MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2014/10.28.22.44-TDI

BENCHMARKING DE RESILIÊNCIA PARA INFRAESTRUTURAS DE SIMULADORES DE SATÉLITES BASEADAS EM HLA

Denise Nunes Rotondi Azevedo

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Ana Maria Ambrosio, e Marco Paulo Amorim Vieira, aprovada em 28 de novembro de 2014.

URL do documento original:
<http://urlib.net/8JMKD3MGP5W34M/3HAN4D5>

INPE
São José dos Campos
2014

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2014/10.28.22.44-TDI

BENCHMARKING DE RESILIÊNCIA PARA INFRAESTRUTURAS DE SIMULADORES DE SATÉLITES BASEADAS EM HLA

Denise Nunes Rotondi Azevedo

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Ana Maria Ambrosio, e Marco Paulo Amorim Vieira, aprovada em 28 de novembro de 2014.

URL do documento original:
<http://urlib.net/8JMKD3MGP5W34M/3HAN4D5>

INPE
São José dos Campos
2014

Dados Internacionais de Catalogação na Publicação (CIP)

Azevedo, Denise Nunes Rotondi.

Az25b Benchmarking de resiliência para infraestruturas de simuladores de satélites baseadas em hla / Denise Nunes Rotondi Azevedo. – São José dos Campos : INPE, 2014.

xxviii + 265 p. ; (sid.inpe.br/mtc-m21b/2014/10.28.22.44-TDI)

Tese (Mestrado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2014.

Orientadores : Drs. Ana Maria Ambrosio, e Marco Paulo Amorim Vieira.

1. Benchmarking de resiliência. 2. Simulador de satélite. 3. Infraestrutura de simuladores de satélite, HLA. I.Título.

CDU 629.783

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Doutor(a)* em

Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais

Dr. Walter Abrahão dos Santos

Presidente / INPE / São José dos Campos - SP

Dra. Ana Maria Ambrosio

Orientador(a) / INPE / São José dos Campos - SP

Dr. Marco Paulo Amorim Vieira

Orientador(a) / UC.PT / Portugal - PT

Dra. Maria de Fátima Mattiello Francisco

Membro da Banca / INPE / São José dos Campos - SP

Dra. Eliane Martins

Convidado(a) / UNICAMP / Campinas - SP

Dra. Emilia Villani

Convidado(a) / Ita / São José dos Campos - SP

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

Aluno (a): ***Denise Nunes Rotondi Azevedo***

São José dos Campos, 28 de Novembro de 2014

“Nada é permanente, exceto a mudança”.

Heráclito

“Quando os ventos de mudança sopram, umas pessoas levantam barreiras, outras constroem moinhos de vento”.

Érico Veríssimo

A minha mãe, Marina, que sempre foi e é para mim um exemplo de luta, força, generosidade e, acima de tudo, uma fonte inesgotável de incentivo e amor.

AGRADECIMENTOS

Agradeço ao Dr. Marco Vieira que, com competência e objetividade, e mesmo à distância, sempre se fez presente por meio de discussões, sugestões e ideias. Também agradeço a ele pela amizade, apoio e consideração demonstrados em todas as minhas estadas em Coimbra e ao longo desses anos de trabalho. Agradeço à Dra. Ana Maria Ambrosio que me acompanhou, guiou e incentivou desde o início, pelas discussões e sugestões, e pela amizade e coleguismo em todo esse tempo de INPE.

Agradeço a meu marido, Paulo Henrique, pela paciência com as minhas ausências, viagens e por compartilhar comigo as angústias que sempre permeiam tarefas longas e incertas. Agradeço a ele também o incentivo, o apoio incondicional, a crença de que eu chegaria até o final, o carinho e, sobretudo, a generosidade.

A minha família. A minha mãe e a minha filha Lygia pela compreensão com as minhas ausências e com a falta do suporte que seria de mim esperado e, principalmente, pelo incentivo e apoio. A minha irmã Luciana por me encorajar e acreditar, e pela amizade de sempre. Á Kátia pelo estímulo e carinho.

Ao Leandro, pelas dicas e sugestões, por me ouvir e por compartilhar as minhas preocupações e ansiedades. À Luciana, pelas palavras de amizade e de incentivo e pela ajuda em um momento importante do percurso. À Simone que tantas vezes supriu as minhas ausências e à cooperação da Waléria. Ao Rubens Gatto, chefe da Divisão de Sistemas de Solo (DSS), que sempre compreendeu as minhas necessidades e dificuldades, colaborando e me apoiando em todas as situações. E a todos os meus colegas da DSS pelo incentivo e carinho durante toda a trajetória.

Ao Instituto Nacional de Pesquisas Espaciais (INPE), Coordenação de Engenharia e Tecnologia Espaciais, pela oportunidade de desenvolver este trabalho de doutorado e pela confiança depositada.

Acima de tudo, a Deus e às interseções por ter colocado no meu caminho oportunidades e por ter me ajudado sempre na caminhada.

RESUMO

Simuladores de satélite são utilizados como ferramentas de apoio às tarefas de análise de missões espaciais e de verificação, validação e operação de satélites. Usados em diferentes fases de uma missão espacial e em diferentes missões, esses sistemas têm características evolutivas no que tange a mudanças em seus requisitos, inserção de novos modelos e reconfigurações, exigindo uma infraestrutura de simulação que suporte essa evolução, preservando os atributos de dependabilidade e resiliência. Esta tese propõe uma metodologia para a definição, instanciação e representação de benchmarks de resiliência e uma abordagem de benchmarking para avaliar e comparar atributos de resiliência no domínio de infraestruturas de simuladores de satélite baseadas em *High Level Architecture* (HLA), um padrão amplamente utilizado no contexto de desenvolvimento de simuladores. A metodologia proposta inclui o processo de especificação de benchmarks de resiliência, uma linguagem para representação e disseminação desses benchmarks e um *framework* para o desenvolvimento de ferramentas. A aplicabilidade da metodologia e dos elementos definidos na especificação do benchmark está demonstrada por meio da instanciação de dois benchmarks concretos - resiliência e robustez. O primeiro benchmark avalia infraestruturas de simulação HLA relativamente à resiliência dos atributos *latência* e *rendimento* quando essas infraestruturas são expostas a mudanças na escala, em aspectos de uso e na distribuição da simulação. O segundo benchmark, visto aqui como um caso particular de benchmarks de resiliência, demonstra a capacidade e flexibilidade da metodologia e da linguagem de representação na instanciação e execução de benchmarks que têm como objetivo avaliar a robustez das infraestruturas HLA. A implementação dos benchmarks e a execução dos experimentos demonstraram a viabilidade da abordagem comparativa apresentada e os resultados obtidos foram capazes de expressar quantitativamente, de forma simples e eficaz, a resiliência e robustez das infraestruturas HLA avaliadas, fornecendo diretrizes para a escolha de produtos em diferentes contextos de avaliação e cenários de mudanças.

RESILIENCE BENCHMARKING OF SATELLITE SIMULATION INFRASTRUCTURES BASED ON THE HLA STANDARD

ABSTRACT

Satellite simulators are used to support space mission analysis and satellite verification, validation and operation. Used in different phases of a space mission and in different missions, these systems have an evolutionary characteristic, requiring an infrastructure that supports evolution and adaptation, while preserving resilience and dependability attributes. This work presents a methodology to define, instantiate and represent benchmarks for assessing and comparing resilience attributes of satellite simulation infrastructures based on the High Level Architecture (HLA), a standard widely used in the context of simulators development. In practice, the methodology includes the benchmark specification process, a language for benchmark representation and dissemination, and a framework for developing benchmark tools. The applicability of the presented methodology and benchmark elements is demonstrated through the instantiation of a resilience benchmark and a robustness benchmark. The resilience benchmark is used to evaluate HLA infrastructures in relation to the resilience of performance attributes when the infrastructure is exposed to changes in simulation scale, usage and distribution. The robustness benchmark demonstrates the flexibility of the methodology and representation language in instantiation and conducting benchmarks that aim to assess the infrastructures robustness, a special case of resilience. The benchmark implementation and the experiments demonstrated the feasibility of the proposed benchmarking approach. The experiments results portrayed, in a simple and effective way, the resilience and robustness of the evaluated infrastructures providing guidelines for choosing HLA products in different evaluation contexts and change scenarios.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

Pág.

AFAP	Tão rápido quanto possível *(as-fast-as-possible)*
AIV	Montagem, Integração e Verificação
AMBER	*Assessing, Measuring and Benchmarking Resilience*
API	Interface de Programação de Aplicações
ATIFS	Ambiente de Testes baseado em Injeção de Falhas de Software
ATV	*Automated Transfer Vehicle*
BD	Banco de Dados
BNF	*Backus-Naur Form*
BT	Alvo do Benchmark (*Benchmark Target*)
CBERS	Satélite Sino-Brasileiro de Recursos Terrestres
CORBA	*Common Object Request Broker Architecture*
CRASH	Catastrófica, Reinício, Aborto, Silenciosa e Oculta
CRC	Componente Central de Infraestrutura de Execução
DBENCH	*Dependability Benchmarking*
DDS	*Data Distribution Service for Real-time Systems*
DEV	*Discrete Event System Specification*
DIS	*Distributed Interactive Simulation*
DMSO	*Defense Modeling and Simulation Office*
DOD	Departamento de Defesa Americano
DTD	Definição de Tipo de Documento *(Document Type Definition)*
EBNF	*Extended Backus-Naur Form*
ECSS	*European Cooperation on Space Standardization*
EFM	*Electrical Functional Model*
EML	*Election Markup Language*
ESA	Agência Espacial Europeia
FBV	*Functional Verification Bench*
FES	*Functional Engineering Simulator*
FOM	Modelo de Objeto de Federação
FVT	*Functional Validation Testbench*
GQM	Objetivo/Questão/Métrica (Goal/Question/Metric)
G-SWFIT	*Generic Software Fault Injection Technique*

GUI	Interface Gráfica de Usuário
HITL	Hardware na Malha
HLA	*High Level Architecture*
HRT	Tempo Real Restrito (*Hard Real Time*)
IEC	Comissão Eletrotécnica Internacional
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IIOP	*Internet InterORB Protocol*
INPE	Instituto Nacional de Pesquisas Espaciais
IP	*Internet Protocol*
ISO	Organização Internacional para Padronização
ISS	Estação Espacial Internacional
LABSIA	Laboratório de Sistemas Inerciais para Aplicação Aeroespacial
LAN	Rede Local (*Local Area Network*)
LRC	Componente Local de Infraestrutura de Execução
LOO	Linguagem Orientada a Objetos
M&C	Monitoramento e Controle
M&S	Simulação e Modelamento
MOM	Modelo de Objeto de Gerenciamento
MPS	*Mission Performance Simulator*
NASA	Administração Nacional da Aeronáutica e do Espaço
OBC	Computador de Bordo
OBSW	Software de Bordo
ODC	Classificação Ortogonal de Defeitos
OLTP	*On-Line Transaction Processing*
OMT	Modelo de Objeto
OO	Orientação a Objetos
POSIX	Interface Portável entre Sistemas Operacionais
PUS	*Packet-Utilization Standard*
QoS	Qualidade de Serviço
RBDL	Linguagem de Descrição de Benchmark de Resiliência
RID	Dados de Inicialização da Infraestrutura de Execução
RTI	Infraestrutura de Execução (*Runtime Infrastructure*)
SGB	Sistema Gerenciador de Benchmark
SBT	*Hybrid System Testbed*

SCD	Satélite de Coleta de Dados
SCS	*System Concept Simulator*
SOM	Modelo de Objeto de Simulação
SMP	*Simulation Model Platform*
SPEC	*Standard Performance Evaluation Corporation*
SRT	Tempo Real Flexível (*Soft Real Time*)
SQL	Linguagem de Consulta Estruturada (*Structured Query Language)*
SUB	Sistema Sob Benchmark
SVF	*Software Validation Facility*
TCP	*Transmission Control Protocol*
TENA	*Test and Training Enabling Architecture*
TP	Processamento de Transações
TPC	*Transaction Processing Performance Council*
TT&C	Telecomunicação de Serviços
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
V&V	Verificação e Validação
WAN	*Rede de longa distância* (*Wide Area Network*)
XBRL	*EXtensible Business Reporting Language*
XSD	*XML Schema*
XML	*EXtensive Markup Language*
XSL	*EXtensible Stylesheet Language*
XSLFO	*EXtensible Stylesheet Language Format Output*
XSLT	*EXtensible Stylesheet Language Transformation*

SUMÁRIO

Pág.

1 INTRODUÇÃO

Sistemas espaciais são complexos, operam em ambientes hostis, não completamente dominados pelo conhecimento humano, e têm manutenção difícil, onerosa e muitas vezes impraticável. Por essas razões, o ciclo de vida de um sistema espacial, da sua concepção à sua operação, exige processos de engenharia e ferramentas de verificação e validação que assegurem um grau adequado de confiabilidade (EICKHOFF, 2009).

Simuladores são ferramentas computacionais utilizadas no processo de engenharia como forma de reduzir custos e prazos, substituindo a construção de modelos físicos e de protótipos de equipamentos (MARTIN; CARVALHO, 2005). Essas ferramentas são amplamente usadas no desenvolvimento de sistemas espaciais como apoio às tarefas de análise de alternativas de projeto, verificação e validação de equipamentos, treinamento de operadores, validação de procedimentos operacionais, etc. A *National Aeronautics and Space Administration* (NASA), por exemplo, utiliza simuladores desde o programa Apollo de 1966.

O Brasil, por meio do Instituto Nacional de Pesquisas Espaciais (INPE), desenvolve sistemas espaciais desde a década de 80, tendo lançado seu primeiro satélite em 1993. Em apoio às missões desenvolvidas pelo INPE, vêm sendo utilizados e propostos simuladores com diferentes escopos. Relativamente aos Satélites de Coleta de Dados (SCD-1 e SCD-2), Perondi (1987, citado por Hoffman e Perondi, 2010) apresentou o uso de simuladores para análise de potência e Orlando et al. (1992) descreveram o uso de simuladores para a validação de operações de solo. Ambrosio et al. (2006) apresentaram um simulador baseado em uma arquitetura comum e reutilizável para o Satélite Sino-Brasileiro de Recursos Terrestres (CBERS) e detalharam a experiência do INPE na construção de simuladores operacionais. O trabalho de Tominaga et al. (2008) propôs o uso de simuladores para a verificação de planos de operação.

Embora simuladores sejam muito utilizados na área espacial, tal utilização só se justifica se o esforço empregado no seu desenvolvimento puder ser diluído por meio do reúso de um mesmo simulador ou de partes de um simulador em diferentes missões (HOFFMAN; PERONDI, 2010; SEBASTIAO et al., 2008). Entretanto, na prática, simuladores vêm sendo empregados na área espacial de forma independente e não coordenada (ECSS, 2010).

Segundo a *European Cooperation on Space Standardization* (ECSS) (2010), ainda que cada missão e cada fase de uma missão espacial tenham necessidades específicas, o uso de simulação de forma sistêmica e coerente ao longo do ciclo de vida de um projeto espacial, utilizando políticas de reúso sempre que possível, viabiliza o desenvolvimento de simuladores, reduzindo custos, prazos e riscos. O artigo de Eickhoff et al. (2007) e a norma *System Modelling and Simulation* (ECSS, 2010) apresentam filosofias nas quais o mesmo ambiente de simulação e modelos simulados padronizados são reusados entre várias fases de uma missão espacial, enquanto modelos específicos de subsistemas ou de equipamentos vão sendo substituídos pelos seus equivalentes físicos ou por outros modelos mais adequados à cada fase específica.

Padrões para desenvolvimento de simuladores foram criados com o objetivo de promover o reúso de modelos e da infraestrutura de simulação. Dentre esses padrões destacam-se o *Simulation Model Platform* (SMP) e a *High Level Architecture* (HLA), esta última centrada em simulação distribuída.

O SMP é um padrão para construção de simuladores definido pela Agência Espacial Europeia (ESA), por meio da *European Cooperation on Space Standardization*, e está descrito em ECSS-E-TM-40-07E (ECSS, 2011). O SMP especifica um *framework* para ambientes de simulação e define os serviços oferecidos por esse ambiente e a interface entre o ambiente e os modelos simulados. O SMP tem como focos principais a independência de plataforma e a reusabilidade dos modelos simulados e do ambiente de simulação. Essa abordagem facilita a integração de modelos desenvolvidos por diferentes

grupos ou empresas, aumentando a produtividade no desenvolvimento de simuladores (ECSS, 2011).

A *High Level Architecture* (HLA), desenvolvida pelo Departamento de Defesa Americano (DOD), é uma arquitetura de software distribuída para criação de simuladores computacionais a partir de componentes de simulação (modelos) (MÖLLER et al., 2005). O padrão HLA especifica serviços que devem ser oferecidos pela Infraestrutura de Execução *(Runtime Infraestructure* - RTI), as regras a serem seguidas pelos simuladores e um padrão de documentação para descrição de modelos (IEEE, 2001). A HLA não é uma implementação, mas sim uma definição de interfaces e regras para a construção de simuladores distribuídos que visa promover reutilização e interoperabilidade. Uma RTI HLA implementada em conformidade com o padrão pode ser vista como um *middleware* que provê um conjunto de serviços para a simulação, tais como comunicação entre modelos, sincronização, gestão e controle de tempo, entre outros.

O padrão HLA, amplamente utilizado na área militar, tem sido aplicado a outras áreas, tais como manufatura e aeronáutica, sendo atualmente um padrão mantido pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE). O HLA também tem sido pesquisado como solução para simulações distribuídas na área aeroespacial (NOULARD et al., 2009; REIS, 2009; TRIVELATO, 2004) e vem sendo prospectado pelo INPE como alternativa para o desenvolvimento de simuladores de satélites.

No contexto desta tese, o trabalho de Azevedo et al. (2012), avaliou o potencial de reúso dos simuladores atualmente em desenvolvimento no INPE usando como base a classificação de simuladores apresentada no memorando técnico ECSS (2010). A avaliação mostrou que, caso tivesse sido adota uma política que favorecesse o intercâmbio de modelos e o reúso da infraestrutura de simulação, aproveitando as comunalidades entre os diferentes simuladores, a duplicação de esforços na construção dos mesmos teria sido reduzida em aproximadamente 50%. O trabalho concluiu que o uso de padrões de desenvolvimento de simuladores, tais como o SMP e o HLA, devem ser

considerados para futuros projetos do INPE com o objetivo de reduzir o retrabalho e agregar esforços complementares.

A presente tese explora o uso de um padrão de infraestrutura de simulação distribuída, nomeadamente o padrão HLA, e se atém a aspectos de dependabilidade e resiliência no âmbito de reúso de simuladores de satélite conforme apresenta a seção seguinte.

1.1. Motivação do Trabalho

Eickhoff (2009) argumenta que embora simuladores sejam usados na verificação e teste de subsistemas críticos do satélite, o fato desses subsistemas não serem testados exclusivamente por meio de simuladores faz com que o grau de criticidade dessas ferramentas não seja considerado alto. Entretanto, o uso de simuladores durante a fase de operação do satélite para a validação de manobras críticas, testes de novas versões do software de bordo e apoio ao diagnóstico de anomalias, aumenta consideravelmente o grau de criticidade dessas ferramentas. O mesmo ocorre nas fases de análise, quando os resultados da simulação são usados como suporte a importantes decisões de projeto. Especialmente nesses contextos, simuladores de satélite devem contemplar requisitos de confiabilidade, robustez e tolerância a falhas.

