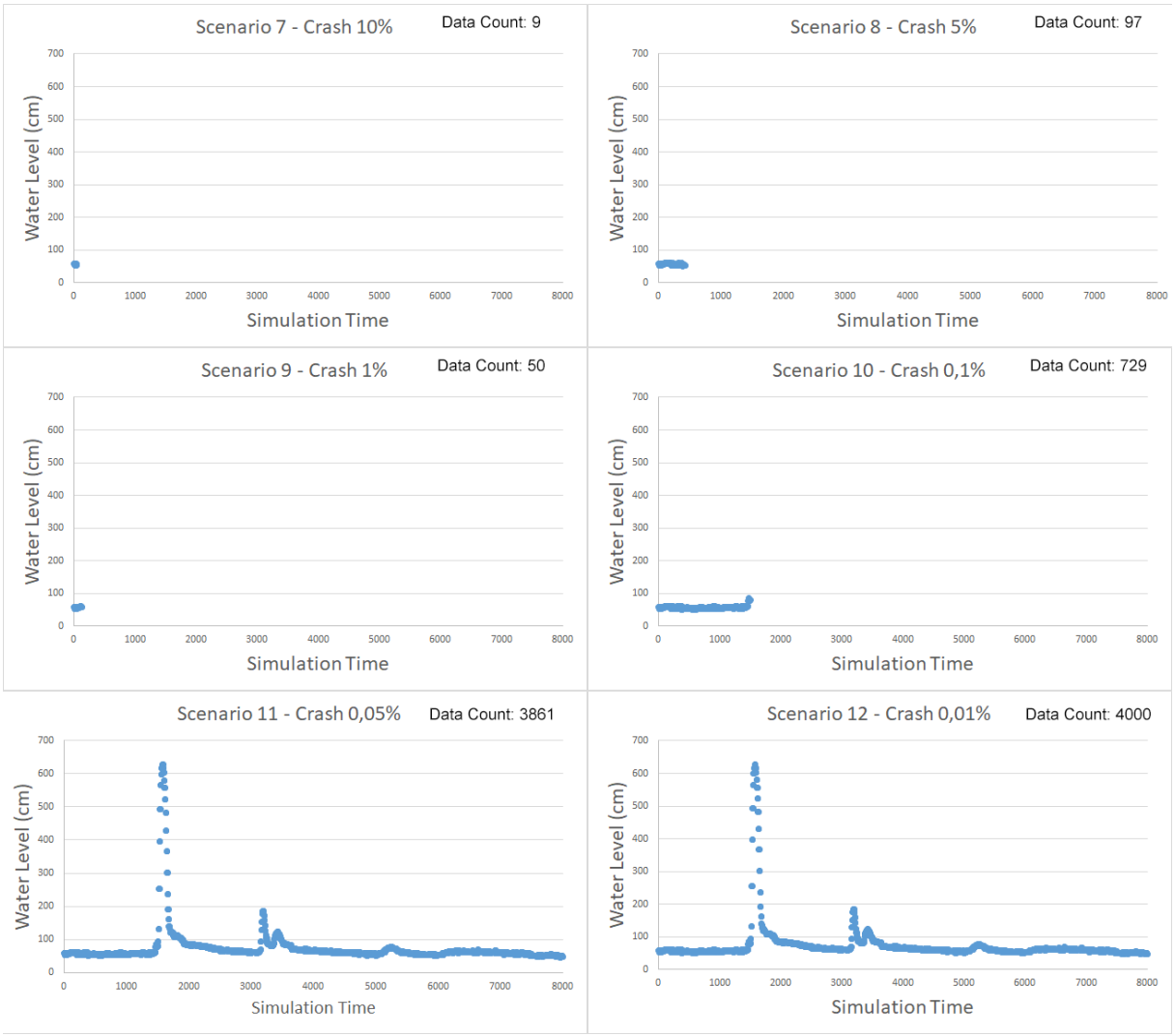
Fonte: Autoria própria.

Ademais. as execuções com falhas do tipo *crash* nos mostram como essa classe de falha é crítica para um sistema. Mesmo com uma baixa chance de ocorrência, o *crash* tem um impacto extremamente alto em um sistema, e essa probabilidade de ocorrência deve ser minimizada, já que quando inviabilizar um componente, dependendo da topologia do SoS, pode comprometer todo o SoS. Conlui-se que é possível observar a simulação emulando as falhas de *crash*.

4.2.1.3 Falha de Timing

Por outro lado, a falha de *timing* provoca uma dessincronia nos modelos e/ou até mesmo na execução em tempo real. Isso causa uma perda de dados sequencial, pois o constituinte deve estar disponível no momento certo para receber os próximos dados no fluxo de dados. Quando houver um atraso nos primeiros dados, todos os outros dados da

Figura 13 – Cenários de *Crash*.



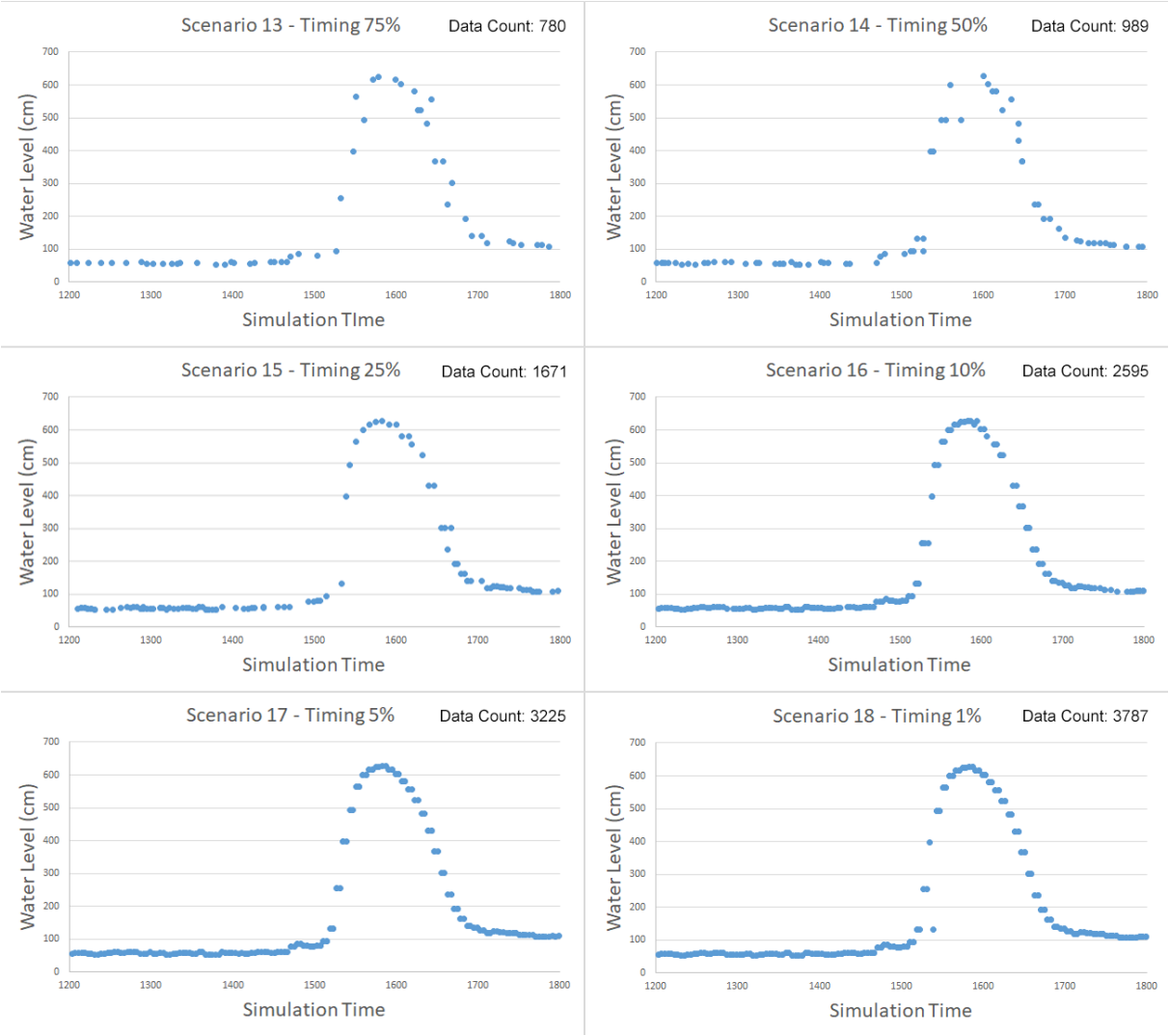
Fonte: Autoria própria.