A adoção de padrões no desenvolvimento de simuladores promove o reúso de modelos, mas levanta questões relativas à dependabilidade das ferramentas de simulação. De um lado, a reutilização de modelos tem impactos positivos sobre atributos de confiabilidade, uma vez que promove qualidade e estabilidade por meio do uso de uma infraestrutura comum e modelos já testados; de outro, pode ter impactos negativos ao promover a permanente integração de novos modelos desenvolvidos por equipes diferentes e em contextos diferentes.

Além disso, o uso de simuladores de satélite ao longo das várias fases do ciclo de vida de uma missão espacial e em diferentes missões faz com estes sistemas tenham características intrinsecamente evolutivas e estejam sujeitos a um conjunto de mudanças contínuas, como por exemplo, alterações no âmbito da escala dos sistemas (número de modelos, número de satélites), dos

objetivos da missão (complexidade), das tecnologias empregadas (evolução das plataformas de hardware e software) ou dos requisitos funcionais e não funcionais (requisitos de fidelidade dos modelos, de distribuição da simulação, etc.).

Desta forma, ao considerarmos o uso de infraestruturas padronizadas no desenvolvimento de simuladores, os requisitos de dependabilidade desses sistemas e o seu caráter evolutivo, somos levados a considerar a relevância da **resiliência** das infraestruturas de simulação.

Segundo Laprie (2008) e Bondavalli et al. (2009), a resiliência pode ser vista como a "*persistência da dependabilidade em face de mudanças*", sendo dependabilidade um conceito que indica a "*capacidade de um sistema em oferecer serviços em que se possa justificadamente confiar*" e que pode ser traduzido por um conjunto de atributos, tais como confiabilidade, disponibilidade, segurança (*safety*), integridade, *manutenabilidade*, etc. (Aviziens, 2004). Desta forma, um sistema resiliente deve ser capaz de acomodar mudanças, mantendo os atributos de dependabilidade na presença das mesmas.

Na prática, o uso de simuladores como ferramenta de verificação, validação e apoio a sistemas críticos traz requisitos de dependabilidade, mas o seu caráter evolutivo e sujeito a mudanças pode dificultar a garantia do atendimento desses requisitos. Nesse contexto, avaliar as infraestruturas de simulação antes de seu uso é de extrema importância para verificar, garantir e atestar que essas infraestruturas terão a capacidade de acomodar mudanças, mantendo níveis satisfatórios no atendimento dos atributos de dependabilidade, tolerando e isolando falhas, de forma a não comprometer e afetar o comportamento dos modelos ou os resultados da simulação.

Essa avaliação deve permitir, por exemplo, que se responda a um conjunto de perguntas pertinentes nesse domínio, nomeadamente:

- Qual é o impacto da inserção de um novo modelo desenvolvido por outra equipe em um simulador existente?
- O nível de dependabilidade de uma infraestrutura de simulação é mantido quando há variações, tais como: aumento ou diminuição no número de modelos, variação na carga dos modelos, mudanças no número de parâmetros intercambiados entre modelos, etc.?
- Qual a viabilidade de se converter uma simulação centralizada em uma simulação distribuída preservando o nível de dependabilidade?
- Uma infraestrutura de *código aberto* poderia ser utilizada em um simulador com requisitos de desempenho e dependabilidade previamente definidos? Qual seria a melhor implementação nesse contexto?

Visando oferecer suporte para que as questões levantadas pudessem ser adequadamente respondidas, este trabalho pesquisou abordagens que permitem avaliar e comparar, de forma sistemática e padronizada, a resiliência das infraestruturas de simulação. Tais abordagens, denominadas *benchmarks*, consistem genericamente em um processo padronizado para avaliar e comparar diferentes sistemas de um mesmo domínio relativamente a características específicas como definido em Bondavalli et al. (2009).

1.2. Contribuições do Trabalho

Levando em consideração o uso de infraestruturas padronizadas que contemplam a distribuição da simulação, promovem o reúso de modelos e a interoperabilidade de simuladores, a contribuição principal deste trabalho foi **a definição de uma abordagem de benchmarking que permite avaliar e comparar infraestruturas de simulação baseadas no padrão HLA no que tange à sua resiliência.** Tal abordagem auxilia na escolha de produtos oriundos de diferentes fornecedores e, também, na avaliação da continuidade da estabilidade e confiabilidade de simuladores já utilizados diante da inclusão de novos modelos, de reconfigurações, de mudanças tecnológicas, garantindo a manutenção dos resultados da simulação.

Este trabalho também contribuiu com a **definição de uma metodologia para a especificação de benchmarks de resiliência no domínio de infraestruturas de simuladores** que contempla todas as etapas da especificação de um benchmarking de resiliência, desde a definição dos seus principais elementos até a representação e instanciação de benchmarks concretos. Para além disso, foi definida uma **linguagem de descrição** (*Resilience Benchmark Description Language - RBDL*) que é uma alternativa para a representação e disseminação de benchmarks de resiliência no domínio de infraestruturas de simuladores de satélites baseadas em HLA.

Dois benchmarks concretos, instanciados por meio da metodologia proposta, avaliaram três infraestruturas HLA (uma comercial e duas de *código aberto*). O primeiro benchmark considerou a evolução dos simuladores no que tange a mudanças na escala dos objetos simulados (número de modelos, volume de dados intercambiados entre modelos, etc.) e avaliou a resiliência de atributos de desempenho (Latência e Rendimento). Já o segundo benchmark avaliou a robustez da infraestrutura de simulação. A instanciação dos benchmarks concretos demonstrou o uso da metodologia proposta, da RBDL e dos elementos especificados na abordagem de benchmarking definida; e os resultados obtidos por meio da execução dos experimentos demonstraram a viabilidade do uso dos benchmarks de resiliência na avaliação e escolha de infraestruturas HLA.

Neste trabalho nós consideramos que as questões relativas às mudanças serão sempre recorrentes e constantes e definimos um paradigma que parte da possibilidade de instanciação de novos benchmarks, visando facilitar a avaliação de novas mudanças e o uso desses benchmarks em diferentes contextos.

A metodologia e o benchmark especificado visam atender as necessidades futuras em modelagem e simulação do INPE, bem como ser uma contribuição na área de sistemas computacionais que requerem alta confiabilidade e resiliência frente, não apenas a falhas, mas também a mudanças.

1.3. Organização do Documento

Neste primeiro capítulo foram apresentados desafios no que concerne à construção de simuladores e seu uso como ferramentas de apoio a projetos da área espacial. O documento completo está estruturado em mais sete capítulos conforme descrito a seguir.

O Capítulo 2 contextualiza o domínio de aplicação do trabalho e descreve as fases de uma missão espacial, os tipos de simuladores usados nas diferentes fases e detalha o padrão *High Level Architecture* (HLA).

O Capítulo 3 apresenta conceitos fundamentais de dependabilidade, resiliência e benchmarking, descrevendo o estado da arte nestes tópicos. O capítulo apresenta, ainda, trabalhos relacionados a benchmarks no domínio de infraestrutura HLA e a benchmarks de resiliência em diferentes domínios.

O Capítulo 4 descreve a metodologia proposta para a especificação de benchmarks de resiliência, apresentando as etapas e passos a serem executados, a sua ordenação, interdependência e inter-relação.

O Capítulo 5 apresenta a nossa abordagem na especificação de um benchmark de resiliência para infraestruturas de simuladores de satélites baseadas em HLA usando a metodologia proposta. São apresentados, também, exemplos dos elementos resultantes dessa especificação.

O Capítulo 6 define a Linguagem de Descrição de Benchmark de Resiliência (RBDL) e propõe uma arquitetura para a ferramenta de benchmarking, mostrando o uso da linguagem e dos arquivos RBDL como insumo para a implementação do ferramental, para a execução dos experimentos e para a análise dos resultados do benchmark.

O Capítulo 7 apresenta dois benchmarks concretos: um benchmark de resiliência para infraestruturas de simulação baseadas em HLA e um segundo benchmark que exemplifica o uso da metodologia e da RBDL no contexto de

benchmarks de robustez. Neste capítulo também são detalhados e analisados os resultados obtidos por meio da execução dos experimentos de benchmark.

O Capítulo 8 sumariza os principais resultados do trabalho, discute as contribuições e propõe direcionamentos e perspectivas para futuros trabalhos.

Finalmente, o Apêndice A apresenta detalhes acerca da norma HLA e o Apêndice B complementa o Capítulo 5 apresentando os elementos definidos pela especificação de benchmarking no contexto de infraestruturas HLA.

2 SIMULADORES DE SATÉLITE

Neste capítulo são apresentadas as filosofias de reúso preconizadas pelos trabalhos de Eickhoff (2007, 2009) e pelo memorando técnico ECSS (2010), bem como as classes e tipos de simuladores de satélite propostos por essas filosofias, enfatizando o aspecto evolutivo das mesmas. É apresentado, também, o padrão de infraestrutura de simulação *High Level Architecture* (HLA) e discutido o uso de simulação distribuída no âmbito de simuladores de satélites.

Vale salientar que, embora simuladores sejam utilizados na área espacial de forma geral, este trabalho está centrado no estudo de simuladores e infraestruturas de simulação no âmbito da engenharia de satélites artificiais.

2.1. Satélites Artificiais

Satélites artificiais podem ser vistos logicamente como um conjunto de subsistemas integrados divididos em: (i) *carga útil*, composta pelos instrumentos relacionados à missão específica, tais como câmeras imageadoras em satélites de sensoriamento remoto, transmissores em satélites de telecomunicação, etc.; e (ii) *plataforma*, que sustenta a operação das cargas úteis e compreende os subsistemas de estrutura mecânica, controle térmico, controle de órbita e atitude, propulsão, suprimento de energia, gestão de bordo e telecomunicação de serviços. A Figura 2.1 ilustra a arquitetura típica de um satélite (ETE, 1999).

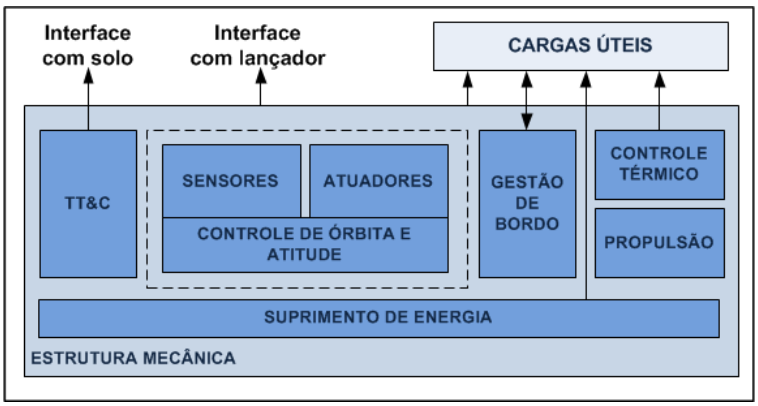


Figura 2.1 - Subsistemas – Satélite

O subsistema de gestão de bordo, composto por computador de bordo e software embarcado, bem como o subsistema de controle de órbita e atitude são críticos por suas características funcionais e por sua complexidade. O computador e o software de bordo são responsáveis pela operação e manutenção do satélite em órbita, já que têm a incumbência de receber telecomandos e encaminhá-los aos equipamentos do satélite, de formatar e enviar telemetrias para as estações terrenas, entre outras. O subsistema de controle de órbita e atitude é o responsável pelo posicionamento do satélite permitindo o apontamento correto tanto das cargas úteis, quanto dos painéis solares responsáveis pelo fornecimento de energia ao satélite. O desenvolvimento desses dois subsistemas críticos são especialmente apoiados pelo uso de simuladores como ferramentas de verificação e validação (EICKHOFF, 2009).

2.2. Simuladores de Satélite no Ciclo de Vida de uma Missão Espacial

A NASA e a ESA dividem o ciclo de vida de projetos espaciais em fases conforme descrito nas normas NPR7120.7 (NASA, 2008a) e ECSS-M-ST-10C (ECSS, 2009), respectivamente. As fases definidas por ambas as agências são muito similares e neste trabalho é utilizada como referência a norma ECSS (2009) que define as seguintes fases: análise da missão e identificação de necessidades (fase 0), análise de viabilidade (fase A), projeto preliminar (fase B), projeto detalhado (fase C), qualificação e produção (fase D), utilização (fase E) e descarte (fase F).

O artigo de Eickhoff et al. (2007) e o memorando técnico ECSS-E-TM-10-21A (ECSS, 2010) preconizam o reúso de modelos padronizados e da infraestrutura de simulação na construção de simuladores que serão empregados ao longo das fases de uma missão espacial. Eickhoff (2009) propõe uma tecnologia conhecida como Desenvolvimento e Verificação Baseado em Modelos (*Model-base Development and Verification*) e o memorando técnico ECSS (2010) especifica uma abordagem conhecida como Projeto Virtual de Plataformas Orbitais (*Spacecraft Virtual Design*). Segundo o ECSS (2010), essa abordagem visa desenvolver um modelo virtual que possa ser usado desde o início da

missão nas atividades de especificação, antecipando as etapas de verificação e validação, e que possa evoluir para apoiar as demais fases.

As duas filosofias propõem um modelo baseado inteiramente em simulação e definem um conjunto de tipos de simuladores que podem ser classificados, de acordo com seus usos, dentro de macroclasses de simuladores: *análise e suporte ao projeto, integração e teste*, e *operacional*.

É importante salientar que o Projeto Virtual de Plataformas Orbitais (ECSS, 2010) é utilizado como principal referência neste trabalho para a definição de elementos do benchmark proposto. Nas próximas subseções deste capítulo são detalhadas as classes de simuladores e os tipos de simuladores definidos para cada uma dessas classes com o objetivo de apresentar o aspecto evolutivo dessa filosofia, bem como, de apresentar as principais características de cada classe de simulador, tendo em vista a compreensão dos seus requisitos de resiliência e dependabilidade. Como o trabalho de Eickhoff (2009) propõe filosofia similar à proposta pelo memorando técnico ECSS (2010), sempre que pertinente, apresentaremos paralelamente a sua visão.

2.2.1. Simuladores para Análise da Missão Espacial e Suporte ao Projeto

Nas fases que vão de 0 a C, os simuladores são construídos para apoiar a análise e especificação da missão, de acordo com as diferentes disciplinas: órbita, atitude, térmica, elétrica, comunicação com estações de solo, etc., e, também, para apoiar as atividades de projeto, auxiliando na definição da *linha de base*[1] da missão, na análise de alternativas e conceitos de alto nível, nas investigações de relações custo-benefício, nas análises de viabilidade técnica e na especificação funcional do sistema.

[1] Conjunto de informações que descreve exaustivamente uma situação em um dado momento ou em um determinado intervalo de tempo. Uma linha de base é geralmente utilizada como referência para análise e comparação com as subsequentes evoluções da informação. (ECSS, 2012).

Os tipos de simuladores que cobrem esta fase são (ECSS, 2010): (i) *System Concept Simulator (SCS)*, usado nas fases iniciais (0 e A), executa modelos de tempo não real, normalmente de baixa fidelidade[2], para apoio à concepção da missão e para definição dos requisitos de alto nível do sistema; (ii) *Mission Performance Simulator (MPS*)[3], centrado nos interesses dos usuários finais, executa análises que levam em consideração as cargas úteis em relação aos requisitos da missão, apoiando, sob esse ponto de vista, análises de custo-benefício (*trade off*) e avaliação de tecnologias e configurações, possibilitando o desenvolvimento antecipado de algoritmos de geração de produtos; (iii) *Functional Engineering Simulator (FES),* ferramenta de apoio à validação do projeto, é usado na consolidação dos requisitos de sistema, na validação dos principais algoritmos de cálculo de órbita e atitude, em análises de alternativas de projeto, na verificação do projeto preliminar e detalhado, e na verificação do desempenho do sistema por meio de análises específicas, por exemplo, pior caso, perturbações, etc.; (iv) *Functional Validation Testbench (FVT)*, utilizado para análise do desempenho do sistema e para teste e verificação dos subsistemas críticos da missão cujo projeto e desenvolvimento começam antecipadamente dentro da mesma, tendo como foco os elementos críticos prototipados; e (v) *Software Validation Facility (SVF),* usado para apoiar a validação e integração do software de bordo (OBSW) com o computador de bordo (OBC), permite avaliações de desempenho, testes de robustez, avaliação de mecanismos de falha e validação das atividades de manutenção do software embarcado; pode ser totalmente configurados em software, tendo o computador de bordo emulado ou utilizar o computador de bordo real em uma configuração com o hardware na malha da simulação (*hardware-in-the-loop*).

Segundo Eickhoff (2009), nas fases 0 e A são utilizados principalmente simuladores específicos como apoio a análises em diferentes disciplinas, bem

[2] É o grau em que a representação dentro de uma simulação é semelhante a um objeto do mundo real (DOD, 2011).
[3] O *Mission Performance Simulator* é também conhecido como simulador fim-a-fim (*end-to-end*).

como, ferramentas comerciais e bibliotecas especiais para engenharia de controle e dinâmica do sistema. No entanto, na sua visão, nessas fases nem sempre são utilizados simuladores que expressem o comportamento funcional do sistema e que exijam modelos padronizados ou uma infraestrutura de simulação geral que os sustente.

Na linha evolutiva proposta por Eickhoff (2009), nessa macrofase, são utilizados o *Functional Verification Bench (FBV)* que corresponde ao *FES* da ECSS, o *Software Verification Facility* (SVF) que equivale à configuração em software do *SVF* da ECSS e o *Hybrid System Testbed* (SBT) equivalente ao *SVF* da ECSS na sua configuração com o computador de bordo real.

2.2.2. Simuladores para Integração e Teste

Durante a fase D, a qual inclui construção, qualificação, integração e aceitação dos satélites, os simuladores são utilizados para substituir os equipamentos reais enquanto os mesmos ainda não estão disponíveis e para simular o ambiente e a dinâmica do satélite, possibilitando a antecipação da preparação das campanhas de verificação e validação, o desenvolvimento de procedimentos de teste e a integração incremental do satélite. Esses simuladores contemplam mais diretamente o hardware na malha da simulação, têm requisitos de malha fechada[4], tempo real, e consideram estímulos ambientais, o recebimento de telecomandos e a geração de telemetrias. Na fase D, também são utilizados simuladores para a validação dos sistemas de solo.

Segundo o memorando técnico ECSS (2010), os simuladores utilizados nessas fases são: (i) *Spacecraft Assembly, Integration and Verification Facility* (*AIV)*, no qual são simulados os equipamentos do satélite e/ou equipamentos de testes ainda não disponíveis e os comportamentos do satélite que não podem ser fisicamente reproduzidos em solo, visando a preparação das campanhas

[4] Um controlador de malha fechada utiliza o *feedback* de um sistema dinâmico para controlar seus estados ou saídas.

de V&V e a antecipação da integração do satélite; e (ii) *Ground System Test Simulator,* usado como apoio para a validação dos equipamentos da estação terrena, do sistema de controle da missão, bem como, para validar a compatibilidade entre os sistemas de solos e o satélite.

O simulador AIV possui um conjunto de comunalidades com o SVF podendo conter modelos comuns e padronizados, e poderá evoluir para ser utilizado na fase operacional. O *Ground System Test Simulator* deve ser mantido e usado durante toda a fase operacional do satélite.

Para essa fase, Eickhoff (2009) apresenta o *Electrical Functional Model* (EFM) no qual os equipamentos faltantes, a dinâmica e o ambiente espacial são simulados para permitir a campanha de V&V e a integração de equipamentos. Este simulador é equivalente ao AIV da ECSS.

2.2.3. Simuladores para Operação de Satélites

Na fase E são usados Simuladores Operacionais para treinamento de operadores, para validação de operações de contingência do satélite e para auxiliar na solução de problemas e na manutenção dos satélites durante a operação, apoiando a análise de falhas e anomalias. Esses simuladores devem representar o comportamento do satélite com alta fidelidade de forma que o comportamento das telemetrias simuladas seja observado pelos operadores de forma indistinguível do comportamento das telemetrias no satélite real (tanto quando possível). Devem, também, modelar as estações terrenas e os *links* com o satélite de forma a poderem ser controlados pelo sistema de controle em solo.

Para essa fase, a proposta de Eickhoff (2009) indica o uso do *Spacecraft Simulation for Operations Suppor*t, que tem os mesmos objetivos do Simulador Operacional da ECSS.

2.2.4. Infraestrutura de Simulação e Filosofia de Reúso

Segundo o ECSS (2010), o uso de infraestruturas de simulação que tenham características genéricas e reutilizáveis é essencial para que uma política de reúso de modelos possa ser implantada. Em linhas gerais, uma infraestrutura de simulação compreende as funções genéricas necessárias para controlar a simulação, executar modelos, gerenciar o tempo, interagir com sistemas externos e gravar informações da simulação, incluindo estados, *logs* de execução, etc.

A Figura 2.2 apresenta um subconjunto dos tipos de simuladores definidos pelo memorando técnico ECSS (2010) com o objetivo de ilustrar o aspecto adaptativo e evolutivo da filosofia proposta pelo memorando e de demonstrar o uso de uma infraestrutura de simulação transversal ao conjunto de simuladores.

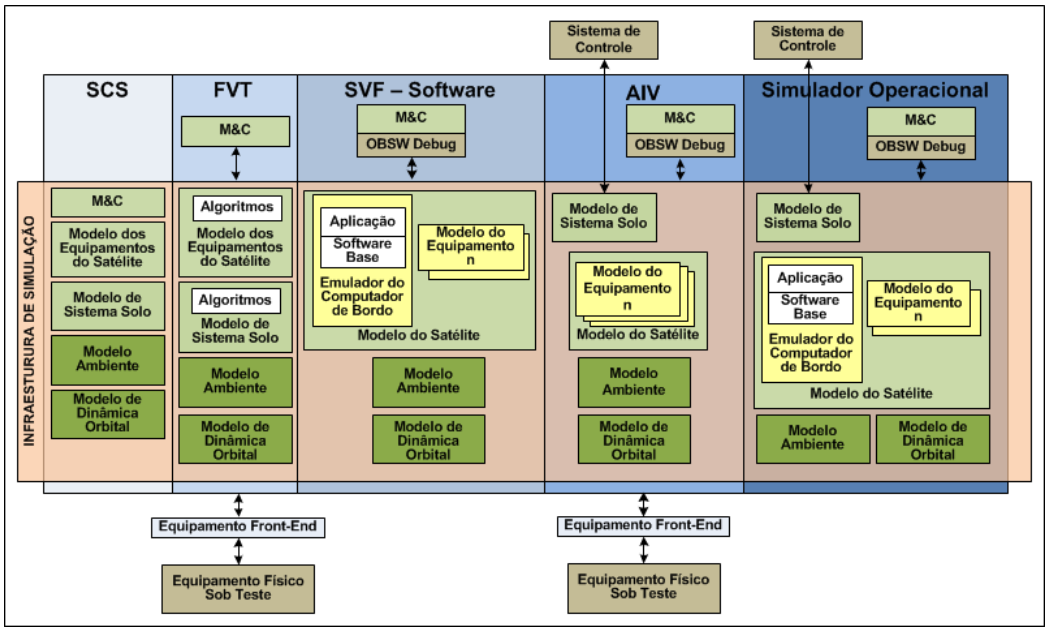


Figura 2.2 - Aspecto Evolutivo - Simuladores de Satélite.

Fonte: adaptado do memorando técnico ECSS (2010).

Na Figura 2.2 é possível observar que alguns modelos simulados são reusados ao longo da linha de evolução dos simuladores (p. ex. modelos de ambiente e da dinâmica orbital), alguns modelos evoluem (p. ex. modelo de sistema de solo, modelo do satélite) e alguns vão sendo substituídos pelos equipamentos reais em uma configuração de simulação com hardware na malha. É

importante notar que a infraestrutura de simulação é comum ao conjunto de simuladores.

Uma vez que é objetivo deste trabalho avaliar a resiliência de infraestruturas de simulação, é necessário identificar o que caracteriza mudanças entre os diversos tipos de simuladores no que tange a configurações e requisitos gerais. Nas filosofias apresentadas, os simuladores evoluem ou se adaptam às necessidades específicas de cada fase da missão, no entanto, é possível identificar um conjunto de características que diferenciam os tipos de simuladores.

Relativamente a características relevantes dos simuladores, podemos citar:

- **Repetibilidade**: indica a necessidade de obtenção dos mesmos resultados quando se realiza o mesmo exercício de simulação. A repetibilidade é um requisito importante em simuladores de análise quando se espera resultados precisos e coerentes, mas pode ser menos relevante em simuladores para treinamento de operadores, por exemplo.

- **Passo da Simulação**: característica do passo de simulação relativamente ao tempo. Alguns tipos de simuladores devem ter a capacidade de serem executados tão rápido quanto possível (*as-fast-as-possible* - AFAP), sendo esse o caso, por exemplo, de um simulador de análise por meio do qual, muitas vezes, são simulados vários meses ou anos da operação do satélite; outros requerem tempo real flexível (*soft real-time* - SRT), como os simuladores operacionais que simulam o satélite em órbita. Os simuladores com hardware na malha exigem tempo real restrito (*hard real-time* - HRT*)*, ou seja, são sensíveis a desvios no tempo.

- **Configuração**: define a configuração geral da malha de simulação. Um simulador pode ter todos os modelos simulados em software ou pode ter modelos que se comuniquem com equipamentos físicos (*hardware-in-the-loop* - HITL). Por exemplo, nas fases iniciais da missão, quando o simulador é usado para analisar requisitos, validar o projeto ou testar algoritmos de

controle, tipicamente todos os modelos são simulados (configuração em software); na fase de produção, os modelos simulados vão sendo substituídos à medida que os equipamentos vão sendo fabricados.

- **Chamadas assíncronas**: indicam se os simuladores aceitam ou não o envio de telecomandos. Os simuladores podem ter o passo de simulação orientado a dados, no qual, em uma sequência de execução de modelos, a saída de dados de um modelo é a entrada de dados do modelo subsequente (orientando o passo de simulação); ou orientado a eventos, no qual um passo de simulação e o transcorrer do tempo está associado a um evento. No caso de simulação de satélites, os simuladores da fase de análise ou simuladores utilizados para teste dos algoritmos de controle têm tipicamente os passos de simulação orientados a dados, já simuladores usados em outras fases, também aceitam o envio de telecomandos que são chamadas assíncronas com efeitos nos parâmetros da simulação.

A Tabela 2.1 resume as características relevantes dos simuladores no contexto desta pesquisa, apresenta a correspondência das propostas de Eickhoff (2009) e do memorando técnico ECSS (2010), bem como situa os simuladores nas diferentes fases da missão.

Tabela 2.1 - Simuladores ECSS (2010) x Eickhoff (2009)

Simuladores- ECSS	Simuladores - Eickhoff	Repetibilidade	Passo de Simulação (1)	Configuração (2)	Chamadas Assincronas
System Concept Simulator (SCS)		Sim	AFAP	SW	Não
Mission Performance Simulator (MPS ou End-to-End)		Sim	AFAP	SW	Não
Functional Engineering Simulator (FES)	Functional Verification Bench (FBV)	Sim	AFAP	SW	Não
Functional Validation Test Bench Simulator (FVT)		Não	HRT	SW/ HITL	Não
Software Validation Facility (SVF)	Software Verification Facility (SVF)	Sim	HRT	SW	Sim
	Hybrid System Testbed			HITL	Sim
Spacecraft AIV Facility (SVV)	Electrical Functional Model (EFM)	Não	HRT	SW/ HITL	Sim
Ground Segment Test Simulator		Não	HRT	SW/ HITL	Sim
Training Simulator		Não	SRT	SW	Sim
Operation and Maintenance Simulator		Sim	SRT/ HTR	SW/ HITL	Sim

Análise e Projeto Integração e Teste Operação

(1) AFAP - tão rápido quanto possível (*as-fast-as-possible*), HRT - tempo-real restrito (*hard real time*), SRT - tempo-real flexível (*soft real time*)
(2) SW - configuração em Software; HITL - hardware na malha de simulação (*hardware-in-the-loop*)

2.3. Simulação Distribuída

Ao considerarmos mudanças de demanda e simuladores que se adaptam e evoluem, um aspecto importante é a possibilidade de distribuição da simulação, viabilizando simulações realizadas com modelos executados em diferentes pontos geográficos, promovendo um melhor uso da capacidade computacional.

Fujimoto (2000) cita alguns benefícios do uso de arquiteturas de simulação distribuídas: (i) redução do tempo de resposta, na qual uma simulação complexa pode ter suas tarefas subdivididas em subtarefas sendo executadas em diferentes processadores independentes; (ii) distribuição geográfica, permitindo que se crie um mundo virtual com vários participantes geograficamente distantes uns dos outros, mas interagindo entre si; (iii) integração de simuladores que são executados em diferentes plataformas operacionais; e (iv) tolerância a falhas, uma vez que a distribuição da simulação entre vários processadores permite que caso um processador sofra uma avaria[5], os demais possam continuar o processamento assumindo o trabalho do processador com problemas, fazendo com que a simulação não seja interrompida.

Outras perspectivas relativamente à distribuição da simulação são (REIS, 2009): (i) aproveitamento da capacidade computacional existente, na qual a distribuição de modelos que podem ser executados em paralelo é uma solução viável e simples; e (ii) escalabilidade, simulações já existentes podem atender a novas exigências por meio da adição de novos modelos e plataformas.

Quanto ao uso de simuladores distribuídos na área espacial, para além das vantagens já apontadas - escalabilidade, paralelismo, etc., podemos citar vantagens na incorporação de hardware na malha da simulação. Desta forma, vários elementos físicos de um laboratório podem estar geograficamente distantes e serem incorporados na simulação como elementos distribuídos, o

[5] Neste trabalho nós usamos a terminologia avaria para o conceito de *failure* em inglês, conforme consta no glossário.

que se mostra bastante interessante uma vez que os equipamentos de teste são bastante específicos, caros, e tendem a estar instalados em laboratórios ou ambientes de integração e teste isolados.

A engenharia concorrente, uma metodologia de trabalho baseada na paralelização de tarefas, é bastante utilizada nas fases de concepção de uma missão espacial para a execução de análises de viabilidade e definição de requisitos de forma rápida e eficiente (DENG et al., 2013). Na área espacial, a engenharia concorrente pode se beneficiar do apoio de simulações distribuídas que utilizem uma infraestrutura comum de distribuição, já que o intercâmbio de informações entre as diferentes disciplinas é uma constante e as análises de custo-benefício (*trade-offs*) devem ser realizadas em conjunto. O trabalho de Zhongshi et al. (2011) analisa o uso de simulação distribuída, projeto colaborativo e técnicas de otimização nas fases de análise e concepção da missão e apresenta uma arquitetura que utiliza o HLA como infraestrutura de distribuição.

A simulação distribuída também pode ser usada como apoio a treinamentos nos quais os atores da operação de satélites estão geograficamente distribuídos. No treinamento para operação do *Automated Transfer Vehicle* (ATV), uma espaçonave europeia projetada para fornecer água, combustível, ar, etc. para a Estação Espacial Internacional (ISS), por exemplo, são utilizados simuladores operacionais interconectados por meio de uma infraestrutura HLA, conforme apresenta Werkman et al. (2012).

Além disso, de maneira geral, um satélite real é constituído por uma série de subsistemas funcionais que podem ser modelados separadamente. Essa abordagem é relativamente natural já que as fronteiras entre os subsistemas são físicas e podem ser logicamente definidas, a divisão em subsistemas pode tornar mais simples a modelagem, e interfaces bem definidas entre os modelos podem ser especificadas. Essa modularidade viabiliza a distribuição da simulação (MIAO et al., 2009). Para o caso de simulação de constelações de satélite, uma arquitetura distribuída também é naturalmente adequada (YIQUN et al., 2010).

Existem arquiteturas e padrões para simulação distribuída cujos aspectos mais importantes estão centrados no reúso de modelos e na interoperabilidade, possibilitando que elementos de software desenvolvidos de forma independente possam trabalhar em conjunto visando um objetivo comum. O *Distributed Interactive Simulation (DIS) e o Test and Training Enabling Architecture (TENA)* são dois exemplos de arquiteturas utilizadas principalmente na área militar na qual o uso de simulação distribuída para modelagem de cenários de guerra ou jogos de guerra é bastante comum.

O padrão HLA é uma evolução do DIS e também se originou e é ainda muito utilizado na área militar. No entanto, esse padrão, por ser genérico no que tange a aspectos de distribuição e de serviços fornecidos pela infraestrutura, tem sido pesquisado e usado em diferentes áreas e é o objeto de avaliação da abordagem de benchmarking proposta neste trabalho.

2.4. *High Level Architecture* (HLA)

O HLA foi inicialmente desenvolvido pelo Departamento de Defesa Americano (DOD), no início dos anos 90, visando aumentar a interoperabilidade e reúso de simuladores em projetos da área militar.

A partir de 2000, como consequência do seu uso em diferentes áreas, o HLA tornou-se um padrão internacional mantido pela IEEE que estabeleceu a norma IEEE P1516-2000 (IEEE, 2001). Em 2010, a IEEE publicou a norma "*HLA Evolved*" que introduziu melhorias na versão anterior, tais como: tolerância a falhas, API para *Web Services*, uso de bibliotecas dinâmicas (dll), mecanismos de redução de taxas de atualização de dados, etc. (MÖLLER et al., 2008; IEEE, 2010a). A Figura 2.3 apresenta a evolução do padrão HLA ao longo dos anos.

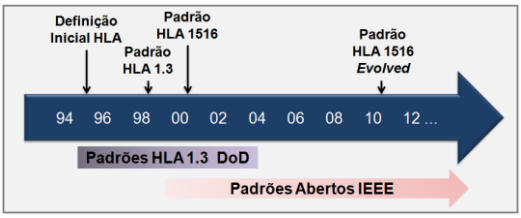


Figura 2.3 - Linha do Tempo - Evolução do Padrão HLA.

Fonte: adaptada de Möller e Olsson (2004).

O HLA é uma arquitetura de software para criação de simuladores a partir de componentes de simulação (modelos), provendo um *framework* por meio do qual um desenvolvedor pode estruturar e descrever as suas aplicações de simulação (MÖLLER et al., 2005).

Na nomenclatura HLA, um modelo de simulação HLA-conforme é chamado de Federado e a simulação, composta por um conjunto de federados, é chamada de Federação. Como a arquitetura permite um alto grau de complexidade, é possível definir um simulador complexo como um conjunto de federações.

2.4.1. Infraestrutura de Execução (RTI HLA)

A infraestrutura de simulação no HLA é chamada de Infraestrutura de Execução (*Runtime Infraestructure* – RTI) e pode ser entendida como um *middleware* de simulação (IEEE, 2010b).

Em termos computacionais, um *middleware* de distribuição é um agente de software que atua como intermediário entre os diferentes processos distribuídos, permitindo que aplicativos sejam executados em um ou vários computadores e interajam através de um canal de comunicação (CHAUDRON et al., 2011). Assim, pode-se classificar uma implementação RTI HLA como um *middleware* para simuladores no qual requisitos comuns de uma infraestrutura de simulação são tratados pela infraestrutura, tais como: intercâmbio de dados entre modelos, sincronização da simulação, salvamento e recuperação de estados e gerenciamento de tempo. A própria RTI, dependendo da implementação, pode utilizar nas camadas de distribuição *middlewares* padrões e genéricos.

De maneira geral, a maior parte das implementações RTI HLA possui os seguintes componentes:

- Dados de Inicialização da RTI (RID): são arquivos específicos, dependentes da implementação, que contêm informações de configuração da RTI.

- Componente Central RTI (CRC ou *RTIExec*): é a aplicação que coordena e gerencia as federações. Este componente permite que federados se associem ou deixem a federação, provê a troca de dados entre federados, executa serviços entre federados, sincroniza a federação, etc.

- Componente Local RTI (LRC ou *RTILib*): biblioteca fornecida com a RTI (lib, dll, so) que é ligada (*linked*) a cada federado para a execução de serviços da infraestrutura e para a execução dos serviços de *callback* - serviços por meio dos quais a RTI se comunica com os federados. Em termos de implementação, a norma HLA prevê duas classes base: a *RTIAmbassador*, implementada pela biblioteca RTI (*RTILib*), que fornece ao federado os serviços disponíveis na API HLA e a classe *FederateAmbassador* que deve ser derivada no código do federado permitindo as chamadas de *callback*. A Figura 2.4 apresenta a estrutura de uma biblioteca RTI (*RTILib*).

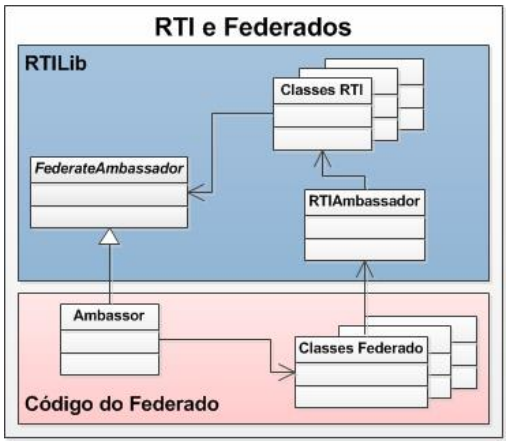


Figura 2.4 - Diagrama de classes - RTI e Federados.

A Figura 2.5 apresenta uma visão lógica da execução de uma federação utilizando uma RTI HLA. Como se pode observar, uma execução HLA é composta pelo processo *RTIExec* (CRC), pelos federados que compõem a federação e que devem incorporar a biblioteca RTI (*RTILib*), e pelo canal de comunicação que provê a forma de interação entre federados e entre os federados e o processo *RTIExec*.

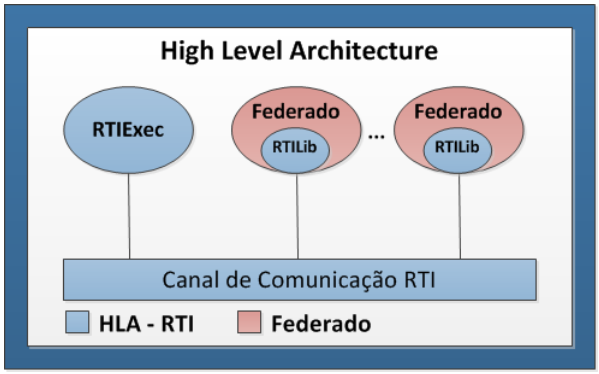


Figura 2.5 - Arquitetura HLA.
Fonte: adaptada de IEEE (2001).

2.4.2. Normas HLA

Em relação à sua normatização, o HLA é estruturado em três elementos principais: Modelo de Objeto (*Object Model Template* - OMT) (IEEE, 2010c), Especificação de Interfaces (IEEE, 2010b) e Regras HLA (IEEE, 2010a).

As subseções seguintes apresentam, de forma resumida, o modelo de objeto OMT e a Especificação de Interfaces (API HLA), visto que essas definições ajudam na compreensão de aspectos da norma HLA que foram utilizados na definição dos elementos do benchmark de resiliência proposto. As regras HLA expressas pela norma IEEE (2010a) e o fluxo de trabalho (*workflow*) de um federado típico são apresentados no Apêndice A.