sequencia serão afetados, aumentando o tempo de inatividade e causando um alto número de perda de dados.

É importante ressaltar que dados atrasados podem comprometer a missão global do SoS. Como visto no cenário 18 da Figura 14, em que com apenas 1% de chance de ocorrer uma falha de *timing*, quando o nível da água estava subindo (tempo de simulação 1540), o *gateway* recebeu uma leitura atrasada indicando que o nível da água era de aproximadamente 110 cm, enquanto a inundação estava acontecendo e o nível água real era de $\cong 500cm$.

Quanto mais a porcentagem de falhas aumenta, mais é possível observar os efeitos da falha de *timing* no SoS. O número de dados de leitura diminui, pois os constituintes com falha de *timing* não estão no estado de leitura e o nível de água não é entregue ao *gateway*. Os cenários 17 e 18, apesar de terem uma baixa contagem de dados, não apresentaram

Figura 14 – Cenários de falhas do tipo *Timing*.



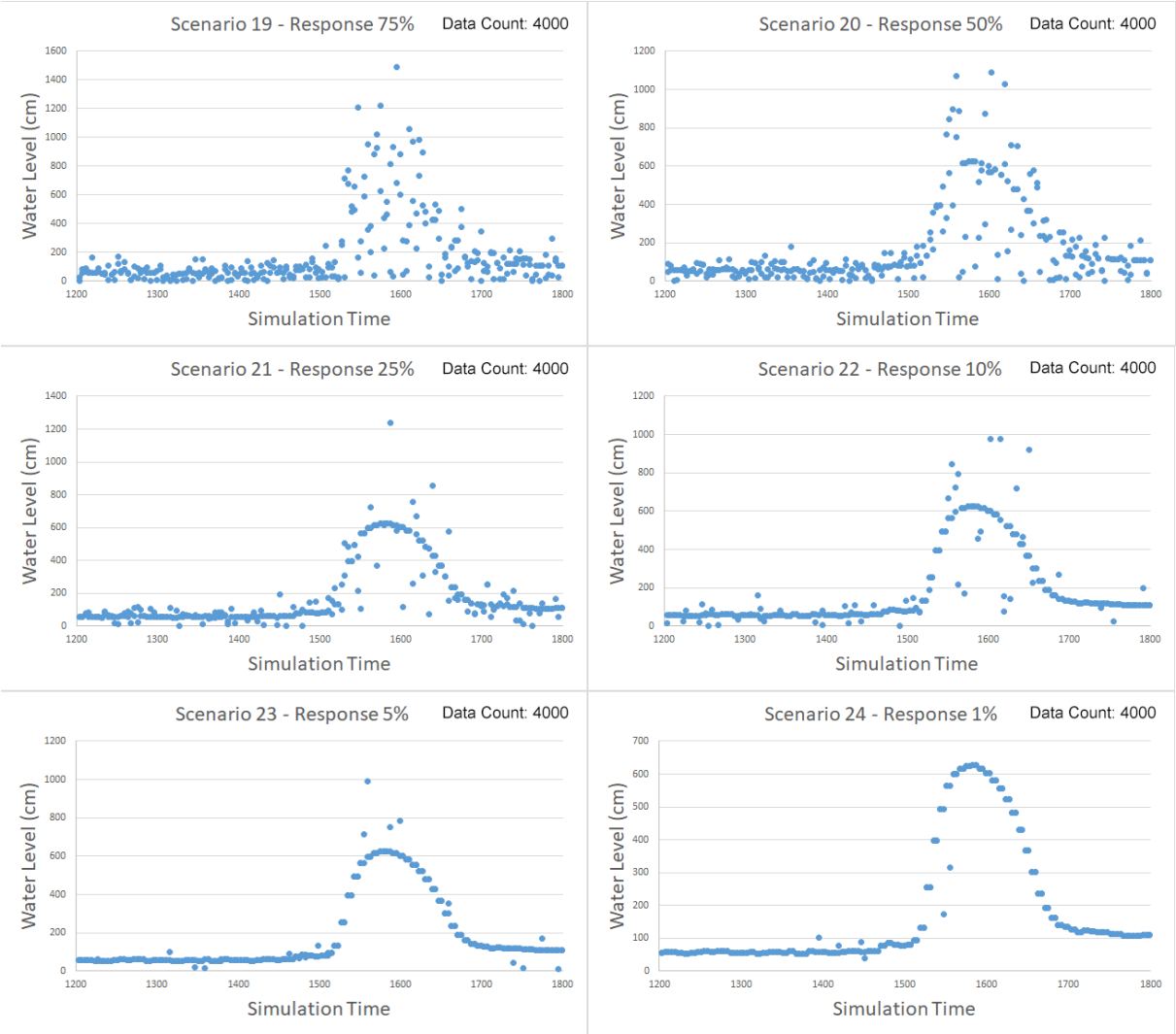
Fonte: Autoria própria.

nenhum problema durante a enchente que pudesse comprometer a missão global do SoS. Porém, uma contagem de dados (Do inglês Data Count) mais baixa compromete a disponibilidade do SoS, e, conforme visto na imagem, está afastada dos esperados 4000 dados recebidos no gateway. Portanto a simulação conseguiu reproduzir as falhas de *timing*, embora ainda não tenha sido validada com o comportamento no mundo real.

4.2.1.4 Falha de Response

As falhas de *response* não são afetadas por atrasos ou problemas de dados perdidos, consequentemente todos os dados chegaram no *gateway*, conforme visto na Figura 15. Além disso, é possível observar o impacto da falha de resposta mesmo com 1% de chance de ocorrência, visto que as leituras de SoS são comprometidas, ou seja, no cenário 24 podemos ver falhas antes e durante a inundação. Da mesma forma, à medida que a chance

Figura 15 – Cenários para falha de *Response*.



Fonte: Autoria própria.

de falha aumenta, podemos observar que o SoS se torna incapaz de cumprir sua missão. Assim sendo, é visível a corrupção dos dados durante a execução da simulação, cumprindo o objetivo de simular a falha de *response*.

Em seguida, dados esses resultados, pode-se finalmente responder uma das Questões de Pesquisa (QP): *Como modelar falhas em simulação de arquitetura de Sistemas-de-Sistemas?* Considerando as métricas *percentual de dados recebidos* (m_1) e *efeitos de falha observáveis* (m_2), é possível observar o impacto da falha de constituintes em todo o SoS e não só é possível observar, mas também é possível perceber a diferença entre as classes de falha e raciocinar sobre os diferentes cenários. Portanto, conclui-se que os modelos são capazes de reproduzir falhas em Sistemas-de-Sistemas intensivos em software.

4.3 Discussão

Os modelos de simulação apresentados neste capítulo permitem aos arquitetos prever, nas fases iniciais do projeto, como o SoS responderá às falhas de disponibilidade. Ao fornecer tais modelos, pretendia-se reduzir as dificuldades de antever o comportamento, as falhas e os resultados no SoS, levando à melhoria da qualidade no mesmo. Além disso, como esses modelos são modulares, os arquitetos de software podem reutilizá-los facilmente em seus projetos de arquitetura, para simular e avaliar a qualidade de seus sistemas. Como ameaças à validade dos modelos e do ensaio experimental desenvolvido neste artigo, notamos o seguinte. (i) Os modelos da falha de *timing* podem não estar representando a realidade, já que na simulação a máquina de estados deve estar no estado correto para receber os dados, embora, em aplicações reais, existam *buffers* e protocolos para tratar parte deste problema. Mas, a representação da falha de *timing* é real ao representar atrasos de rede, atrasos de processamento, isto é, casos em que há atraso na entrega do pacote. (ii) Cenários onde todos os tipos de falhas podem acontecer não foram executados, com a intenção de filtrar o comportamento de cada um, permitindo-nos fazer uma análise detalhada do impacto da falha no SoS. (iii) Os resultados não foram comparados com dados e resultados reais, para validar sua confiabilidade.

4.4 Considerações Finais

Neste capítulo, apresentamos um modelo de simulação para falhas de disponibilidade e um ensaio experimental foi executado como verificação e validação. Os resultados obtidos neste experimento nos permitiram responder um das questões de pesquisa propostas. Os modelos de simulação reproduzem o comportamento das falhas corretamente e cada classe de falha apresenta um impacto diferente no comportamento do SoS. Como trabalho futuro, pretendemos comparar as diferentes classes de falhas e também investigar novos tipos de falhas. Um exemplo é aquele produzido pela interoperabilidade entre sistemas constituintes de SoS, isto é, uma falha no mediador, isto é, meios de comunicação apresentados por (FERREIRA; SANTOS, 2019). No próximo capítulo, analisaremos a disponibilidade com base nos modelos de falha apresentados neste capítulo.

Parte V

Apresentação do Modelo de Disponibilidade

5 Modelagem do Atributo de Qualidade Disponibilidade em Modelos de Simulação de SoS

Para que seja possível medir e prever um atributo de qualidade, é necessário uma *Medição da Resposta*. Na disponibilidade, de acordo com (BASS; CLEMENTS; KAZMAN, 2012), a medição da resposta pode envolver (i) o momento ou o intervalo em que um sistema deve estar disponível no SoS ou (ii) a porcentagem de disponibilidade. Para obter a *Medição da Resposta* em nosso modelo de simulação, uma *Fonte de Estímulos* foi criada como uma entidade artificial implantada junto com a simulação, conforme já visto no capítulo 4. Essa entidade gera um *Estímulo* para um *Artefato* que está sendo executado sob certas condições, chamadas de *Ambiente*, produzindo uma *Resposta*. Neste capítulo serão avaliadas as medições da resposta do SoS à falha de omissão como forma de prover subsídios para uma avaliação mais detalhada da disponibilidade.

5.1 Modelo Proposto

Os modelos de simulação são descritos na linguagem DEVS, seguindo as diretrizes de Modelagem e Simulação de Sistemas-de-Sistemas propostas por (ZEIGLER; SARJOUGHIAN, 2017). Em primeiro lugar, o arquiteto especifica o modelo acoplado que representa a coalizão inicial. A coalizão a ser simulada é composta de modelos atômicos que representam os constituintes individuais. Sob a perspectiva de simulação DEVS para SoS, existem dois tipos de entidades: (i) constituintes e seus links de interoperabilidade, que representam entidades do mundo real e (ii) entidades artificiais, como o *Geradores de Estímulos de Falhas* (FSG), que não tem correspondência no mundo real, e apóia-se na contribuição produzida e apresentada no Capítulo anterior desta monografia (o modelo de falhas para SoS). O FSG é o núcleo para avaliar a disponibilidade em um modelo de simulação. Para simular as falhas, o FSG gera as falhas de acordo com as probabilidades pré-estabelecidas para cada cenário, e entrega as falhas a cada sensor, reproduzindo cada categoria de falhas, e possibilitando a análise das consequências da falha em todo o comportamento sendo entregue pelo SoS.

Justificativa. Neste contexto, durante o estudo levantou-se a hipótese de que a disponibilidade de um SoS consiste em ter um ou mais constituintes que fornecem os resultados mínimos que mantêm o SoS em operação. Então, a hipótese é que um SoS não fica indisponível devido à indisponibilidade individual dos constituintes, mas sua disponibilidade é

afetada quando um conjunto mínimo de constituintes se torna indisponível, ou seja, um conjunto de constituintes que ainda é capaz de entregar os resultados esperados de um SoS.

5.1.1 Design Experimental

Esta seção apresenta o protocolo e materializa o projeto experimental com base nas diretrizes para experimentos baseados em simulação propostos por França e Travassos e (FRANÇA; TRAVASSOS, 2016). O objetivo deste capítulo é realizar um estudo *in silico*, ou seja, um estudo em que tanto os sujeitos quanto o ambiente são representados por modelos computadorizados (simulação).

Métricas. Para este estudo, as métricas estabelecidas foram *(m_1) disponibilidade dos constituintes*, uma vez que a porcentagem de constituintes disponíveis poderia impactar em toda a disponibilidade de SoS, *(m_2) porcentagem de resultados entregue pelo SoS*, uma vez que a disponibilidade de todo o SoS está diretamente relacionada ao número de resultados entregues em relação ao número de unidades de processamento (constituintes) envolvidos.