2.4.2.1. Modelo de Objeto (*Object Model Template* – OMT)

O modelo de objeto OMT é uma especificação padronizada que descreve a sintaxe e a semântica de elementos dos federados e federações - dados intercambiados, interações entre eles, etc. - permitindo que diferentes implementações HLA os reconheçam. A especificação garante reusabilidade e interoperabilidade, possibilitando que um dado federado (simulador ou modelo), por meio de sua descrição, possa ser identificado e reutilizado em diferentes contextos.

O HLA OMT define basicamente três especificações: (i) *modelo de objeto de simulação* (SOM)**,** define os tipos de informação e serviços que um federado irá

prover e requerer da federação (*Classes de Objeto e Classes de Interação*[6]); (ii) *modelo de objeto de federação* (FOM), define os dados intercambiados entre federados (publicados e subscritos) e as interações entre federados de uma federação, reunindo as informações SOM do conjunto de federados participantes; e (iii) *modelo de Objeto de Gerenciamento (MOM)*, especifica o conjunto de dados e serviços que serão providos pela RTI HLA e utilizados pelos federados para obter informações sobre a federação da qual fazem parte.

2.4.2.2. Especificação de Interface

A *Especificação de Interface* define as interfaces funcionais entre os federados e a Infraestrutura de Execução (RTI), definindo tanto os serviços que podem ser utilizados pelos federados em uma simulação HLA, quanto os serviços de *callback* que os federados devem prover para que a RTI possa se comunicar com os mesmos.

Os serviços especificados na *Especificação de Interface* (API HLA) estão divididos em seis áreas principais de gerenciamento: *Gerenciamento de Federação*, *Gerenciamento de Declaração*, *Gerenciamento de Objeto*, *Gerenciamento de Propriedade*, *Gerenciamento de Distribuição de Dados* e *Gerenciamento de Tempo*. Além disso, há um grupo de serviços genéricos providos pela RTI. Em termos lógicos, os serviços definidos pela API HLA e implementados pela RTI podem ser categorizados em: serviços de coordenação da federação, serviços para troca de informações e interação entre federados, e serviços de gestão da linha de tempo da federação.

a) Serviços de Coordenação da Federação

Serviços relacionados ao gerenciamento da federação que são divididos em: (i) *serviços de coordenação da execução da federação,* permitem que um dado federado crie ou destrua uma federação, se ligue a uma federação ou

[6] No HLA OMT, que utiliza lógica análoga à da Orientação a Objetos (OO), as informações trocadas entre federados e os serviços providos pelos federados são descritos por meio de *Classes de Objetos* e seus *Atributos* e *Classes de Interação* e seus *Parâmetros*, respectivamente.

abandone uma federação; (ii) *serviços de sincronização*, permitem que a federação ou grupos de federados se sincronizem em determinados pontos previamente cadastrados (a RTI é responsável por estabelecer a sincronização da federação); (iii) *serviços de salvamento e recuperação*, permitem que seja salvo o estado geral da federação de forma que o mesmo possa ser restaurado a partir do ponto de salvamento; e (iv) *serviços de transferência de propriedade*, permitem que um federado passe a outro, em tempo de execução, a responsabilidade por atualizar o conjunto de seus atributos, possibilitando a continuidade da federação mesmo depois de sua saída.

Os serviços que tratam da transferência de responsabilidade entre federados estão contemplados na área de *Gerenciamento de Propriedade*, todos os demais serviços de coordenação fazem parte da área de *Gerenciamento de Federação*.

b) Serviços de Informação e Interação entre Federados

Os serviços de informação e interação permitem que federados dentro de uma federação troquem dados entre si e executem serviços de outros federados de acordo com as especificações do arquivo FOM, usando um modelo de publicação/subscrição. A RTI tem o papel de intermediária na interação entre federados e no intercâmbio de dados, chamando serviços e mantendo a interconexão entre eles.

As principais áreas de gerenciamento envolvidas no serviço de informação são as áreas de *Gerenciamento de Declaração*[7] e *Gerenciamento de Objetos*[8].

Relativamente à troca de informações, existe um conjunto de serviços avançados da área de *Gerenciamento de Distribuição de Dados* que permite que filtros de valores (dimensões) sejam usados para indicar o interesse dos

[7] Os serviços de *Gerenciamento de Declaração* funcionam como a declaração de variáveis em linguagens de programação no que tange a publicação e subscrição de *Classes de Objetos* e *Classes de Interação*.
[8] Os serviços do *Gerenciamento de Objetos* permitem instanciar *objetos* declarados anteriormente como publicáveis (*Classes de Objetos*).

federados em determinados atributos quando os mesmos estão dentro de uma faixa de valores específica. Isso permite que se diminua a carga no canal de comunicação, embora exija maior capacidade de processamento da RTI.

c) Serviços de Linha de Tempo

Por meio de serviços de tempo, a RTI coordena o avanço do tempo dos federados em uma linha de tempo comum à federação, promovendo a sincronização temporal, permitindo que federados enviem mensagens ordenadas uns para os outros tendo a garantia de que essas mensagens não chegarão ao destino em um tempo passado, o que possibilita a repetibilidade da simulação.

Importa salientar que cada federado pode especificar o formato e o significado interno do seu tempo em uma seção do arquivo SOM OMT.

2.4.3. Características da HLA e Aspectos de Resiliência

Embora uma RTI HLA-conforme deva prover os serviços previstos na *Especificação de Interface*, bem como respeitar as regras HLA, o padrão não define detalhes relativamente a algoritmos utilizados na implementação dos serviços, a protocolos de comunicação e a arquitetura da RTI. Essa liberdade prevista pelo padrão permite que surjam diferentes soluções, algumas centradas em desempenho, outras em confiabilidade, etc. No entanto, até o momento, essa mesma liberdade é ainda um obstáculo para a completa interoperabilidade entre RTIs de diferentes fabricantes (WATROUS et al., 2006; ROSS, 2012).

Uma vez que existem variações nas implementações do padrão HLA, o trabalho de Ross (2012) analisou as diferenças entre as principais RTIs do mercado considerando os seguintes critérios: arquitetura (*modus operandi*), meio de comunicação, canais de comunicação e formato de mensagem. Relativamente à arquitetura, uma implementação pode ser: (i) centralizada, quando um processo central gerencia a distribuição (CRC), (ii) descentralizada, quando a própria biblioteca RTI (*RTILib* ou LRC) ligada a cada federado é

responsável pela comunicação entre eles e pela execução de todos os serviços a serem disponibilizados, e (iii) hierárquica, quando são executados vários processos de comunicação (*CRCs*). No que diz respeito ao canal de comunicação, os protocolos mais utilizados são *User Datagram Protocol* (UDP), *Transmission Control Protocol* (TCP) e UDP *multicast*. No entanto, algumas implementações utilizam na camada de comunicação outros *middlewares* ou bibliotecas, tais como *Jgroups*, *Common Object Request Broker Architecture* (CORBA), *Data Distribution Service for Real-time Systems* (DDS), entre outros.

Relativamente ao desempenho das RTIs, o artigo de Watrous et al. (2006) apresenta uma análise detalhada do conjunto de fatores que pode ter impacto no desempenho de uma federação. Esse artigo classifica os principais aspectos em: (i) requisitos característicos da especificação HLA que podem direcionar escolhas de projeto; (ii) arquitetura da RTI, modo de operação (síncrono/assíncrono, *single-thread/multithread*), canal de comunicação e algoritmos usados na implementação de serviços HLA; (iii) requisitos da federação quanto ao número de objetos, número de federados, uso dos serviços, etc.; e (iv) contingências de recursos físicos, disponibilidade de rede, equipamentos, etc.

A partir das análises apresentadas nos artigos citados acima e do nosso estudo da norma HLA no que se refere à *Especificação de Interface*, foram identificados elementos e características da especificação e de alternativas de implementação que podem ter impacto em atributos de resiliência da RTI HLA, especialmente no que se refere ao desempenho. A Tabela 2.2 apresenta os principais elementos subdivididos em classes e descreve o potencial impacto dos mesmos.

Tabela 2.2 - Elementos HLA - Impactos na dependabilidade.

Item	Descrição
Transporte	
Tipo	A configuração do tipo de transporte para os dados intercambiados entre federados pode ser: c*onfiável* (*reliable*), no qual a entrega da mensagem é garantida; ou *não-confiável* (*best effort*), caso contrário. O tipo de transporte normalmente guia a escolha do protocolo de rede utilizado pela RTI. Frequentemente, utiliza-se o protocolo TCP para envio de dados *confiáveis* e protocolo UDP para dados *não-confiáveis*.
Passelization	Uma *Classe de Objeto* HLA pode ser declarada tendo parte dos atributos configurados como *confiáveis* e outra parte como *não-confiáveis*. Sempre que isso ocorrer, a RTI quebrará a mensagem em duas partes e a esse processo dá-se o nome de *Passelization*. O processo de *passelization* pode gerar sobrecarga no canal de comunicação e demandar mais processamento da RTI.
Serviços	
Tempo	Os algoritmos usados para a execução da gestão do tempo podem ter impacto no desempenho da RTI.
Distribuição de Dados	O uso de serviços do Gerenciamento de Distribuição de Dados nos quais os federados subscrevem não apenas a atributos, mas também selecionam faixas de valores de interesse, diminui o tráfego da rede, mas exige maior capacidade de processamento da RTI.
Transferência de Propriedade	Um federado pode transferir a propriedade dos atributos que publica e atualiza. Quando isso ocorre é estabelecido um protocolo entre federado-RTI-federação visando à transferência da responsabilidade pela atualização dos dados para outro federado. Essa transferência de propriedade pode ter impacto no desempenho em federações nas quais a entrada e saída de federados ocorre com frequência.
Mensagens RTI<->Federados	
Chaves de consulta (*advisory key*)	A RTI pode ser configurada para informar à federação sobre a entrada de um federado que publica objetos/serviços aos quais outros federados subscreveram. Há um conjunto de mensagens de aviso que podem ser enviadas da RTI para os federados. Esse envio de avisos pode ter impacto no processamento e na carga do canal de comunicação (esse serviço pode ser desabilitado).
Serviços MOM	Permite que os federados solicitem à RTI informações sobre a federação. Esses serviços têm impacto análogo aos impactos das chaves de consultas descritas acima.
Arquitetura	
Modelo de *Callback*	Há três estratégias básicas de *callback* (exemplo para intercâmbio de dados): (i) *callback* síncrono executado em *thread* única, neste caso, o processamento só será realizado e os dados só serão recebidos pela *RTILib* e entregues ao federado quando o mesmo fizer uma chamada a um serviço específico da *RTILib* que possibilita a efetivação do *callback*; (ii) *callback* síncrono executado com I/O assíncrono, na qual a *RTILib* é executada em *thread* separada. Neste caso, a *RTILIB* realiza o processamento necessário e recebe os dados assim que eles chegam, no entanto, os dados só são entregues ao federado mediante chamada ao serviço da *RTILib* que possibilita a efetivação do *callback*; (iii) *callback* assíncrono, os dados são entregues ao federado assim que recebidos da RTI, não há necessidade de chamadas do federado a serviços da *RTILib*. Isso implica em federados *multithread*. A norma prevê a configuração do tipo de callback: síncrono (*HLA_Evoked*) ou assíncrono (*HLA_Imadiate*). Vale salientar que as estratégias usadas para intercâmbio de dados entre federados são também usadas para todas as demais chamadas de *callback*.

Como grande parte das características apresentadas na Tabela 2.2 são importantes do ponto de vista de dependabilidade e resiliência, esses aspectos foram considerados na especificação da abordagem de benchmarking proposta, conforme será apresentado no Capítulo 5.

2.5. Considerações Finais

Neste capítulo foi apresentado o conjunto de simuladores definidos por Eickhoff (2009) pelo memorando técnico ECSS (2010), suas principais características e como os mesmos são utilizados ao longo do ciclo de vida de uma missão espacial. As duas propostas definem modelos evolutivos de simuladores nos quais podem ser reutilizadas a infraestrutura de simulação e os modelos padronizados, principalmente, modelos de ambiente e de dinâmica do satélite. Nomeadamente, os simuladores definidos pelo memorando técnico ECSS (2010) são utilizados neste trabalho para a especificação de elementos da abordagem de benchmarking de resiliência proposta, bem como para a identificação de requisitos de dependabilidade e resiliência da infraestrutura de simulação.

A metodologia de especificação de benchmarking detalhada no Capítulo 5 tem como alvo de avaliação infraestruturas para simuladores baseadas no padrão HLA, por esse motivo buscou-se neste capítulo apresentar uma visão geral do padrão, já que muitos elementos do benchmark proposto foram definidos com base nas características da arquitetura das RTIs e da API HLA, em particular aquelas que podem ser modificadas e configuradas e que podem ter impacto nos atributos de resiliência e dependabilidade.

3 DEPENDABILIDADE, RESILIÊNCIA E BENCHMARKING

Este capítulo apresenta os conceitos que embasam este trabalho no que tange a dependabilidade, resiliência e benchmarking, incluindo aspectos relacionados à injeção de falhas, uma técnica essencial em processos de benchmarking de dependabilidade e resiliência baseados em métodos experimentais (BARBOSA et al., 2012).

Os principais trabalhos relacionados ao objeto desta pesquisa, i.e., benchmarks para simuladores baseados na infraestrutura HLA e benchmarks de resiliência em outros domínios, também são apresentados.

3.1. Dependabilidade e Resiliência

Há na literatura duas definições amplamente aceitas para dependabilidade: "*Dependabilidade é a capacidade do sistema em fornecer serviços no qual se possa justificadamente confiar*" e "*Dependabilidade é a capacidade do sistema em prevenir falhas que são inaceitavelmente severas ou frequentes*" (AVIZIENIS et al., 2004).

Dependabilidade não é uma propriedade isolada de um sistema, mas sim um conjunto de atributos que deve ser atendido em níveis adequados. No entanto, não existe na literatura uma definição única e consolidada sobre o conjunto de atributos que define a dependabilidade de um sistema e esse conjunto de atributos geralmente varia de um autor para outro e de uma comunidade para outra. Além disso, tanto os atributos, quanto os níveis aceitáveis de atendimento aos mesmos varia de acordo com o domínio do sistema a ser desenvolvido ou avaliado (RUS et al., 2003; WEINSTOCK et al., 2004).

Segundo Avizienis et al. (2004), a dependabilidade de um sistema é caracterizada pelos seguintes atributos: *disponibilidade*, prontidão para a oferta de serviços corretos; *confiabilidade*, continuidade de oferta de serviços corretos; *segurança (safety)*, inocuidade do sistema (não prejudica o meio/sistema envolvente); *integridade*, ausência de alterações impróprias no

sistema, nos serviços ou nos dados; *confidencialidade*, garantia do resguardo das informações e *manutenabilidade*, facilidade de manutenção.

Os atributos disponibilidade, confiabilidade, segurança (*security*) e segurança (*safety)* são comuns à maioria dos autores (MELHART; WHITE, 2000; AVIZIENIS et al., 2004; BASILI et al., 2004; BOEHM et al., 2004; ROMANI et al., 2010). Alguns autores incluem, também, outros atributos, tais como robustez (FIRESMITH, 2003; RUS et al., 2003; BOEHM et al., 2004; ROMANI et al., 2010), sobrevivência (*survivability*) (MELHART; WHITE, 2000; RUS et al., 2003; BASILI et al., 2004; BOEHM et al., 2004; ROMANI et al., 2010), desempenho (BOEHM et al., 2004), correção (BOEHM et al., 2004), completeza (ROMANI et al., 2010), interoperabilidade (BOEHM et al., 2004), etc.

Mais recentemente, a característica evolutiva e sujeita a mudanças de grande parte dos sistemas computacionais tem levado, também, a preocupações com a **resiliência** desses sistemas.

Resiliência, da raiz latina *resiliere* (saltar para trás), significa tendência ou capacidade de se recuperar ou voltar atrás. Este conceito está associado, de forma figurativa, à capacidade de tolerar, acomodar ou se recuperar de adversidades e mudanças e tem sido aplicado em diferentes disciplinas, tais como psicologia, ecologia, sociologia, economia, organizações e, também, no contexto de sistemas computacionais. Entretanto, embora haja concordância no entendimento do que é resiliência, não há uma definição que seja transversal às diferentes áreas.

O conceito de resiliência na física, por exemplo, está associado à propriedade que um corpo tem de ***recuperar*** a sua forma original após sofrer choque ou deformação (PRIBERAM, 2013*)*.

Em psicologia, resiliência pode ser definida como "*o processo ou capacidade ou resultado de uma adaptação bem sucedida em função de circunstâncias desafiadoras ou ameaçadoras*" e tem sido usado para descrever três

fenômenos distintos: (1) ***bons resultados***, apesar de alto risco, (2) competência ***sustentada*** sob ameaça; e (3) ***recuperação*** de traumas. (MASTEN et al., 1990).

Na área dos sistemas ecológicos, segundo Holling (1973), resiliência determina a persistência das relações dentro de um sistema e é uma medida da capacidade que esses sistemas têm para ***absorver*** as perturbações e ainda ***persistirem***. Na sua visão, a resiliência é medida pela magnitude de perturbação que um sistema pode tolerar e a estabilidade, por outro lado, é a capacidade de um sistema em retornar a um estado de equilíbrio após uma perturbação temporária. Já para Pimm (1984), a medida adequada de resiliência é a capacidade do sistema para ***resistir*** a uma perturbação e a taxa na qual ele ***retorna*** ao equilíbrio após a sua ocorrência.

Resiliência é um termo não apenas multidisciplinar, mas também multifacetado. Segundo Madni e Jackson (2009), o termo resiliência está relacionado à capacidade que um sistema tem de ***evitar***, ***resistir a***, se ***adaptar*** e se ***recuperar de*** rupturas e perturbações. A Figura 3.1 ilustra os aspectos da Resiliência.

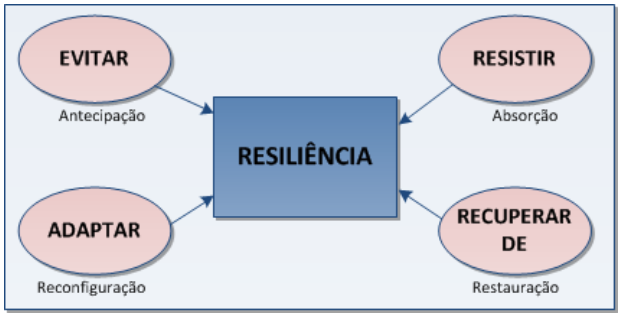


Figura 3.1 - As Faces da Resiliência.

Fonte: Madni (2009)

Nos sistemas computacionais, segundo Laprie (2008), o adjetivo resiliência vem sendo utilizado há décadas na área de dependabilidade como sinônimo de tolerância a falhas, não levando em consideração os aspectos evolutivos desses sistemas. No entanto, um conceito mais amplo de resiliência deve levar em consideração a capacidade dos sistemas em acomodar e tolerar

mudanças, mantendo a dependabilidade. Partindo dessa visão, a resiliência pode ser vista como "*a persistência de dependabilidade em face de mudanças*".