Variáveis. Para prosseguir com a avaliação, planejou-se analisar as seguintes variáveis: a *(v_1) Topologia Arquitetural*, isto é, o arranjo e a implantação geográfica dos constituintes e as variáveis correspondentes para as métricas estabelecidas, que são v_2 para *Disponibilidade de Constituintes* e v_3 para *Disponibilidade do SoS*. v_1 é uma variável independente. v_3 é uma variável de resposta (dependente), isto é, assumimos que a disponibilidade de SoS depende das outras variáveis, particularmente v_2 (disponibilidade de constituintes). É importante observar que, para o escopo deste estudo, nos concentramos em falhas de omissão para investigar a disponibilidade, o que significa que não utilizamos as outras classes de falhas descritas no capítulo 4, ou seja, *Crash*, *Timing* e *Response*.

Cenários. Em conformidade com as diretrizes de estudo baseadas em simulação, a Tabela 2 exibe os cenários e as variáveis escolhidas, usadas para analisar a disponibilidade no contexto do SoS. Duas topologias foram planejadas (elas serão apresentadas na próxima seção): linha e árvore. Na topologia da linha, como mostrado na Figura 6, quatro sensores inteligentes são posicionados nas bordas do rio em uma linha e eles encaminham os dados até alcançar o *gateway*. Nesta última topologia, o *gateway* é posicionado entre os sensores 2 e 3. Foram elaborados doze cenários para avaliar a arquitetura do SoS quanto à sua disponibilidade. Para cada arranjo arquitetural diferente (linha e árvore), planejamos medir como as mesmas probabilidades de omissão de cada constituinte poderiam afetar toda a disponibilidade da arquitetura do SoS.

Para cada cenário, as simulações foram executadas usando as seguintes probabilidades de falhas a serem atribuídas separadamente para cada constituinte durante a simulação: 75%, 50%, 25%, 10%, 5% e 1%. Uma vez que as probabilidades foram definidas para as

Tabela 2 – Cenários de Simulação do Modelo de Disponibilidade.

Cenário	Topologia	Probabilidade de Omissão para cada constituinte
1	Linha	75%
2	Árvore	75%
3	Linha	50%
4	Árvore	50%
5	Linha	25%
6	Árvore	25%
7	Linha	10%
8	Árvore	10%
9	Linha	5%
10	Árvore	5%
11	Linha	1%
12	Árvore	1%

falhas, essa decisão permite que o arquiteto observe resultados diferentes dos mesmos modelos e dados, mas com uma distribuição diferente das falhas durante a execução.

Verificação do experimento. Foi realizado um ensaio experimental dos modelos concebidos para assegurar que os modelos (i) estavam de fato reproduzindo as falhas, (ii) o modelo não interfere na coleta de dados e (iii) o modelo de simulação reproduz os comportamentos pretendidos. Em seguida, executamos os modelos com as diferentes arquiteturas e com probabilidades de 0% de disponibilidade, nas quais nenhum dado foi entregue pelo SoS devido à indisponibilidade e com 100 %, no qual todos os dados foram entregues. As outras porcentagens foram testadas durante o estudo. Em seguida, prosseguimos com o experimento após essa etapa.

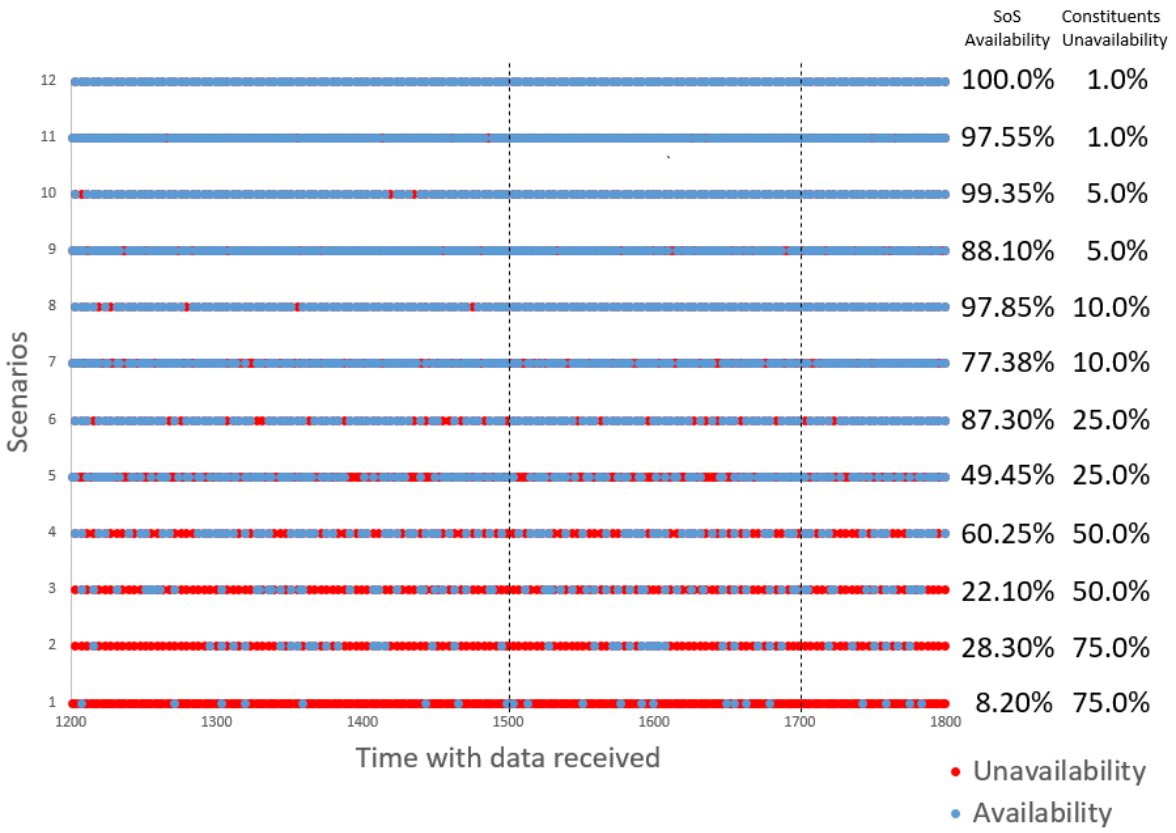
Validade do Modelo de Simulação. Além de atestar a validade dos modelos no Capítulo 4, também incluímos um procedimento para a validade do modelo, que é uma preocupação importante para a simulação. Um especialista foi designado para avaliar a validade do modelo elaborado. Ele é PhD com três anos de experiência em modelos de simulação DEVS. A próxima seção relata a avaliação das execuções dos cenários.

5.2 Relato dos resultados

Esta seção relata a realização da avaliação da proposta de acordo com o *design* experimental. O estudo foi executado em uma máquina i7 7700k 4-Core 8-Thread a 4.5GHz, GTX 1080 8GB GDDR5, 8GB RAM DDR4 e o ambiente de simulação foi o MS4 Me. Cada cenário para a topologia de árvore levou 5 minutos e 20 segundos. Por sua vez, os cenários de linha levaram cerca de 10 minutos e 40 segundos para concluir uma execução.

A topologia linha levou mais tempo porque os dados utilizam mais transições de estados até serem encaminhadas e alcançarem o *gateway*. Toda a simulação levou cerca de 95,4 minutos (1,59 horas).

Figura 16 – Comparação da disponibilidade do SoS em doze cenários.



Fonte: Autoria Própria.

Avaliamos a arquitetura SoS usando (i) duas topologias diferentes (v_1, *topologia arquitetural*), (ii) probabilidades diferentes de falhas de omissão, ou seja, a possível indisponibilidade de cada constituinte (v_2) e (iii) a disponibilidade de todo o SoS (v_3). Figura 16 mostra os resultados da experiência realizada. Cada linha do gráfico corresponde a um dos cenários planejados. Cada ponto do eixo X representa cada instante do tempo em que o *gateway* efetivamente recebeu um dos dados coletados dos sensores, considerando sua possível disponibilidade/indisponibilidade. Uma vez que o conceito de disponibilidade é altamente dependente da porcentagem de vezes que o sistema é capaz de fornecer resultados durante todo o período de operação, a disponibilidade total de SoS obtida é a porcentagem de vezes que pelo menos uma amostra de dados coletada por um dos constituintes chegou ao *gateway*. Os pontos vermelhos representam instantes de tempo quando nenhum dado foi recebido pelo *gateway*, representando que o SoS estava indisponível naquele momento. Pontos azuis representam momentos no tempo em que pelo menos um dado foi recebido pelos *gateways* de pelo menos um dos constituintes. Os números à direita na Figura 16 representam, respectivamente, (i) a disponibilidade de todo o SoS, e (ii) a indisponibilidade

relativa dos constituintes para cada cenário avaliado.

Para cada topologia, a mesma probabilidade de falhas foi avaliada. No Cenário 1, por exemplo, a disponibilidade de SoS era muito baixa (8,2%), pois a probabilidade de falhas era muito alta para cada constituinte (75%). Para o Cenário 1, já é possível observar como a topologia árvore apresentou resultados melhores que a topologia linha, mostrando uma disponibilidade quase quatro vezes maior que a topologia linha. À medida que a probabilidade de falhas diminuiu, a disponibilidade em ambas as topologias também aumentou. No entanto, é possível observar que, em todos os cenários, o grau de disponibilidade da topologia árvore foi maior que a topologia linha. O cenário 12, por sua vez, é o caso mais emblemático: pode-se observar que, para um caso em que a probabilidade de falha foi de 1% para cada constituinte, o SoS não se tornou indisponível em nenhum momento porque a indisponibilidade de um dos constituintes foi compensado pelos outros. Por sua vez, no caso da topologia linha ainda havia uma indisponibilidade de quase 3%, o que pode ser grave para domínios críticos, que devem estar *altamente disponíveis*[1].

A figura 16 também mostra o momento em que a enchente começa e termina. Os números na primeira coluna à direita representam a disponibilidade total obtida da simulação de SoS para cada cenário. Por sua vez, cada ponto no gráfico representa 0,625 minuto. Então, os dados plotados (600 pontos) representam 6,25 horas do tempo real total simulado para fins de exemplificação. Em particular, o período mostrado representa parte do dia em que ocorreu uma inundação real. A seção delimitada por linhas verticais (entre os pontos 1500 e 1700 na linha do tempo) representa o período em que ocorreu a inundação. No caso analisado, o nível da água foi aumentado em seis metros em 30 minutos (das 2,08 horas delimitadas pelas linhas verticais). No Cenário 2 é possível observar que havia uma indisponibilidade de todo o SoS no momento em que a enchente começou. O SoS só se tornou disponível novamente no momento 1550 (cerca de 30 minutos após os últimos dados coletados), ou seja, naquele momento, a inundação já estava em andamento, e a população não foi alertada.

Resultados. Como esperado, os cenários em que a disponibilidade de constituintes era alta levaram a altos níveis de disponibilidade de SoS. Também foi possível concluir que a topologia influencia intensamente no sucesso de um SoS para fornecer disponibilidade. Na topologia árvore, mesmo uma alta indisponibilidade dos constituintes não teve um impacto equivalente na disponibilidade de SoS. No entanto, na topologia linha, a indisponibilidade de SoS foi ainda menor do que a relativa indisponibilidade dos constituintes.

Então, podemos responder à pergunta de pesquisa: *Como avaliar disponibilidade de SoS utilizando modelos de simulação de arquitetura de SoS?* Considerando as métricas que estabelecemos para avaliar os cenários, concluímos que *disponibilidade do SoS (m_2)* é altamente dependente de *disponibilidade dos constituintes (m_1)*. No entanto, a *topologia*

[1] Disponibilidade acima de 99,999%, segundo (BASS; CLEMENTS; KAZMAN, 2012)

arquitetural é um fator que influencia muito a disponibilidade fornecida pelo SoS. Também é importante observar que, como mencionado por (FRANÇA; TRAVASSOS, 2016), não foi realizado o teste de hipótese, uma vez que em um estudo baseado em simulação a definição das métricas permite ao arquiteto elaborar as questões de pesquisa como hipóteses e testes. Concluímos então que o grau de disponibilidade que o SoS exibe depende (i) do número de constituintes disponíveis, (ii) do seu grau de disponibilidade e (iii) da topologia do SoS.

5.3 Discussão

Neste capítulo, foram analisadas duas topologias diferentes, ou seja, duas formas diferentes de conectar os mesmos componentes para analisar como sua disponibilidade individual afeta a disponibilidade total do SoS. É importante destacar que lidamos com componentes redundantes, ou seja, mesmo que possam ser de diferentes fornecedores, eles oferecem funcionalidades semelhantes. Então, a disponibilidade de um SoS em que a redundância de funcionalidades é menor poderia ser mais afetada pela indisponibilidade individual de seus constituintes.