Bondavalli et al. (2009) definem que resiliência engloba os atributos de qualidade para "*trabalhar bem em um mundo que se transforma e que está sujeito a falhas, erros e ataques*" e Almeida e Vieira (2011) definem resiliência como "a c*apacidade de manter serviços que podem ser justificadamente confiáveis apesar de alterações nos seus contextos internos e externos".*

De maneira geral, de acordo com as definições apresentadas, as capacidades chaves relacionadas à resiliência são: *acomodação* (adaptar, absorver, resistir, lidar, sobreviver, tolerar)*, recuperação* (retornar, reestabelecer) e manutenção dos serviços (sustentar, persistir, manter). Será utilizada neste trabalho a definição adaptada de Almeida et al. (2013) na qual *"resiliência é uma propriedade emergente de um sistema que representa a sua capacidade de acomodar e se recuperar de mudanças em sentido amplo, mantendo a qualidade do serviço prestado".* Considera-se assim que um sistema é resiliente quando acomoda essas mudanças de forma eficaz e eficiente, sem perda da qualidade.

No que tange ao desenvolvimento de sistemas computacionais, é possível destacar tecnologias que têm por objetivo a resiliência dos sistemas (LAPRIE, 2005, 2008; RESIST, 2007): *evolubilidade*, promove a evolução dos sistemas de forma a acomodar mudanças nos requisitos de usuários, no contexto operacional (mudanças de curto e longo prazo no ambiente, na tecnologia e nos cenários de ameaças), nos recursos disponíveis, incluindo a capacidade de evoluir enquanto executa (*adaptabilidade*); *avaliabilidade*, promove a avaliação, inclusive em tempo de execução, da habilidade que um sistema tem de funcionar adequadamente e de manter a qualidade dos serviços que provê; *usabilidade*, promove o atendimento de aspectos relacionados ao uso - fácil aprendizado, fácil memorização, eficiente e livre de erros - com o objetivo de atingir maior resiliência; e *diversidade***,** promove o aproveitamento de redundâncias e diversidade antevendo vulnerabilidades.

Neste trabalho nós consideramos a resiliência sob o ponto de vista de *tecnologias de avaliabilidade*, em especial no campo de abordagens de benchmarking.

3.2. Benchmarking

Benchmarks são procedimentos ou ferramentas padronizadas que permitem avaliar e comparar diferentes sistemas de um mesmo domínio de acordo com características específicas, por exemplo, desempenho, dependabilidade, segurança, etc. (BONDAVALLI et al., 2009; VIEIRA et al., 2008, 2012). Essas ferramentas, importantes na tomada de decisões relativas à escolha de sistemas dentro de um mesmo domínio ou na avaliação da adequação de um sistema a um dado contexto, também desempenham um importante papel na evolução geral de sistemas nos domínios considerados.

As subseções seguintes descrevem, em linhas gerais, diferentes aspectos relacionados com benchmarking de sistemas computacionais, com ênfase em abordagens de benchmarking de resiliência.

3.2.1. Benchmarking de Desempenho e de Dependabilidade

As primeiras iniciativas de benchmarking surgiram no início da década de 70 e tinham como principal objetivo a avaliação de aspectos de desempenho, tendo como alvo desde sistemas específicos, por exemplo, um hardware ou componente, até sistemas complexos como sistemas operacionais, sistemas gerenciadores de banco de dados, etc. (BONDAVALLI et al., 2009, VIEIRA et al., 2012).

Com o objetivo de comparar o desempenho de sistemas em um dado domínio, os benchmarks de desempenho baseiam-se nos seguintes componentes principais: *carga de trabalho*, que é uma representação da carga real à qual o sistema avaliado estaria submetido, o que inclui aspectos estáticos, dinâmicos e operações críticas; o *conjunto de métricas* que caracteriza o desempenho do mesmo; e *regras de aplicação*.

No contexto de benchmarks de desempenho, organizações de padronização como a *Standard Performance Evaluation Corporation* (SPEC) e a *Transaction Processing Performance Council* (TPC) exercem um papel importante na definição de benchmarks em domínios específicos.

A SPEC oferece um conjunto de benchmarks relativos ao domínio de processadores (atualmente SPEC CPU2006) e também conta com publicações em outros domínios, tais como computação de alto desempenho (SPEC MPI2007, SPEC MPI2012), java (SPECjbb2013, SPECjEnterprise2010, etc.), *Network File System* (NFS) (SPECsfs2008), entre outras. A SPEC fornece os seus benchmarks em forma de ferramentas comercializadas e disponibiliza em sua página na internet[9] os resultados submetidos por fabricantes.

A TPC tem a missão de definir benchmarks que avaliam o desempenho do Processamento de Transações (TP) e de Banco de Dados (BD) e de divulgar para a indústria resultados confiáveis e verificáveis do desempenho desses sistemas. Atualmente, a TPC define benchmarks com diferentes propósitos, tais como: (i) TPC-C (TPC, 2010a) e TPC-E (TPC, 2010b), benchmarks para sistemas *On-Line Transaction Processing* (OLTP) que representam sistemas comerciais e diferem entre si em relação à carga de trabalho; e (ii) TPC-DS (TPC, 2012) e TPC-H (TPC, 2013), benchmarks para sistemas de apoio à decisão. A TPC fornece os seus benchmarks na forma de especificações (documentos) que definem as cargas de trabalho, as regras de aplicação do benchmark e o formato dos relatórios de resultados (*disclosure reports*); e disponibiliza em sua página na internet[10] os resultados sumarizados de diversos fabricantes.

Na década passada, a crescente percepção da necessidade de avaliação da dependabilidade dos sistemas computacionais fez com se intensificasse a pesquisa na área de benchmarks de dependabilidade. Esses benchmarks têm

[9] SPEC http://www.spec.org/.
[10] TPC http://www.tpc.org/default.asp.

como objetivo principal avaliar, de forma estruturada e padronizada, sistemas computacionais relativamente a atributos de dependabilidade quando na presença de falhas e introduzem dois novos elementos em relação aos benchmarks de desempenho: *carga de falhas*, que é uma representação das possíveis falhas (projeto, ambiente, etc.) às quais o sistema poderia eventualmente estar exposto e *medidas de dependabilidade e desempenho-dependabilidade* (RESIST, 2007; VIEIRA et al., 2008; BONDAVALLI et al., 2009; COSTA et al., 2009).

Em benchmarks de dependabilidade, um dos elementos mais complexos é a carga de falhas para os diferentes domínios já que, assim como acontece com a carga de trabalho, a carga de falhas está intrinsecamente associada ao domínio do problema e deve representar tanto quanto possível as falhas que poderiam afetar o sistema a ser avaliado. No que tange a falhas gerais de software, há na literatura um conjunto de trabalhos que avaliam a representatividade de cargas de falhas. Por exemplo, a partir da Classificação Ortogonal de Defeitos (ODC) proposta por Chillarege et al. (1992), os trabalhos de Durães e Madeira (2003, 2006) apresentaram resultados que indicavam uma tendência observada em mais de 600 falhas, mostrando que apenas um conjunto bem definido de tipos de falhas (13 tipos) é o responsável por um grande número de avarias nos sistemas e que, assim, a sua proporcionalidade é candidata a compor uma carga de falhas para avaliação de sistemas de software. Já outros trabalhos mostraram especificamente a representatividade das falhas de software na composição de cargas de falhas (DURÃES; MADEIRA, 2004, 2006).

Relativamente a benchmarks de dependabilidade, podem-se encontrar na literatura diversos trabalhos aplicados a diferentes domínios: Sistemas de Processamento de Transações em Tempo Real (OLTP) (VIEIRA; MADEIRA, 2003; VIEIRA et al., 2008), *Web Servers* (DURÃES et al., 2008), Sistemas de Controle Automotivos (RUIZ et al., 2008), Sistema Embarcados da Área Espacial (COSTA et al., 2008), entre outros. Alguns trabalhos visaram aspectos

mais específicos, tais como recuperação automática de sistemas (MAURO et al., 2004) e avaliação de detectores de anomalias (TAN; MAXION, 2008).

Alguns tipos de benchmark podem ser vistos como categorias especiais de benchmarks de dependabilidade, tais como: *benchmarks de robustez,* nos quais a carga de falhas é composta por falhas de interface e métricas específicas de robustez são avaliadas (KOOPMAN et al., 2008; KANOUN et al., 2008; MICSKEI et al., 2007); e *benchmarks de segurança* (*security*), nos quais a avaliação leva em consideração, por exemplo, a injeção de vulnerabilidades e as métricas consideradas são capazes de caracterizar o grau em que os objetivos de segurança foram atendidos por um determinado sistema (VIEIRA et al., 2012).

O *framework* para benchmarking de dependabilidade apresentado em alguns dos trabalhos citados acima (VIEIRA, 2005; DURÃES et al., 2008) e pelo projeto DBENCH (2004) foram usados como base para a definição de vários elementos do benchmarking de resiliência proposto nesta tese.

3.2.2. Benchmarking de Resiliência

Embora um benchmark de dependabilidade estenda o conceito tradicional de benchmarking de desempenho para incluir medidas e caracterizações relativas a atributos de dependabilidade e incorpore a carga de falhas na avaliação, ele ainda não leva em consideração a característica evolutiva dos sistemas, a dinâmica dos ambientes nos quais estes sistemas estão inseridos e as mudanças às quais o sistema poderia estar potencialmente exposto.

Mudanças e perturbações podem induzir o sistema a um estado de degradação no que tange à sua operação, ainda que não leve necessariamente a uma falha observável (MEYER, 2009). Assim, faz-se necessário a avaliação de variações nas propriedades de interesse, sejam elas de desempenho ou dependabilidade, quando o sistema está sob condições de alteração em seu contexto ou sob novos e diferentes contextos (BONDAVALLI et al., 2009; VIEIRA et al., 2012). Na prática, benchmarks de resiliência levam em

consideração esses aspectos evolutivos e proveem formas genéricas de caracterizar e comparar sistemas computacionais quando sujeitos a mudanças e perturbações, permitindo a realização de medidas de resiliência (ALMEIDA; VIEIRA, 2011).

Quando comparado a benchmarks de dependabilidade, um benchmark de resiliência incorpora mais três elementos: *condições operacionais*, que em conjunto com a carga de trabalho compõe um cenário de referência ou Cenário Base; *carga de mudanças*, compostas pelas mudanças esperadas e *métricas de resiliência*. A Figura 3.2 mostra os elementos do benchmark de resiliência em comparação a benchmarks de desempenho e de dependabilidade.

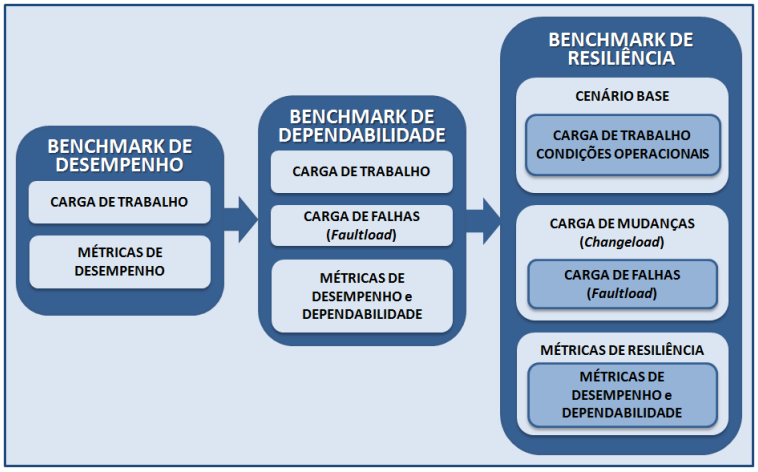


Figura 3.2 - Benchmark de Resiliência - Componentes Principais.
Fonte: Adaptada de Almeida e Vieira (2011).

Em um benchmark de resiliência, a carga de mudanças, que incorpora e estende o conceito da carga de falhas em benchmarks de dependabilidade, é o elemento mais complexo e deve caracterizar, tanto quanto possível, as mudanças às quais o sistema estará sujeito, sejam elas de caráter funcional (p. ex. novas funcionalidades e requisitos), ambiental (p. ex. novos perfis de uso do sistema, falhas, ataques), ou tecnológico (p. ex. novas configurações de hardware e software).

Já as métricas de resiliência devem caracterizar um sistema relativamente à variabilidade nas medidas das propriedades de interesse em função da

aplicação de mudanças, perturbações ou em condições de estresse (BONDAVALI et al., 2009).

As mudanças e perturbações são gerais e podem afetar tanto características da carga de trabalho, quanto representar um novo contexto tecnológico ou de uso. Considerando esses aspectos, um benchmark de resiliência deve definir uma referência de execução por meio da especificação da carga de trabalho do sistema e, também, das condições gerais de sua operação. Neste trabalho, nós utilizaremos o conceito de *Cenário Base* no qual uma carga de trabalho está associada a condições operacionais de referência. Assim, avaliações de um sistema são realizadas submetendo o *Cenário Base* a um conjunto de potenciais mudanças (ALMEIDA; VIEIRA, 2012a).

No que tange às cargas aplicadas e métricas, um benchmark de resiliência pode ser visto como um superconjunto de benchmarks de desempenho e dependabilidade, incluindo benchmarks de segurança (*security*) e benchmarks de robustez. A Figura 3.3 ilustra esta perspectiva.

Figura 3.3 - Perspectivas - Benchmark de Resiliência.

3.2.3. Propriedades de Benchmarks

O que diferencia um benchmark das demais formas de comparação e avaliação de um sistema é que ele deve ter credibilidade junto à indústria ou à comunidade de usuários. Assim, para que um benchmark seja útil ele deve apresentar um conjunto de propriedades contra as quais será avaliado e validado. No domínio de benchmarks de desempenho, Gray (1992) define

quatro propriedades que um benchmark deve atender: *relevância*, o resultado do benchmark deve medir o desempenho de uma operação típica dentro do domínio do problema, refletindo características importantes do mesmo; *portabilidade*, deve ser de fácil implementação em diferentes sistemas e arquiteturas; *escalabilidade*, deve poder ser executado em pequenos e grandes sistemas; e *simplicidade*, deve ser compreensível para ter credibilidade. Huppler (2009) acrescenta que um benchmark deve ser: *repetível*, o resultado do benchmark deve poder ser reproduzido quando reaplicado em condições semelhantes; *justo e portável*, todos os sistemas medidos devem ser considerados de forma igualitária, possibilitando a comparação de diferentes sistemas no mesmo domínio; *verificável*, tem de haver confiança de que os resultados são reais, verdadeiros e auditáveis; e *econômico*, o custo do benchmark deve ser acessível.

Vários trabalhos na área dependabilidade definiram que um benchmark deve, ainda, ser *representativo*, representando tanto quanto possível cenários reais em relação ao comportamento a ser medido e *não-intrusivo*, mantendo inalterado o sistema que avalia (DBENCH, 2004; VIEIRA, 2005).

Neste trabalho nós consideramos que um benchmark de resiliência deve ser: *relevante, representativo, repetível, escalável, portável, não-intrusivo* e *simples*, adotando, assim, a maioria das propriedades apresentadas nos trabalhos de dependabilidade, por considerar que essas propriedades são mais abrangentes e mais afins às características de benchmarks de resiliência. Por exemplo, a propriedade *não-intrusividade* é uma propriedade que deve ser validada em benchmarks de resiliência uma vez que a aplicação de mudanças e falhas pode modificar o alvo do benchmark, invalidando a avaliação de um produto.

3.3. Injeção de Falhas

Tolerância e recuperação são conceitos chave para a resiliência dos sistemas e todas as técnicas que auxiliem na avaliação da capacidade dos mesmos relativamente a esses mecanismos são importantes nesse domínio.

As técnicas de injeção de falhas de hardware e software podem ser utilizadas com os seguintes propósitos (BONDAVALLI et al., 2009, BARBOSA et al., 2012): (i) avaliar a eficiência dos mecanismos de tolerância a falhas; (ii) estudar a propagação e latência de erros de forma a aprimorar mecanismos de tratamento de falhas; (iii) testar o quão correto são os mecanismos de tratamento de falhas ou os protocolos de sistemas distribuídos, (iv) medir o tempo que um sistema leva para detectar ou se recuperar de falhas; (v) verificar as assunções relativas aos modos de falhas de componentes ou subsistemas; e (vi) estudar o impacto das falhas e dos mecanismos de tolerância a falhas na qualidade de serviço (QoS) de um sistema.

Sob o ponto de vista de um benchmarking de resiliência, os itens (i), (iii), (iv) e (vi) são de suma importância, fazendo com que essas técnicas e parte de seus conceitos possam ser diretamente utilizados, ajudando a compor e implementar a carga de mudanças.

As subsecções seguintes apresentam, em linhas gerais, técnicas de injeção de falhas que cobrem aspectos de interesse deste trabalho: injeção de falhas de software, injeção de erros em interfaces e injeção de falhas em protocolos.

Importa salientar que, como na definição da carga de mudanças apresentada nesta tese não foram consideradas falhas de hardware de baixo nível e ataques, técnicas de injeção de falhas de hardware e injeção de vulnerabilidades não foram abordadas nesta seção. Descrições abrangentes sobre técnicas de injeção de falhas podem ser encontradas em Barbosa et al. (2012) e Bondavalli et al. (2009).

3.3.1. Injeção de Falhas de Software

As técnicas de injeção de falhas de software são baseadas na modificação dos sistemas via manipulação do código fonte, código objeto ou executáveis. Uma dessas técnicas, bastante aplicada na área de testes de software, é conhecida como mutantes e foi estendida para a área de injeção de falhas para avaliar os mecanismos de tolerância por meio da emulação de falhas latentes que

poderiam não ter sido previamente detectadas. As primeiras ferramentas que utilizaram este tipo de técnica foram FINE (KAO et al., 1993) e DEFINE (KAO; YVER, 1994).

Ferramentas de injeção de falhas como Jaca e G-SWFIT, por exemplo, são independentes do código fonte. Jaca é usada para injeção de falhas e avaliação de sistemas Orientados a Objetos (OO) desenvolvidos em Java, e utiliza recursos de programação de alto nível para corromper valores de atributos, parâmetros de métodos ou valores de retorno (MARTINS et al. ,2002). A ferramenta *Generic Software Fault Injection Technique* (G-SWFIT) emula falhas de software usando técnicas de mutantes, injetando as falhas no código objeto por meio da alteração desse código em pontos que corresponderiam a um erro em uma linguagem de alto nível (DURÃES; MADEIRA, 2002).

3.3.2. Injeção de Erros de Interface

Testes de robustez normalmente se concentram nas falhas ativadas por meio das interfaces do sistema, apresentando uma carga de falhas que as exercita por meio da aplicação de entradas excepcionais ou condições de estresse. Essa categoria de testes pode ser usada para avaliar ou comparar produtos como no caso de benchmarks de robustez.

Há na literatura um conjunto de técnicas usadas para a geração da carga de falhas em benchmarks de robustez. Podem-se utilizar, por exemplo, injetores de falhas físicas com o objetivo de observar como um componente reage a erros de interação originados ou iniciados por falhas de ambiente injetadas (camadas inferiores ou interfaces). Outras técnicas utilizam valores gerados randomicamente como entradas para o sistema, como a ferramenta Fuzz (KOSKI et al., 1995).

Testes de robustez também podem ser realizados por meio da definição de valores válidos e inválidos para cada tipo de dado usado por uma determinada interface. Neste caso, os testes são realizados por meio da injeção desses

valores válidos e inválidos nos parâmetros de cada serviço da interface. Koopman e DeVale (1999), por meio da ferramenta Ballista, utilizaram essa abordagem em um teste feito para comparar 15 sistemas operacionais POSIX[11]. A ferramenta Ballista serviu como base para vários trabalhos em diferentes domínios (FERNSLER; KOOPMAN, 1999; SHELTON et al., 2000; PAN et al., 2001) e o artigo de Koopman et al. (2008) apresenta um resumo da experiência no uso da mesma.

Diferentes trabalhos apresentaram testes de robustez e benchmarks de robustez[12] por meio do uso das técnicas citadas, por exemplo, na área de sistemas operacionais (SHELTON et al., 2000), na área de *middlewares* CORBA e *Java Message Service* (JMS) (PAN et al., 2001; LARANJEIRO et al., 2008), Serviços Web (TSAI et al., 2005; VIEIRA et al., 2007; MARTIN et al., 2007), controladores em sistemas autoadaptativos (CAMARA et al., 2013b), etc.

3.3.3. Injeção de Falhas em Protocolos

Injeção de falhas em sistemas distribuídos visa revelar falhas de projeto e implementação em mecanismos de comunicação e *middlewares*.

Normalmente esses mecanismos de comunicação são testados por meio de injeção de falhas em mensagens trocadas através dos protocolos, técnica conhecida como injeção de falhas baseadas em mensagens. As falhas injetadas podem se basear na alteração da sintaxe das mensagens por meio da modificação do cabeçalho ou do conteúdo das mesmas ou podem se basear na alteração da semântica da troca de mensagens por meio da supressão de mensagens, duplicação das mensagens, inserção de mensagens espúrias ou atrasos no envio de mensagens.

---

[11] Sistemas que atendem à família de normas ***P**ortable **O**perating **S**ystem **I**nterface* (POSIX), designadas formalmente pela norma **IEEE 1003,** que visam a manutenção da compatibilidade entre sistemas operacionais.

[12] Vale salientar que na literatura muitas vezes testes de robustez são referidos como benchmarks de robustez mesmo quando não apresentam um procedimento padronizado ou métricas bem definidas.

ORCHESTRA é uma ferramenta para testar aplicações distribuídas e protocolos de comunicação por meio de injeção falhas em mensagens, tais como destruição, atraso, reordenação, duplicação, modificação e injeção de novas mensagens. A ferramenta utiliza a inserção de uma camada de injeção de falhas entre duas camadas consecutivas de uma pilha de protocolo (DAWSON et al., 1996).

No âmbito do projeto de Ambiente de Testes baseado em Injeção de Falhas de Software (ATIFS), a ferramenta *Ferry-clip with Software Fault-Injection Support Tool* (FSoFIST) incorpora técnicas de injeção de falhas por software para testes de conformidade em protocolos de comunicação (MARTINS, MATTIELLO-FRANCISCO, 2003).

O trabalho de Marsden et al. (2002) apresenta uma técnica de injeção de falhas para testes do *middleware* CORBA centrada em falhas de protocolo de rede. Nessa técnica, as falhas são injetadas no nível de aplicação antes de serem encapsuladas pela camada de transporte da rede. O trabalho apresenta a injeção de falhas no protocolo CORBA-IIOP e os erros na rede (*network corruption*) são emulados por meio da inserção de *bit-flips* nas mensagens. O trabalho considerou que o mecanismo utilizado revelou um conjunto grande de informações acerca dos tipos de falha do IIOP.

Wierman e Narasimhan (2008) apresentam um framework chamado VAJRA que tem a capacidade de injetar falhas acidentais e maliciosas como o objetivo de avaliar a capacidade de sobrevivência de sistemas distribuídos, principalmente, sistemas tolerantes a falhas Bizantinas[13]. Essa ferramenta utiliza como ponto de injeção as APIs comuns a vários sistemas tolerantes a falhas Bizantinas e considera: (i) falhas de perda, atraso e corrupção de mensagens; (ii) *crash* e paralisação de processos; e (iii) uso indevido de memória. Uma peculiaridade do VAJRA é que o injetor de falhas é distribuído

[13] Falhas Bizantinas são aquelas em que o componente defeituoso continua a executar, mas produz resultados incorretos e arbitrários.

e, assim, embora comandado por um processo central, são os nodos do sistema distribuído que têm a capacidade de injetar as falhas. Esse mecanismo permite que se injetem falhas em um nodo baseado no estado de outro nodo.

3.3.4. Considerações Sobre o Uso de Injeção de Falhas

Os elementos que interagem com infraestruturas de simulação baseadas em HLA são principalmente formados pelos modelos simulados (federados), pelo sistema operacional e pelo canal de comunicação. Por esse motivo, mudanças e falhas que afetam esses elementos foram consideradas para compor a carga de mudanças especificada nesta tese e as técnicas de injeção de falhas de software, de interface e em protocolos de comunicação podem ser diretamente utilizadas na geração e instanciação das mudanças e falhas definidas.

Por exemplo, a técnica de injeção de erros de interface baseada na ferramenta Ballista (KOOPMAN; DEVALE,1999) é utilizada no benchmark de robustez apresentado no Capítulo 7. A ideia de injetores de mudanças distribuídos é usada neste trabalho embora não tenha sido empregada a mesma filosofia do modelo apresentado na ferramenta VAJRA (WIERMAN; NARASIMHAN, 2008). E, para a composição da carga de mudanças, no que se refere a falhas de softwares, os trabalhos de Durães e Madeira (2004, 2006) são utilizados como referência.

Relativamente à geração e aplicação de falhas de software que compõem a carga de mudanças, especialmente as falhas relacionadas aos modelos (federados) cujos códigos fonte encontram-se disponíveis, podem ser utilizadas técnicas de mutantes. Ainda, a geração e aplicação das falhas de software em sistemas operacionais podem usar técnicas de injeção de falhas baseadas na ferramenta G-SWFIT (DURÃES; MADEIRA, 2002). Uma opção interessante para a aplicação de falhas no canal de comunicação é apresentada no trabalho de Marsden et al. (2002), já que o mesmo considera a injeção de falhas na camada de aplicação.

Em resumo, o conjunto de técnicas de injeção de falhas aqui apresentado não esgota a literatura, mas exemplifica e indica algumas possibilidades e referências. Essas técnicas devem ser avaliadas como alternativas para a implementação da carga de mudanças apresentada neste trabalho, podendo ser diretamente usadas ou adaptadas para este fim.

3.4. Trabalhos Relacionados

Nesta subseção são apresentados outros trabalhos que estão mais diretamente relacionados ao tema desta tese. Foram analisadas pesquisas que avaliaram a dependabilidade ou que apresentaram benchmarks de desempenho e robustez para simuladores e para a arquitetura HLA e, também, artigos sobre benchmarks de resiliência aplicados a outros domínios.

3.4.1. Avaliação de Desempenho e Dependabilidade em Simuladores

Em Wainer et al. (2011) foi apresentado um benchmarking para avaliar o desempenho de simuladores discretos, baseados em eventos, que usam o *framework Discrete Event System Specification* (DEV). Esse trabalho utiliza uma abordagem de geração automática de modelos com estrutura e comportamento variados para avaliar o desempenho desses ambientes, exercitando diferentes características das implementações. A abordagem apresentada é semelhante à usada no nosso trabalho no que tange à geração da carga de trabalho e à geração de mudanças nos modelos simulados, no entanto, o nosso trabalho difere do apresentado tanto no que se refere ao tipo de ambiente de simulação avaliado, o que modifica as características dessas cargas, quanto na flexibilidade de inserção e avaliação de novas mudanças nos modelos. Além disso, mudanças nos modelos simulados e na carga de trabalho compõem apenas um subconjunto das mudanças da carga de mudanças especificada.

Fernsler e Koopman (1999) apresentaram uma proposta de uso da ferramenta Ballista para benchmarks de robustez em RTIs HLA cujo principal objetivo era demonstrar a adaptação da ferramenta a outros domínios. Nesse trabalho são

apresentados resultados da avaliação de três implementações HLA distintas. O nosso trabalho, por meio de um estudo de caso, também apresenta um benchmark de robustez para RTIs HLA que teve como base a metodologia usada na ferramenta Ballista. O nosso objetivo com o estudo de caso apresentado é demonstrar o uso da metodologia e da abordagem de benchmarking de resiliência proposta nesta tese na definição de diferentes tipos de benchmarks, no caso um benchmark de robustez, mostrando o uso dos elementos especificados, a flexibilidade e a capacidade de generalização da abordagem, demonstrando que benchmarks de dependabilidade, desempenho e robustez podem ser vistos como instâncias de benchmarks de resiliência.

Nemeth (1999) avalia o uso da infraestrutura HLA em simulações de comunicação via rádio. Nesse trabalho o principal atributo avaliado é a latência em função do volume do fluxo de dados de áudio, dada a importância desse aspecto no domínio de comunicação por voz. Foram avaliadas duas implementações RTI HLA e conduzidos três tipos de teste de envio de dados, todos utilizando atributos e parâmetros do tipo *não-confiáveis* (*best effort*) que é o tipo mais adequado à troca de dados no domínio. No primeiro teste foi realizada uma análise do tamanho dos pacotes para verificar a sobrecarga (*overhead* em bytes) ocasionada por especificações de projeto de cada implementação no que tange à passagem de parâmetros e troca de dados (atributos). O segundo teste calculava o tempo gasto no recebimento de cada pacote, de cada atributo dentro do pacote e de cada byte de dados (*callback time*), com o objetivo medir o tempo gasto pela RTI no tratamento de diversos serviços. Finalmente, o terceiro teste mediu a latência do recebimento das mensagens trocadas entre dois federados em duas máquinas diferentes. Embora esse trabalho tenha apresentado uma comparação das RTIs HLA no domínio de voz e tenha analisado o atributo *latência*, ele difere do nosso por apresentar apenas uma comparação *ad hoc* das implementações, levando em consideração apenas um atributo específico, não considerando mudanças de contexto, nem aspectos gerais de um benchmark, tais como definição da carga de trabalho e validação do benchmark.

A sequência de artigos de Knight et al. (2001, 2002a, 2002b, 2002c) apresentam benchmarks de desempenho para RTIs HLA relativamente a atributos de *latência* e *rendimento* (*throughput*). Os artigos Knight et al. (2002a, 2002b) apresentam a definição de um benchmark flexível por meio do qual é possível avaliar RTIs variando um conjunto de parâmetros de teste, um a um, por exemplo, número de atributos e parâmetros trocados entre federados, tamanho dos atributos e parâmetros, número de *classes de objetos*, tipos de *classes de objeto,* tipos de *classes de interação* e número de federados. Foram apresentados os resultados do benchmark para três implementações diferentes. O artigo Knight et al. (2002c) apresenta detalhes da implementação do benchmark proposto nos artigos anteriores, mostrando tanto o projeto da ferramenta de benchmark, quanto o projeto dos federados utilizados como carga de trabalho. Embora a descrição do projeto da ferramenta mostre a possibilidade de variação de diversos parâmetros relativamente à escalabilidade e, também, da configuração do tipo de transporte, não há uma definição sistemática dessa variação, nem um projeto de sequência de variações e, além disso, os resultados dos testes foram apresentados exercitando os parâmetros de forma isolada, embora, algumas combinações de resultados tenham sido avaliadas em conjunto. O experimento utilizou um FOM fixo que tolda a flexibilidade de variação de objetos e parâmetros.

O conjunto de trabalhos de Knight et al. (2001, 2002a, 2002b, 2002c) apresentaram a evolução de um benchmark de desempenho que, tal como os estudos de caso apresentados nesta tese, avaliaram as RTIs HLA relativamente aos atributos *latência* e *rendimento*. Esses trabalhos consideram o projeto da carga de trabalho, uma ferramenta flexível para benchmark de desempenho e fornecem resultados comparativos. No entanto, diferentemente do nosso trabalho, esses trabalhos não consideraram de forma sistemática as mudanças às quais as RTIs estariam expostas, não avaliaram a variabilidade dos atributos medidos pelos benchmarks aplicados, tampouco definiram o benchmark seguindo um processo padronizado. Contudo, a definição do *template* para um federado típico e métricas de latência apresentadas nos trabalhos citados são utilizadas no nosso trabalho para a especificação dos

elementos do benchmark de resiliência e também nos estudos de caso apresentados.

Em Fitzgibbons et al. (2002) é apresentado um estudo acerca dos benchmarks realizados com RTIs HLA até aquele momento. O trabalho utiliza, a princípio, um conjunto de federados para benchmarking de desempenho disponíveis no *Defense Modeling and Simulation Office* (DMSO) para avaliação de RTIs relativamente à *latência*, *rendimento* e serviços de gerenciamento de tempo. O artigo considera implementações síncronas e assíncronas e apresenta os resultados dos três tipos de teste para três implementações HLA diferentes. Os autores argumentam que os benchmarks disponíveis avaliavam as RTIS, mas não consideravam a carga dos federados, ou seja, não avaliavam as RTIs em presença da federação. Esse trabalho apresenta, ainda, um projeto de um federado genérico que pode ser estendido para o uso de serviço de avanço de tempo e que permite avaliar tanto o atributo *latência* quanto o atributo *rendimento*, permitindo, ainda, a associação de carga de processamento aos federados. Não foram apresentados resultados utilizando o novo paradigma. Esse trabalho difere do nosso, principalmente, por ter apresentando um benchmark *ad hoc* e não validado. No entanto, o trabalho considerou aspectos importantes relativamente aos benchmarks de desempenho no domínio HLA – o uso de serviços de tempo, a carga dos federados e a importância da federação na avaliação da RTI. Nesta tese, esses aspectos foram considerados na definição da carga de trabalho e de mudanças.

Drake et al. (2003) desenvolveram um benchmark para avaliar o desempenho de implementações RTI HLA em cenários de tempo real. Para tanto, consideraram simulações que incorporavam hardware na malha de simulação (HITL) e conduziram um experimento no qual dois hardwares específicos eram intermediados por dois federados (*proxies*). O trabalho considera federados com cargas de processamento e foram utilizados três tipos de execução: (i) com nenhuma carga, na qual a troca de dados ocorria apenas entre os federados intermediários (*proxies*) (considerada como referência); (ii) com carga constante, na qual a carga fluía de um hardware emissor para o

hardware receptor por meio dos federados intermediários; e (iii) com carga processada, na qual imagens eram processadas nos hardwares gerando uma carga variada de trabalho. O benchmark de desempenho apresentado por Drake et al. (2003), uma extensão dos benchmarks apresentados em Knight et al. (2001, 2002a, 2002b), avalia, além dos atributos de *latência* e *rendimento*, também o atributo *jitter* e define uma carga de trabalho que considera o processamento dos federados e o hardware na malha da simulação, aspectos bastante importantes e que configuram mudanças relativamente aos federados básicos apresentados nos trabalhos anteriores. Mas, diferentemente do benchmark de resiliência apresentado nesta tese, o trabalho não considera as mudanças que podem ter impacto em uma RTI ou o aspecto de variabilidade das métricas apresentadas e não utiliza nenhum processo sistemático para a especificação do benchmark.

Malinga e Roux (2009) avaliaram o desempenho de três RTIs HLA relativamente à latência na troca de mensagens entre dois federados executados em duas máquinas diferentes. Basicamente utilizaram um único parâmetro e mediram a média de tempo de atualização deste atributo (*round-trip*), variando o seu tamanho. Dois resultados distintos, uma para parâmetros configurados para serem trocados em comunicação confiável (*reliable*) e outro para comunicação não-confiável (*best effort*) foram apresentados. Embora esse trabalho seja mais recente, o benchmark apresentado é bastante simples, considera apenas dois federados e um atributo de desempenho, e adota um processo *ad hoc* para a sua definição. Além disso, não foi feita uma especificação consistente e embasada da carga de trabalho e não se considera nenhuma mudança para além do tipo de comunicação (confiável e não-confiável).

Os trabalhos relacionados a benchmarks de infraestruturas HLA apresentados, centrados no desempenho das RTIs, aspecto bastante relevante em simuladores e infraestruturas distribuídas, foram utilizados como referência para a nossa pesquisa na definição de elementos da carga de trabalho e de mudança, de *templates* dos federados típicos e de métricas de desempenho.

Por exemplo, um dos atributos da carga de trabalho apresentada nesta tese define a carga de processamento dos modelos (FITZGIBBONS et al., 2002; DRAKE et al., 2003) e a carga de mudanças inclui variações na malha de simulação (DRAKE et al., 2003). No entanto, esses trabalhos, mesmo os que apresentaram detalhes de procedimentos, cargas de trabalho e ferramentas de benchmark, se ativeram a poucas variações de contexto nos experimentos e propuseram soluções *ad hoc*. Por esse motivo, embora os trabalhos apresentados tenham servido de base para a nossa pesquisa e tenham relação direta com os estudos de caso apresentados nesta tese, nós acreditamos que o nosso trabalho dá um passo à frente no que se refere a benchmarks para infraestruturas HLA, já que considera aspectos de dependabilidade e resiliência, sistematiza as mudanças às quais a infraestrutura pode estar exposta e utiliza uma metodologia para especificação de abordagens de benchmarking.

3.4.2. Benchmarking de Resiliência

Bondavalli et al. (2009) e Wolter et al. (2012) apresentam um conjunto de temas relacionados à resiliência de sistemas computacionais, entre os quais destacam-se: técnicas de modelagem, pesquisas sobre métricas de resiliência, injeção e prevenção de falhas e técnicas de teste. O trabalho de Vieira et al. (2012), por exemplo, apresenta o estado da arte em benchmarking de resiliência e indica um conjunto de desafios de pesquisa na área, tais como a identificação de elementos de benchmarks de resiliência, a definição de um processo de benchmarking que possa ser aceito pelas partes interessadas e adaptado a diferentes produtos dentro de um domínio, a generalização de *frameworks* reutilizáveis (componentes e ferramentas) para a criação de benchmarks em diferentes domínios e aspectos da disseminação de abordagens de benchmarking. Os trabalhos compilados em Wolter et al. (2012) foram usados como embasamento teórico para o nosso trabalho e os desafios de pesquisa como motivadores.

Há na literatura trabalhos que apresentam definições, propostas e modelos para resiliência em redes de computadores (TRIVEDI et al., 2009; STERBENZ

et al., 2010; SCHAEFFER-FILHO et al., 2011), entre outros. Entretanto, até onde sabemos, não há uma proposta consolidada de abordagens de benchmarking que considere a resiliência nesse domínio.

Alguns trabalhos têm como foco especificamente a definição de métricas de resiliência. O trabalho de Henry e Ramirez-Marquez (2012) define uma métrica genérica de resiliência que pode ser aplicada a diferentes tipos de sistemas. Essa métrica é baseada em figuras de mérito que representam diferentes aspectos do sistema e quantifica as relações entre a medida da figura de mérito no estado inicial, no estado pós-perturbação e no estado pós-recuperação do sistema, em função tempo. Barker et al. (2013) definem métricas, no domínio de redes de computadores, que levam em consideração o impacto de cada componente relativamente à resiliência geral dos sistemas com o objetivo de guiar as políticas de redução de vulnerabilidades. Uma métrica de resiliência genérica baseada em um índice de perda de serviços de um sistema quando em face de mudanças pode ser encontrada em Almeida et al.(2013). O trabalho de Tamvakis e Xenidis (2013) apresenta uma pesquisa abrangente e compara métricas de resiliência definidas no domínio de infraestruturas.

No que tange a métricas de resiliência, as métricas definidas em Henry e Ramirez-Marquez (2012) e Almeida e Viera (2013) são consideradas para uso nesta tese, já que são métricas que podem ser mensuradas de forma empírica e são fáceis de serem usadas e compreendidas, atendendo a propriedade de simplicidade de benchmarks de resiliência. A métrica proposta por Almeida e Vieira (2013) foi efetivamente usada em dois dos estudos de caso apresentados no Capítulo 7.