O estudo foi conduzido sob as diretrizes de França e Travassos (FRANÇA; TRAVASSOS, 2016). O estudo adota uma abordagem baseada em simulação para reunir evidências para uma investigação empírica no contexto da Engenharia de Software para Sistemas de Sistemas. Como tal, este estudo também sofre ameaças à validade. Sobre a *validade da conclusão*, a confiabilidade de nossos resultados pode ser afetada pelos procedimentos estatísticos que foram adotados durante o planejamento experimental. Para aliviar essa ameaça, a probabilidade de falhas de constituintes foi gerada de forma aleatória, foram adotadas diversas porcentagens diferentes de disponibilidade e uma linha de base de comparação entre os cenários elaborados foi estabelecida. Em relação à *validade interna*, foi estabelecida uma possível relação causal entre a topologia de SoS e sua disponibilidade. Para aliviar essa ameaça, foram executados testes com diferentes topologias, usando os mesmos dados de entrada, o que permitiu analisar resultados diferentes; isso não garante generalidade para a conclusão, mas reduz o impacto dessa ameaça. *Validade de construção*, por sua vez, diz respeito às relações entre teoria e observação. Como se trata de um estudo baseado em simulação, uma preocupação recorrente é se o modelo de simulação representa a realidade. O modelo foi submetido à avaliação de um especialista em monitoramento e simulação de inundações. Como ele concordou com a verossimilhança do modelo apresentado aqui, essa ameaça teve seu impacto reduzido. Finalmente, *validade externa* se preocupa com a generalização dos resultados experimentais. Como o escopo é modesto, os resultados não podem ser generalizados para outros domínios e topologias sem estudos adicionais. Um número crescente de constituintes também deve ser analisado. No entanto, foram investigados 12 cenários diferentes. Assim, a confiabilidade de nossos resultados é garantida para, pelo menos, esse escopo, esse tamanho de problema e domínio.

5.4 Estratégias de Projeto para Disponibilidade em SoS - O método Salva-Vidas

Durante o desenvolvimento deste trabalho, foi pensada uma forma de reduzir a indisponibilidade em casos de falhas que inviabilizassem seu funcionamento. A ideia é prover um mecanismo de *substituição de funcionalidade* no caso de falha de algum constituinte crítico, isto é, aquele constituinte que se indisponível inviabiliza a missão global. Sabendo que um constituinte de SoS possui software, hardware e ainda é influenciado por características do ambiente no qual está inserido, uma substituição de funcionalidade pode se dar tanto por um constituinte com as mesmas características, quanto por outro essencialmente diferente, mas que possa prover funcionalidade semelhante e/ou complementar. Foi idealizado então um constituinte Salva-Vidas, um drone e uma base de drone, que integrariam o FMSoS. A base do drone receberia os dados assim como o gateway, e decidiria, baseado em uma série de regras, quando seria necessária a atuação do drone. O Salva-Vidas, quando ativado, se deslocaria pelo ar filmando até as coordenadas do constituinte falho e, assim que estivesse sobre o rio, realizaria uma medida, que seria imediatamente enviada ao gateway. Dessa forma, as autoridades teriam ao menos medidas do nível do rio em momentos críticos e ainda uma visão aérea do que estaria havendo na região onde o constituinte faltoso está inserido. Em caso de furto, vandalismo ou outros crimes de destruição de constituintes, os responsáveis teriam imagens aéreas do local e poderiam tomar medidas punitivas contra os criminosos. É bom ressaltar também que, no caso do FMSoS, a enchente não necessariamente ocorre no mesmo local onde houve a chuva, e de qualquer forma, existem drones que voam durante a chuva. Portanto, o drone poderia alçar voos mesmo em situações críticas e substituir a funcionalidade de um sensor, mesmo possuindo características físicas e de implementação completamente diferentes.

5.5 Considerações Finais

Neste capítulo, foram apresentados resultados preliminares de uma investigação sobre a disponibilidade de SoS para responder à seguinte pergunta de pesquisa: *Como avaliar disponibilidade de SoS utilizando modelos de simulação de arquitetura de SoS?*. Um SoS de Monitoramento de Inundação Urbana foi utilizado para investigar este tópico. Os resultados obtidos mostram que a disponibilidade de SoS não depende apenas da disponibilidade individual fornecida pelos sistemas constituintes, mas também é influenciada pela topologia de SoS, ou seja, como os constituintes estão posicionados geograficamente e como eles se conectam e interagem entre si. Uma conclusão importante é que talvez a análise da disponibilidade de SoS deva acontecer em conjunto com outros atributos de qualidade. Apesar de mostrar que o SoS está disponível, ou seja, que pelo menos um constituinte pode entregar o resultado esperado, também é necessário garantir que os

dados entregues estejam corretos e confiáveis. Portanto, uma análise conjunta de vários atributos de qualidade deve ser realizada para garantir a qualidade do SoS como um todo. Trabalhos futuros incluem a avaliação da disponibilidade no SoS para (i) outros domínios, (ii) outras topologias, (iii) outras classes de falhas e (iv) instâncias maiores. Além disso, pretendemos desenvolver mecanismos arquiteturais para tolerância e recuperação de falhas no SoS. No entanto, os resultados comunicados neste capítulo são um passo importante para consolidar o estudo da engenharia de software no domínio SoS. Particularmente, os resultados apresentados aqui trazem uma contribuição significativa, uma vez que o aumento da qualidade no domínio SoS não está relacionado apenas para aumentar a satisfação do usuário; mas, principalmente, isso está relacionado a garantir a confiabilidade dos serviços prestados à população e aumentar sua confiança nos sistemas que estão sendo preparados para emergir nos próximos anos.

Parte VI

Considerações Finais

6 Conclusão

Devido à necessidade de estudos de atributos de qualidade para Sistemas-de-Sistemas, este trabalho de conclusão de curso apresentou, avaliou e analisou um modelo de geração de falhas para simulação de arquitetura de Sistemas-de-Sistemas. Além disso, a partir do modelo de falhas, um experimento foi realizado de forma a prover subsídios para uma futura análise mais aprofundada do atributo de qualidade disponibilidade a partir da experimentação de um Sistema de monitoramento de enchentes.

6.1 Resultados Obtidos

A partir dos resultados dos experimentos desenvolvido no capítulo 4, foi possível observar o impacto dos diferentes tipos de falha de disponibilidade como forma de avaliar e validar os modelos de simulação. Conclui-se que o modelo de geração de falhas conseguiu reproduzir o comportamento dos 4 tipos de falha de disponibilidade, possibilitando então a mensuração da disponibilidade e avançando o estado da arte, uma vez que os outros estudos existentes na literatura de modo geral tratam apenas de um ou dois dos tipos de falhas, mas não dos quatro tipos (BOGADO; GONNET; LEONE, 2014; O'Brien; Merson; Bass, 2007) .

Da mesma forma, os resultados dos experimentos executados no capítulo 5 possibilitaram a comparação das topologias linha e árvore, para isso foram utilizadas probabilidades de falhas dos constituintes diferentes e conclui-se que a indisponibilidade dos constituintes afeta a disponibilidade do SoS junto com a topologia, que é um dos fatores de influência direta no grau disponibilidade de um SoS.

Por fim, durante o desenvolvimento deste projeto, foram desenvolvidos dois artigos e submetidos a eventos nacionais e internacionais. O primeiro foi para o ESEM 2019 [1], na categoria "Resultados Emergentes", com o seguinte título: "Simulation-Based Measurement and Prediction of Availability in Systems-of-Systems". Já o segundo artigo foi escrito para o evento MSSiS 2019 [2] na categoria "Artigos Completos"e possui o seguinte tema: "A Simulation Model for Faults in Software-Intensive Systems-of-Systems".