Almeida e Vieira (2011) apresentaram uma discussão acerca das perspectivas de Benchmarks de Resiliência para sistemas autoadaptativos que foi utilizada como ponto de partida para a proposta do nosso trabalho no domínio de infraestruturas de simuladores HLA. O trabalho de Almeida e Vieira (2012b) discute a importância de uma carga de mudanças na avaliação de sistemas autoadaptativos. Já a técnica usada para a definição da representatividade de

uma carga de mudanças para benchmarks de resiliência baseada em análise de riscos, que também é aplicada no nosso trabalho, foi primeiramente proposta também para o domínio de sistemas autoadaptativos e pode ser encontrada em Almeida e Vieira (2012a). Camara et al. (2013a) apresentam um modelo para avaliação comparativa de mecanismos de adaptação de sistemas autoadaptativos e formalizam várias definições acerca do processo de avaliação; e Camara et al. (2014) apresentam a comparação da resiliência de sistemas autoadaptativos baseados em arquitetura e sistemas autoadaptativos convencionais (baseados em código).

Em relação a benchmarks de resiliência, paralelamente ao desenvolvimento desta tese, os artigos de Almeida e Vieira (2011, 2012a, 2012b) e CAMARA et al. (2013a) conceituaram e formalizaram vários elementos no domínio de sistemas autoadaptativos. No nosso domínio, as ideias similares foram consolidadas e elementos e conceitos comuns foram diretamente utilizados, outros conceitos foram referenciados ou foram adaptados e estendidos. Os elementos e conceitos foram sistematizados por meio da metodologia de especificação de benchmarking de resiliência e da linguagem de representação e disseminação de benchmarks, propostas nos Capítulos 4 e 6 respectivamente, e são apresentados em maiores detalhes ao longo de toda a tese. Esta tese difere dos trabalhos apresentados no que tange ao domínio da aplicação do benchmark de resiliência e dá um passo a mais por meio da definição de uma metodologia e de uma linguagem de representação e disseminação de benchmarks.

3.5. Considerações Finais

Este capítulo apresentou os conceitos de dependabilidade e resiliência, bem como a definição e a contextualização de benchmarking de resiliência para sistemas computacionais. Foram também apresentadas ferramentas e técnicas de injeção de falhas, principalmente, injeção de falhas de software, de interface e baseadas em mensagem, já que essas técnicas podem ser usadas, diretamente ou de forma adaptada, na implementação da geração e instanciação da carga de mudanças do benchmarking de resiliência proposto.

Por meio da descrição dos trabalhos relacionados buscou-se apresentar o estado da arte em domínios correlatos ao desta tese, procurando mostrar em que ponto esses trabalhos nos serviram como base e guia na execução do nosso e no que o nosso trabalho se diferencia dos anteriores.

4 METODOLOGIA PARA ESPECIFICAÇÃO DE BENCHMARKS DE RESILIÊNCIA

Neste capítulo é apresentada uma metodologia para a definição, instanciação e representação de benchmarks de resiliência. Esta metodologia foi proposta para apoiar o processo de especificação de uma abordagem de benchmarking de resiliência no domínio de infraestruturas de simuladores de satélite que utilizam o padrão HLA, mas é genérica e pode ser estendida a outros domínios.

O principal objetivo da metodologia proposta é tornar o processo de especificação de benchmarks de resiliência mais sistemático, definindo as etapas e passos a serem executados, a sua ordenação, interdependência e inter-relação, apontando, também, diretrizes gerais e, quando pertinente, alternativas de técnicas para a execução do passo ou etapa.

Este capítulo apresenta as fases, etapas e passos da metodologia, enquanto os demais capítulos da tese apresentam a sua aplicação. Assim, o Capítulo 5 apresenta o uso da metodologia na especificação de um benchmark de resiliência no domínio de infraestruturas HLA, o Capítulo 6 apresenta a linguagem de representação de benchmark de resiliência nesse domínio e o Capítulo 7 apresenta a instanciação de benchmarks concretos.

4.1. Visão Geral da Metodologia

A metodologia proposta neste trabalho compreende 8 etapas, divididas em três fases: a **definição** do benchmark, a **instanciação** de benchmarks concretos, e a **representação** de elementos definidos e de benchmarks instanciados.

Uma visão abrangente da metodologia é apresentada na Figura 4.1. É importante observar que a figura apresenta a interdependência lógica entre os diferentes passos a serem executados na especificação de um benchmark de resiliência, mas não necessariamente a relação temporal entre eles. A coluna *Experimentos* mostra a correlação entre os passos da metodologia e os elementos que são materializados durante a implementação e execução dos experimentos de benchmark.

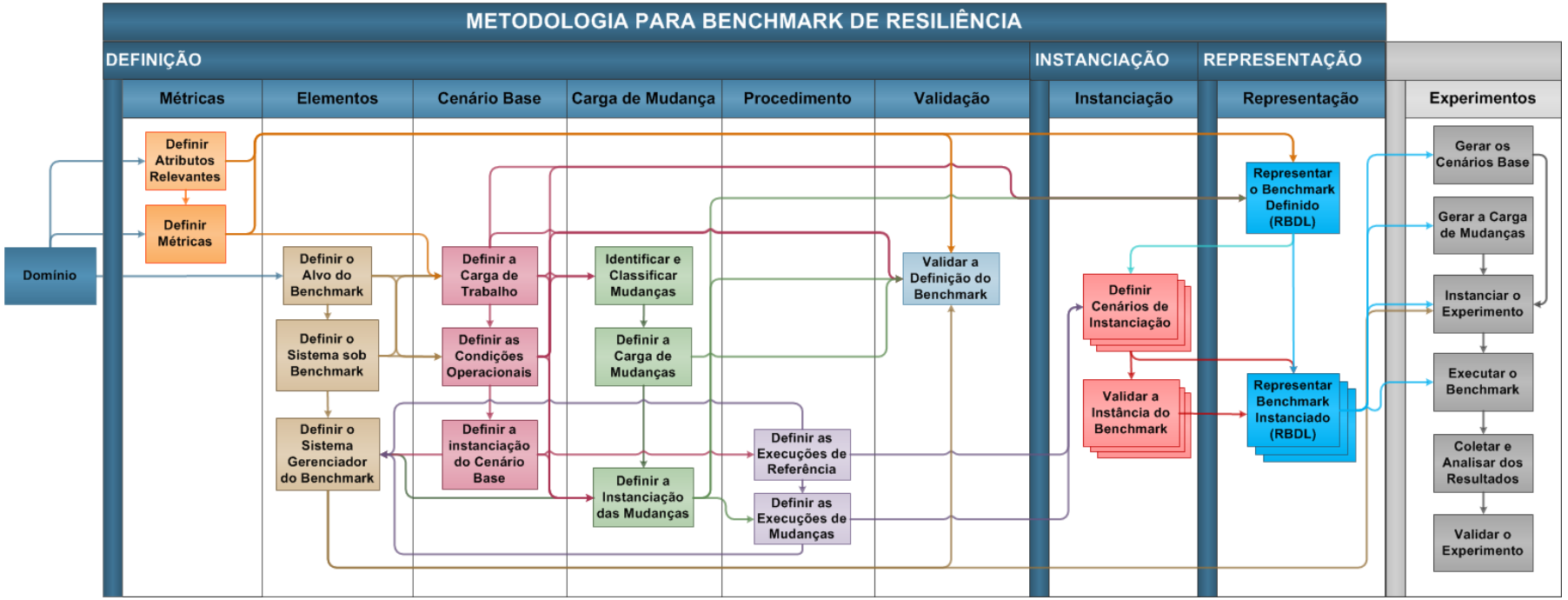


Figura 4.1 - Metodologia - Benchmark de Resiliência.

A especificação de benchmarks de resiliência usando a metodologia proposta se divide em três fases:

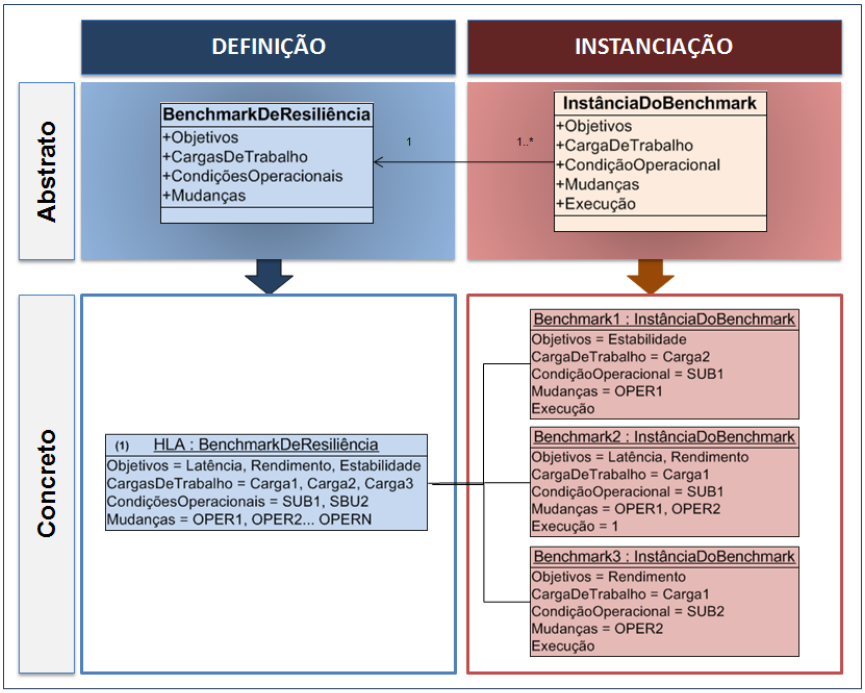
1. **Definição do benchmark,** que compreende: (i) definir o conjunto de atributos relevantes e métricas para o domínio em questão; (ii) delimitar os elementos do benchmark, i.e., o alvo da avaliação, os sistemas envolvidos na avaliação e a arquitetura da ferramenta de aplicação do benchmark; (iii) definir cargas de trabalho relevantes para o domínio do problema, as condições operacionais e os métodos de instanciação desses elementos; (iv) definir uma carga de mudanças que represente as mudanças às quais um sistema pode estar exposto; (v) definir as regras de aplicação dos experimentos; e (vi) atestar a validade dos elementos definidos (atributos e métricas, cargas de trabalho, etc.).

2. **Instanciação de benchmarks concretos,** essa etapa compreende, para cada benchmark concreto: (i) instanciá-lo a partir dos elementos predefinidos, estabelecendo os atributos a serem avaliados, as métricas a serem usadas na avaliação, a carga de trabalho a ser executada, as mudanças a serem aplicadas, etc.; e (ii) atestar a validade dos elementos da instância de benchmark concreto e definir os elementos a serem validados durante a execução do experimento de benchmark.

3. **Representação dos elementos do benchmark,** essa fase compreende: (i) representar o conjunto de elementos definidos e validados usando uma linguagem de representação; e (ii) representar benchmarks concretos usando uma linguagem de representação.

Importa frisar que, dada a diversidade, recorrência e imprevisibilidade das mudanças em um benchmark de resiliência, a metodologia proposta separa a fase de definição de um benchmark da fase de instanciação dos benchmarks que serão efetivamente implementados e executados. Um benchmark de resiliência concreto pode ser visto como uma instância do benchmark definido no qual os elementos que o compõem são um subconjunto dos elementos previamente especificados. Essa abordagem permite que, a partir da definição

do benchmark, se possa derivar vários benchmarks concretos para a avaliação dos sistemas em diferentes contextos, quando em presença de diferentes mudanças ou com foco em diferentes objetivos. Um experimento de benchmark sempre será realizado a partir de um benchmark concreto. A Figura 4.2 ilustra esse paradigma, apresentando os níveis abstrato e concreto, tanto para a Definição como para a Instanciação do Benchmark.



(1) Os elementos que compõe a parte concreta do benchmark (definição e instanciação) são meramente ilustrativos: atributos (Latência, Rendimento e Estabilidade), Cargas de trabalho (Carga1, Carga2 e Carga3), mudanças (OPER1, OPER2 até OPERN).

Figura 4.2 - Níveis de Abstração - Definição e Instanciação.

4.2. Benchmark de Resiliência: Definição

As diretrizes gerais para a fase de definição de um benchmark de resiliência são apresentadas nas subseções seguintes.

4.2.1. Identificação das Métricas

Esta etapa consiste em dois passos: (i) definir os atributos relevantes para o domínio do problema; e (ii) definir o conjunto de métricas a serem consideradas em benchmarks concretos. A Figura 4.3 resume os passos desta etapa.

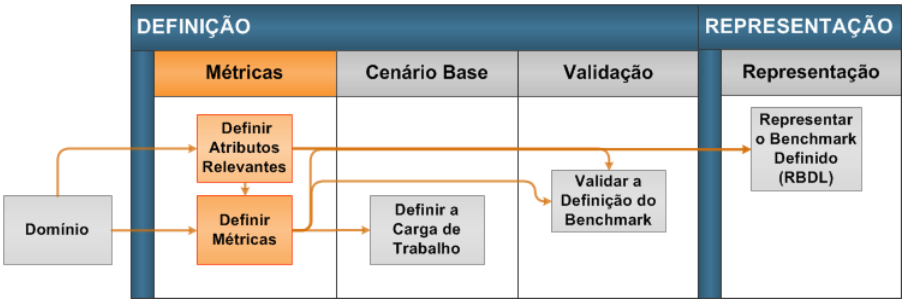
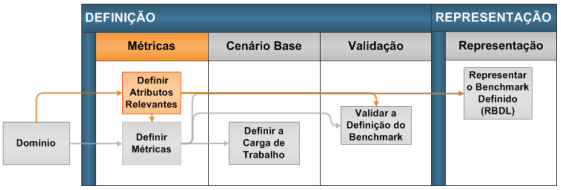


Figura 4.3 - Definição de Métricas.

4.2.1.1. Definir Atributos Relevantes



Na especificação de um benchmark de resiliência devem ser definidos quais são os atributos mais relevantes em um dado domínio ou, em outras palavras, quais atributos teriam impacto maior na resiliência geral do sistema caso atingissem níveis indesejados. Simplicidade, viabilidade e baixo custo são propriedades importantes para que um benchmark seja aceito como ferramenta de avaliação e, por isso, nem sempre é possível ou viável medir o conjunto completo de atributos de dependabilidade.

A dependabilidade é traduzida por um conjunto de atributos que devem ser atendidos em níveis adequados e, além disso, tanto a lista de atributos, quanto os níveis aceitáveis da presença dos mesmos variam de acordo com o domínio do sistema a ser desenvolvido ou avaliado (RUS et al., 2003; WEINSTOCK et al., 2004). Por exemplo, o atributo *integridade* da informação pode ser extremamente importante no domínio de banco de dados, enquanto, *disponibilidade* pode ser mais relevante do que a integridade dos dados no domínio de Serviços Web. O Capítulo 3 apresenta um conjunto de atributos de dependabilidade definidos na literatura e que servem de base para a escolha de atributos de dependabilidade relevantes para um domínio específico.

De acordo com Rus et al. (2003), a dependabilidade de um sistema só pode ser estimada e medida quando o mesmo está sendo utilizado. Essa definição vem ao encontro dos objetivos de um benchmark, já que este se baseia primordialmente em medidas diretas de um sistema em funcionamento. Assim,

na composição do conjunto de atributos a serem avaliados devem ser considerados, principalmente, os atributos de dependabilidade orientados à utilização do sistema e devem ser usadas as definições mais amplamente aceitas ou as definições mais afins ao domínio do problema.

Metodologias e técnicas de elicitação de requisitos não funcionais podem ser adaptadas para o levantamento do conjunto de atributos de dependabilidade potencialmente relevante para o domínio analisado. Muitas dessas metodologias e técnicas utilizam entrevistas e pesquisas com *stakeholders* como parte do processo. Trabalhos como os de Mostert e Von Solms (1995), Boehm et al. (2004), Basili et al. (2004) e Liu et al. (2009) propõem técnicas que podem ser adaptadas para a execução deste passo.

O levantamento de atributos de dependabilidade definidos na literatura e afins ao domínio do problema, bem como pesquisas e entrevistas com especialistas podem auxiliar na execução do passo em questão. Como resultado do mesmo, devem-se obter o conjunto de atributos relevantes e a padronização da definição a ser adotada visando minimizar divergências de interpretação.

A Figura 4.4 resume as entradas e as saídas esperadas nesse passo.

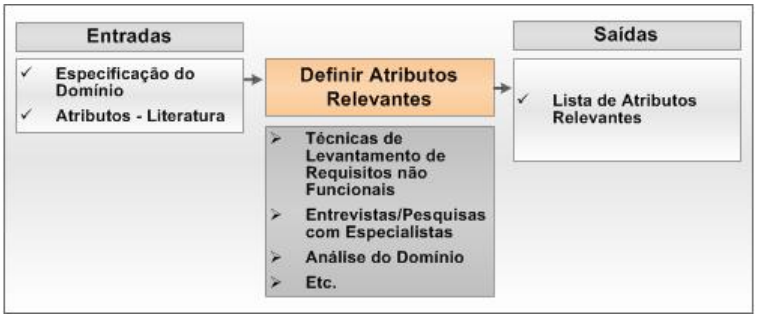
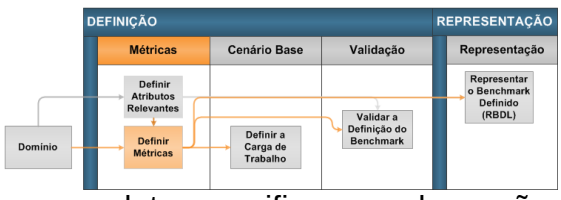


Figura 4.4 - Definir Atributos Relevantes - Entradas e Saídas.

4.2.1.2. Definir Métricas para Atributos de Dependabilidade e Resiliência



A definição das métricas é o cerne da especificação de um benchmark uma vez que é por meio das suas quantificações que se pode comparar produtos, verificar a adequação de

produtos aos objetivos dos usuários, identificar as necessidades de melhorias, etc.

Uma métrica pode ser vista como uma abstração que reduz a complexidade dos atributos do sistema, atribuindo aos mesmos um significado em termos avaliativos, classificativos e comparativos. Um atributo, normalmente, pode ser medido por uma ou mais métricas. Tipicamente, as métricas devem refletir dois princípios fundamentais: a relação entre os sistemas relativamente a um dado atributo deve ser refletida pela relação correspondente entre suas medidas, bem como a operação possível em um atributo do sistema deve se refletir na operação correspondente em suas medidas (BÖHME; REUSSNER, 2008).

A norma da Organização Internacional para padronização (ISO/ 2001) indica que as métricas que têm por objetivo comparar produtos devem ser: *objetivas* e especificadas em procedimentos escritos e acordados, avaliando um conjunto de atributos dos produtos; devem ser *empíricas*, sendo obtidas a partir da observação dos sistemas por meio de experimentos ou a partir de questionários psicometricamente válidos; devem ter uma *escala válida*, baseada em itens de igual valor ou de um valor conhecido; e devem ser *reproduzíveis*, ou seja, a medição deve obter os mesmos resultados com as apropriadas tolerâncias quando obtidas por diferentes pessoas, em diferentes ocasiões.

Além das propriedades descritas acima, as métricas definidas no contexto de benchmarks de resiliência devem considerar, ainda, os seguintes objetivos: (i) as métricas devem simples e de fácil compreensão, pois um benchmark visa ser uma ferramenta facilmente utilizada pelos seus usuários; (ii) as métricas devem representar as principais características do atributo medido, ou seja, não se deve considerar em um benchmark concreto um grande número de métricas que possa confundir a análise global do atributo, nem considerar métricas que não o representem; (iii) as medidas obtidas diretamente por meio dos experimentos não devem ser extrapoladas; e (iv) devem ser considerados aspectos de resiliência da dependabilidade, em outras palavras, métricas que meçam a variabilidade das demais métricas de dependabilidade.