6.2 Contribuições

Como contribuição para este trabalho de conclusão de curso, foi obtido um modelo de geração das falhas de disponibilidade para simulação de modelos de arquitetura de

[1] http://eseiw2019.com/esem
[2] http://inf.ufg.br/mssis

software de Sistemas-de-Sistemas. O modelo é modular e desacoplado, o que garante a fácil reutilização futura do modelo por arquitetos e engenheiros de software como forma de avaliar e predizer certos comportamentos em seus sistemas. Além disso, no capítulo 5, este trabalho contribuiu com subsídios para uma avaliação mais aprofundada da disponibilidade de SoS, tópico que carece de estudos mais aprofundados, através de um modelo de simulação de disponibilidade para SoS.

6.3 Trabalhos Futuros

Como trabalhos futuros, vislumbrou-se: (i) realizar uma revisão de literatura aprofundada sobre falhas e disponibilidade de Sistemas-de-Sistemas; (ii) desenvolver um modelo de avaliação de disponibilidade com base nos resultados alcançados; (iii) testar como lidar com ocorrências simultâneas diferentes tipos de falhas em uma mesma arquitetura de SoS; (iv) averiguar a fidedignidade do modelo de simulação em relação ao mundo real; e (v) verificar com especialistas se as conclusões não são óbvias e se são críveis para serem utilizadas como base na engenharia de software para SoS.

Referências

ALAMPALLI, S.; PARDO, T. A study of complex systems developed through public private partnerships. In: *Proceedings of the 8th International Conference on Theory and Practice of Electronic Governance*. New York, NY, USA: ACM, 2014. (ICEGOV '14), p. 442–445. ISBN 978-1-60558-611-3. Disponível em: <http://doi.acm.org/10.1145/2691195.2691212>. Citado na página 15.

BANKS, J. et al. *Discrete-Event System Simulation (3rd Edition)*. 3. ed. [S.l.]: Prentice Hall, 2000. Citado na página 28.

BARBOSA, G. M. G. *Um Livro-texto para o Ensino de Projeto de Arquitetura de Software*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte - UFRN, 2009. Citado na página 24.

BASILI, V. R. *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. College Park, MD, USA, 1992. Citado 2 vezes nas páginas 19 e 40.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 3rd. ed. [S.l.]: Addison-Wesley Professional, 2012. ISBN 0321815734, 9780321815736. Citado 10 vezes nas páginas 16, 17, 18, 24, 25, 26, 27, 37, 49 e 53.

BHATTA, B. Urban growth and sprawl. In: *Analysis of urban growth and sprawl from remote sensing data*. [S.l.]: Springer, 2010. p. 1–16. Citado na página 15.

BIANCHI, T.; SANTOS, D. S.; FELIZARDO, K. R. Quality attributes of systems-of-systems: A systematic literature review. In: *3rd IEEE/ACM SESoS*. Florence, Italy: [s.n.], 2015. p. 23–30. Citado 3 vezes nas páginas 16, 17 e 26.

BILLAUD, S.; DACLIN, N.; CHAPURLAT, V. Interoperability as a key concept for the control and evolution of the system of systems (sos). In: SINDEREN, M. van; CHAPURLAT, V. (Ed.). *Enterprise Interoperability*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. p. 53–63. ISBN 978-3-662-47157-9. Citado na página 22.

BOGADO, V.; GONNET, S.; LEONE, H. Modeling and simulation of software architecture in discrete event system specification for quality evaluation. *SIMULATION*, v. 90, n. 3, p. 290–319, 2014. Citado 2 vezes nas páginas 31 e 58.

CHAPURLAT, V.; DACLIN, N. System interoperability: definition and proposition of interface model in mbse context. *IFAC Proceedings Volumes*, v. 45, n. 6, p. 1523 – 1528, 2012. ISSN 1474-6670. 14th IFAC Symposium on Information Control Problems in Manufacturing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1474667016333675>. Citado na página 15.

CLEMENTS, P. et al. *Documenting software architectures: views and beyond*. [S.l.]: Pearson Education, 2002. Citado na página 16.

EUSGELD, I.; NAN, C.; DIETZ, S. System-of-systems approach for interdependent critical infrastructures. *Reliability Engineering & System Safety*, v. 96, n. 6, p. 679 – 686, 2011. ISSN 0951-8320. ESREL 2009 Special Issue. Citado na página 30.

FERREIRA, F. H. C.; SANTOS, R. P. dos. Arquitetura para prevenção e tolerância a falhas em sistemas-de-sistemas virtuais. 2019. Citado na página 47.

FRANÇA, B. B. N. de; TRAVASSOS, G. H. Reporting guidelines for simulation-based studies in software engineering. In: *Proc. of 16th EASE*. Ciudad Real, Spain: [s.n.], 2012. p. 156–160. Citado 3 vezes nas páginas 16, 31 e 40.

FRANÇA, B. B. N. de; TRAVASSOS, G. H. Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. *Empirical Software Engineering*, v. 21, n. 3, p. 1302–1345, 2016. ISSN 15737616. Citado 5 vezes nas páginas 16, 19, 28, 50 e 54.

GAGLIARDI, M.; WOOD, W. G.; KLEIN, J. A uniform approach for system of systems architecture evaluation. In: . [S.l.: s.n.], 2009. Citado na página 17.

Garro, A.; Tundis, A. On the reliability analysis of systems and sos: The ramsas method and related extensions. *IEEE Systems Journal*, v. 9, n. 1, p. 232–241, March 2015. ISSN 1932-8184. Citado na página 17.

Graciano Neto, V. V. Validating Emergent Behaviors in Systems-of-Systems through Model Transformations. In: *Proceedings of the ACM PhD Student Research Competition at MODELS 2016 co-located with the 19th International Conference on Model Driven Engineering Languages and Systems* . St. Malo, France: [s.n.], 2016. (MODELS 2016). Disponível em: <https://hal.archives-ouvertes.fr/hal-01443187>. Citado na página 15.

Graciano Neto, V. V. *A simulation-driven model-based approach for designing software-intensive systems-of-systems architectures*. Tese (Doutorado) — Universidade de São Paulo - USP, 2018. Citado na página 15.

Graciano Neto, V. V. et al. ASAS: An Approach to Support Simulation of Smart Systems. Proceedings of HICSS 2018. Big Island, Hawai., 2018. Citado na página 22.

Graciano Neto, V. V. et al. Supporting simulation of systems-of-systems software architectures by a model-driven derivation of a stimulus generator. *ser. WDES*, v. 16, p. 61–70, 2016. Citado na página 22.

Graciano Neto, V. V.; SANTOS, R. dos; ARAUJO, R. Sistemas de sistemas de informação e ecossistemas de software: Conceitos e aplicações. In: _____. [S.l.: s.n.], 2017. p. 22–41. ISBN 78-85-7669-379-6. Citado 3 vezes nas páginas 22, 23 e 24.

GRAY, J.; RUMPE, B. Models in simulation. *Software & Systems Modeling*, v. 15, n. 3, p. 605–607, Jul 2016. Citado na página 28.

HORITA, F. E. et al. Development of a spatial decision support system for flood risk management in brazil that combines volunteered geographic information with wireless sensor networks. *Computers Geosciences*, v. 80, p. 84 – 94, 2015. ISSN 0098-3004. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0098300415000746>. Citado na página 34.

IEEE. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. *IEEE Std 610*, p. 1–217, Jan 1991. Citado na página 15.

ISO. *ISO 25010:2011 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Geneva, Switzerland, 2011. v. 2011. Citado 3 vezes nas páginas 16, 25 e 27.

Johnson, P. et al. Modeling and analyzing systems-of-systems in the multi-attribute prediction language (mapl). In: *Proc. of 4th IEEE/ACM SESoS*. Austin, Texas: IEEE, 2016. p. 1–7. Citado na página 30.

MAIER, M. W. Architecting principles for systems-of-systems. *Systems Engineering*, v. 1, n. 4, p. 267–284, 1998. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291520-6858%281998%291%3A4%3C267%3A%3AAID-SYS3%3E3.0.CO%3B2-D>. Citado 4 vezes nas páginas 22, 23, 24 e 33.

MAIER, M. W. The role of modeling and simulation in system of systems development. In: _____. *Modeling and Simulation Support for System of Systems Engineering Applications*. John Wiley Sons, Ltd, 2015. cap. 2, p. 11–41. ISBN 9781118501757. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118501757.ch2>. Citado na página 23.

Meilich, A. System of systems (sos) engineering amp; architecture challenges in a net centric environment. In: *2006 IEEE/SMC International Conference on System of Systems Engineering*. [S.l.: s.n.], 2006. p. 5 pp.–. Citado na página 17.

MENDES, A. B. *Mandala - interoperabilidade baseada em sistemas de sistemas no âmbito de cidades inteligentes*. Dissertação (Mestrado) — Centro de Ciências Exatas e da Terra, Universidade Federal do Rio Grande do Norte, Natal, 2018. Citado 2 vezes nas páginas 22 e 23.

NAKAGAWA, E. et al. The state-of-the-art and future perspectives in systems-of-systems software architectures. *1st ACM SIGSOFT/SIGPLAN International Workshop on Software Engineering for Systems-of-Systems, SESoS 2013 Proceedings*, p. 13–20, 07 2013. Citado 3 vezes nas páginas 15, 22 e 25.

NAM, T.; PARDO, T. A. Conceptualizing smart city with dimensions of technology, people, and institutions. In: *Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times*. New York, NY, USA: ACM, 2011. (dg.o '11), p. 282–291. ISBN 978-1-4503-0762-8. Disponível em: <http://doi.acm.org/10.1145/2037556.2037602>. Citado na página 15.

NETO, V. V. G. et al. Stimuli-sos: a model-based approach to derive stimuli generators for simulations of systems-of-systems software architectures. *Journal of the Brazilian Computer Society*, v. 23, n. 1, p. 13, Oct 2017. ISSN 1678-4804. Disponível em: <https://doi.org/10.1186/s13173-017-0062-y>. Citado 3 vezes nas páginas 33, 35 e 39.

NIELSEN, C. B. et al. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 48, n. 2, p. 18:1–18:41, set. 2015. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2794381>. Citado 4 vezes nas páginas 22, 23, 24 e 27.

O'Brien, L.; Merson, P.; Bass, L. Quality attributes for service-oriented architectures. In: *International Workshop on Systems Development in SOA Environments (SDSOA'07: ICSE Workshops 2007)*. [S.l.: s.n.], 2007. p. 3–3. Citado na página 58.

OQUENDO, F. Formally describing the software architecture of systems-of-systems with sosadl. In: IEEE. *System of Systems Engineering Conference (SoSE), 2016 11th*. [S.l.], 2016. p. 1–6. Citado 3 vezes nas páginas 33, 37 e 38.

OQUENDO, F. Software architecture challenges and emerging research in software-intensive systems-of-systems. In: SPRINGER. *Proc. of 10th ECSA*. [S.l.], 2016. p. 3–21. Citado na página 16.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, ACM, New York, NY, USA, v. 17, n. 4, p. 40–52, out. 1992. ISSN 0163-5948. Citado na página 27.

PETCU, V.; PETRESCU, A. Systems of systems applications for telemedicine. In: *9th RoEduNet IEEE International Conference*. [S.l.: s.n.], 2010. p. 208–211. ISSN 2247-5443. Citado na página 22.

RUDD, J.; STERN, K.; ISENSEE, S. Low vs. high-fidelity prototyping debate. *Interactions*, ACM, New York, NY, USA, v. 3, n. 1, p. 76–85, jan. 1996. ISSN 1072-5520. Citado na página 16.

TENDELOO, Y. V.; VANGHELUWE, H. An evaluation of devs simulation tools. *Simulation*, Society for Computer Simulation International, San Diego, CA, USA, v. 93, n. 2, p. 103–121, fev. 2017. ISSN 0037-5497. Disponível em: <https://doi.org/10.1177/0037549716678330>. Citado na página 28.

Walker, T. O.; Tummala, M.; McEachen, J. A system of systems study of space-based networks utilizing picosatellite formations. In: *Proc. of 5th SoSE*. Loughborough, UK: [s.n.], 2010. p. 1–6. Citado na página 30.

Wang, R.; Dagli, C. H. An executable system architecture approach to discrete events system modeling using sysml in conjunction with colored petri net. In: *2008 2nd Annual IEEE Systems Conference*. [S.l.: s.n.], 2008. p. 1–8. Citado na página 17.

WOLNY, S. A runtime model for sysml. *Doctoral College Cyber-Physical Production Systems*, v. 43, 2017. Citado na página 17.

WOLNY, S. et al. Thirteen years of sysml: a systematic mapping study. *Software & Systems Modeling*, May 2019. ISSN 1619-1374. Disponível em: <https://doi.org/10.1007/s10270-019-00735-y>. Citado na página 17.

ZEIGLER, B.; SARJOUGHIAN, H. *Guide to Modeling and Simulation of Systems of Systems*. [S.l.: s.n.], 2017. ISBN 978-3-319-64133-1. Citado 3 vezes nas páginas 28, 29 e 49.

ZEIGLER, B. P.; SARJOUGHIAN, H. S. Devs natural language models and elaborations. In: _____. *Guide to Modeling and Simulation of Systems of Systems*. London: Springer London, 2013. p. 39–62. Citado na página 28.

ZHU, L.; STAPLES, M.; JEFFERY, R. Scaling up software architecture evaluation processes. In: WANG, Q.; PFAHL, D.; RAFFO, D. M. (Ed.). *Making Globally Distributed Software Development a Success Story*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 112–122. ISBN 978-3-540-79588-9. Citado na página 22.