Não há uma técnica ou metodologia única para definir métricas para um benchmark, mas, de forma análoga ao levantamento dos atributos relevantes para o domínio, técnicas de elicitação de requisitos não funcionais podem auxiliar na determinação de métricas a serem utilizadas. Por exemplo, o paradigma Meta/Questão/Métrica (Goal/Question/Metric - GQM), inicialmente definido por Basili e Weiss (1984) e amplamente utilizado para definição de métricas de software, pode ser usado para esse propósito. O GQM é um modelo *top-down* cujo objetivo é obter a definição de métricas de qualidade para produtos e processos de software. Em uma primeira etapa, são identificados o objeto, o propósito da avaliação, as metas de qualidade a serem avaliadas e a partir de qual ponto de vista será feita essa avaliação. A seguir, para cada meta, com base principalmente nas características do objeto a ser avaliado, deve ser levantado um conjunto de perguntas que avaliem e verifiquem a meta, e essas perguntas serão respondidas por uma ou mais métricas.

A norma ISO/IEC 9126-2 (2002) define um conjunto de métricas para avaliação da qualidade do software e, sempre que possível, devem-se utilizar métricas definidas por essa norma ou métricas já amplamente aceitas na literatura. Nos demais casos, pode-se utilizar a metodologia GQM para identificar as métricas pertinentes.

Em relação às métricas específicas de resiliência, deve-se levar em consideração a capacidade de um sistema em *tolerar*, *acomodar* e se *recuperar* de mudanças, mantendo a qualidade do serviço prestado. Assim, podem-se medir: (i) a relação entre a qualidade dos serviços prestados pelo sistema em presença de perturbação relativamente à qualidade dos serviços prestados em condições normais de operação (ALMEIDA; VIEIRA, 2013); (ii) a estabilidade do sistema na presença de perturbações relativamente ao seu comportamento normal; (iii) o quanto de perturbação um sistema pode tolerar sem perder níveis de dependabilidade; (iv) o tempo gasto por um sistema para voltar ao seu estado normal após ter sido submetido a uma dada mudança; ou,

ainda, (v) a relação entre o nível de serviço prestado antes da perturbação e após a sua recuperação (HENRY; RAMIREZ-MARQUEZ, 2012).

A obtenção de uma medida única de resiliência será sempre dependente dos atributos de dependabilidade que estão sendo medidos e de sua relevância para o domínio do problema ou para os usuários do benchmark. Um índice único de resiliência para um sistema poderia ser obtido, por exemplo, por meio da definição de pesos para atributos avaliados em conjunto em uma instanciação do benchmark. A análise de especialistas pode ser usada para atribuição de pesos a cada um dos atributos sempre que houver interesse na determinação de um índice geral de resiliência. Vale ressaltar que, embora haja a possibilidade de definição de uma métrica única que englobe a avaliação de variabilidade de vários atributos, o uso desta abordagem reduz a precisão da comparação, não deixando claro em quais aspectos um produto tem vantagem comparativa em relação a outro.

Em resumo, neste passo devem ser definidas uma ou mais métricas para cada atributo relevante elencado no passo anterior, incluindo métricas específicas de resiliência. Em um paralelo com a norma ISO/IEC 9126-2 (2002), para cada métrica definida, deve ser especificado: (i) sua descrição; (ii) seu objetivo; (iii) a fórmula a ser utilizada; (iv) a forma como a medida obtida por meio da métrica deve ser avaliada e comparada (p. e. quanto menor melhor etc.); e (v) a sua unidade.

A Figura 4.5 resume as entradas e as saídas esperadas neste passo.

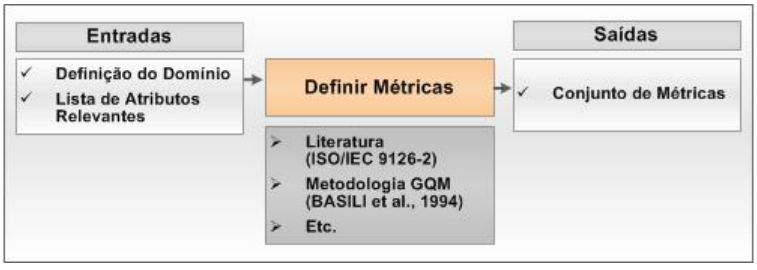


Figura 4.5 - Definir Métricas - Entradas e Saídas.

4.2.2. Elementos

Os elementos que compõe a instalação e arquitetura de um Benchmark - Sistema Alvo, Sistema Sob Benchmark e Sistema Gerenciador de Benchmark - , bem como suas definições e nomenclaturas estão estabelecidos na literatura e são transversais aos diferentes tipos de benchmark (DBENCH, 2004; VIEIRA, 2005; VIEIRA et al., 2008), devendo ser adaptadas a cada domínio específico.

Nesta etapa devem ser definidos os elementos que compõem a arquitetura geral do benchmark para o domínio de interesse por meio da definição dos componentes do sistema a serem avaliados, dos sistemas envolvidos na avaliação e da arquitetura necessária para a realização dos experimentos de benchmark.

A Figura 4.6 descreve os passos desta etapa.

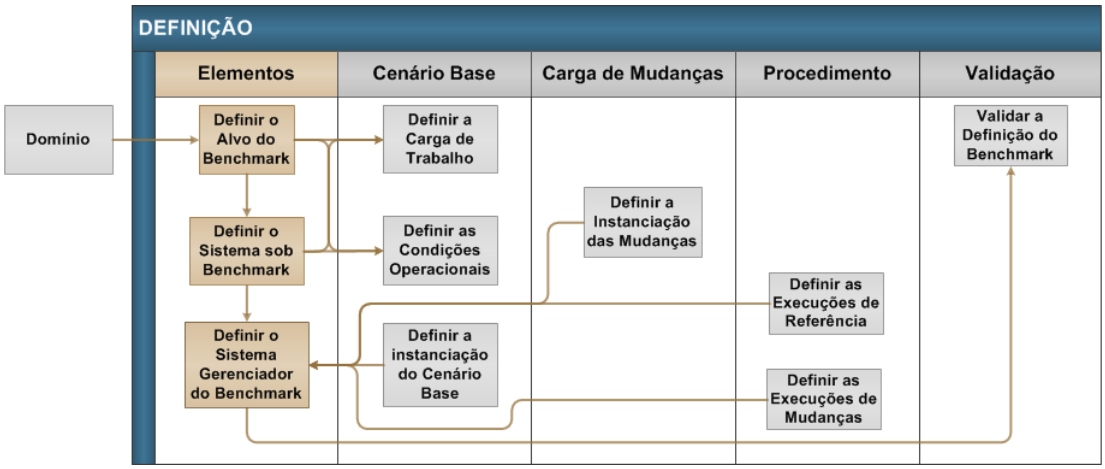
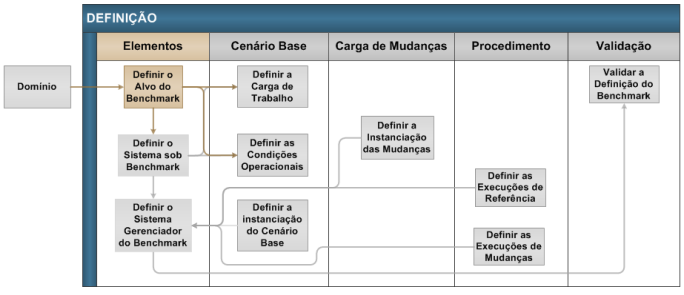


Figura 4.6 - Definição dos Elementos do Benchmark.

4.2.2.1. Definir o Alvo do Benchmark (*Benchmark Target* – BT)



Uma vez definido o domínio do problema, neste passo deve-se especificar com exatidão o que será, dentro desse domínio, o alvo do benchmark, ou seja, deve-se

definir e delimitar o sistema ou a parte do sistema que será caracterizada e avaliada em relação aos atributos de dependabilidade e resiliência a serem medidos.

A saída esperada para este passo é a descrição das características do sistema a ser avaliado e sua arquitetura. A Figura 4.7 apresenta as entradas e as saídas do passo.

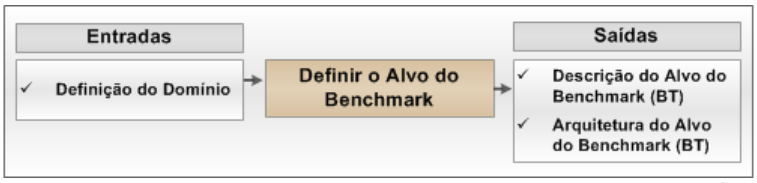
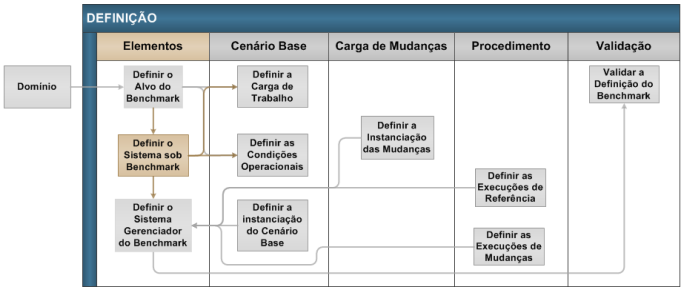


Figura 4.7 - Definir o Alvo do Benchmark - Entradas e Saídas.

4.2.2.2. Definir o Sistema Sob Benchmark (*System Under Benchmark* – SUB)



Neste passo devem ser definidos todos os elementos que interagem diretamente com BT, mas que não serão avaliados pelo processo de benchmark. Fazem parte do Sistema Sob Benchmark os elementos que dão suporte e/ou auxiliam na execução do benchmark, por exemplo, um injetor de falhas, um instanciador de mudanças, etc.; além de todos os elementos de hardware e software necessários para a execução do BT (DBENCH, 2004; VIEIRA, 2005; DURÃES et al., 2008; COSTA et al., 2008).

A saída deste passo deve ser a lista dos elementos do SUB e a sua arquitetura. A Figura 4.8 apresenta as entradas e as saídas esperadas para o mesmo.

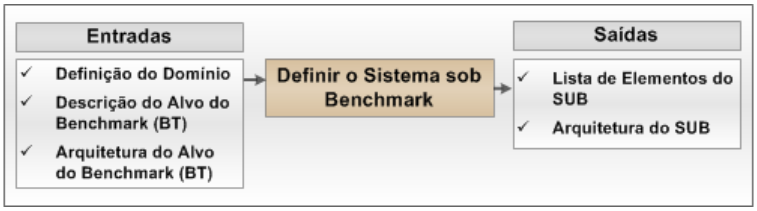
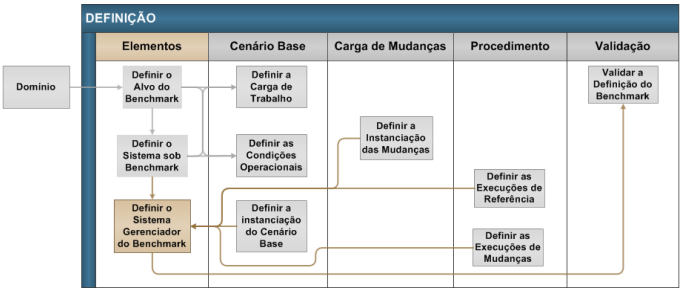


Figura 4.8 - Definir o Sistema sob Benchmark - Entradas e Saídas.

4.2.2.3. Definir o Sistema Gerenciador do Benchmark



A partir do Sistema Gerenciador do Benchmark (SGB) serão gerados e realizados os experimentos do benchmark. O SGB deve ser projetado a partir dos principais elementos do benchmark, tais como cargas de trabalho, carga de mudanças, regras de instanciação dessas cargas, regras de execução dos experimentos e medidas a serem coletadas e analisadas.

De maneira geral, o SGB deve ser composto por três subsistemas: (i) subsistema gerador de cargas, responsável por gerar as cargas de trabalho e mudanças a serem utilizadas durante os experimentos; (ii) subsistema coordenador, responsável por coordenar a execução do benchmark instanciando apropriadamente as cargas de trabalho e mudanças e executando os experimentos; e (iii) subsistema coletor e analisador de resultados, responsável por coletar e analisar os dados dos experimentos, por calcular as medidas e por gerar os resultados sumarizados.

A arquitetura e os requisitos específicos de cada um desses subsistemas dependem do domínio do benchmark. Em termos temporais, deve-se definir cada subsistema do SGB assim que se tenha realizado os passos de definição da instanciação de cargas de trabalho e mudanças e os passos da fase de definição de procedimentos e regras do benchmark. O Sistema Gerenciador do Benchmark se materializará no ferramental do benchmark.

Espera-se como saída deste passo, a definição do SGB por meio da descrição da sua arquitetura e da especificação dos seus requisitos gerais. A Figura 4.9 resume as entradas e as saídas do passo.

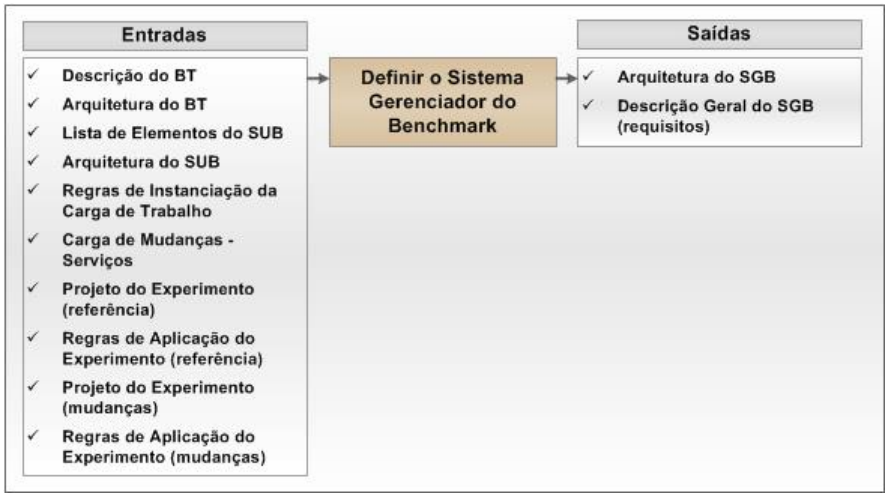


Figura 4.9 - Definir o Sistema Gerenciador de Benchmark - Entradas e Saídas.

4.2.3. Definição do Cenário Base

Nesta etapa devem ser analisadas, caracterizadas e definidas as cargas de trabalho às quais o Alvo do Benchmark estará exposto durante a execução dos experimentos, bem como as condições típicas de operação do sistema. O *Cenário Base* inclui a carga de trabalho e as suas características estáticas e dinâmicas e também as configurações do sistema e as condições de funcionamento - equipamentos e tecnologias utilizadas, perfil de uso esperado para o SUB, etc. Um *Cenário Base* está relacionado a uma execução de referência e compreende todos os elementos que a caracterizam e que podem estar sujeitos a mudanças.

Nesta etapa deve-se, também, definir a forma como o *Cenário Base* será instanciado durante o experimento de benchmark. Essa definição servirá de base para o projeto e implementação da ferramenta de benchmark.

A Figura 4.10 apresenta a etapa de definição do *Cenário Base*.

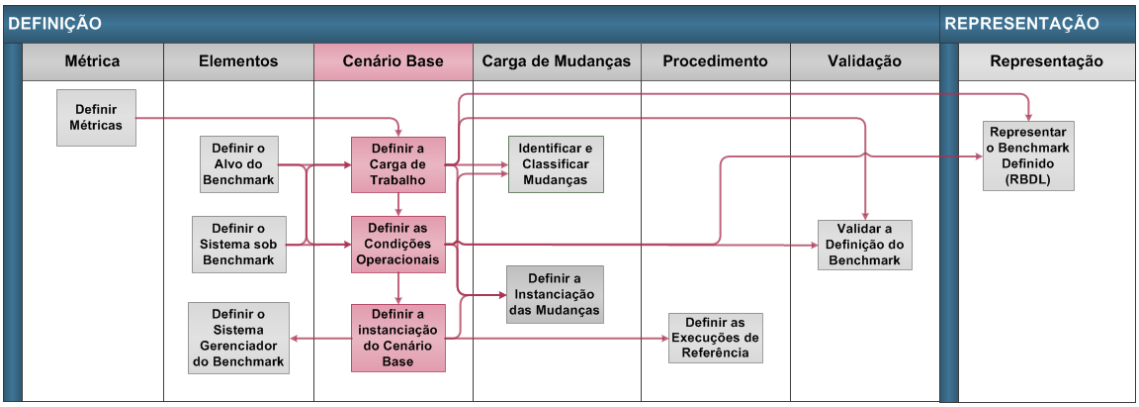
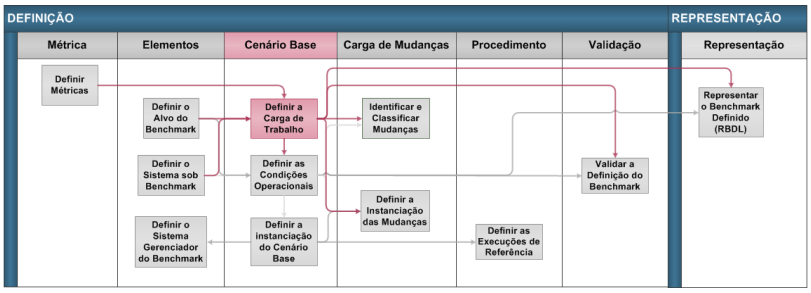


Figura 4.10 - Definição do Cenário Base.

4.2.3.1. Definir a Carga de Trabalho



A caracterização de cargas de trabalho é um requisito para vários estudos de desempenho, tais como agendamento, planejamento, balanceamento de carga, análise de escalabilidade, configuração e ajustes (*tunning*) de sistemas, benchmarks, etc. Na prática, a carga de trabalho define o modelo de uso do sistema na forma de parâmetros quantitativos e funcionalidades, devendo ser típica de forma a representar uma carga real à qual o sistema estaria exposto. Com o objetivo de cumprir as metas de avaliação de um sistema, uma carga de trabalho deve ser representativa, compacta e precisa (CALZAROSSA; SERAZZI, 1993), deve descrever as características estáticas e dinâmicas do sistema (ELNAFFAR; MARTIN, 2002) e, deve, ainda, ser concisa e portável para os diferentes ambientes a serem avaliados.

Os sistemas, dependendo do seu domínio, possuem cargas de trabalho específicas, por exemplo, um sistema OLTP terá como carga de trabalho consultas SQL, transações de atualização, criação e exclusão de tabelas, etc. (TPC, 2010a); um sistema de Servidores Web terá como carga de trabalho as

requisições típicas aceitas por páginas Web (DURÃES et al., 2008); um computador de bordo terá como carga de trabalho os serviços *Packet-Utilization Standard* (PUS) executados (COSTA et al., 2008); entre outros exemplos. Também em um mesmo domínio, a definição de uma carga de trabalho deve levar em consideração o interesse em um dado contexto de uso do sistema, por exemplo, pode-se avaliar o desempenho de um gerenciador de banco de dados considerando um conjunto de clientes executando consultas e efetuando uma série de atualizações em tabelas como em um sistema comercial (reserva de passagens, telecomunicação, etc.) (TPC, 2010a), ou pode-se avaliar o desempenho de um banco de dados quando são executadas consultas complexas como as utilizadas em sistemas de suporte à decisão (TPC, 2012). A carga de trabalho deve, ainda, levar em consideração características dinâmicas, como volume, questões temporais, etc.

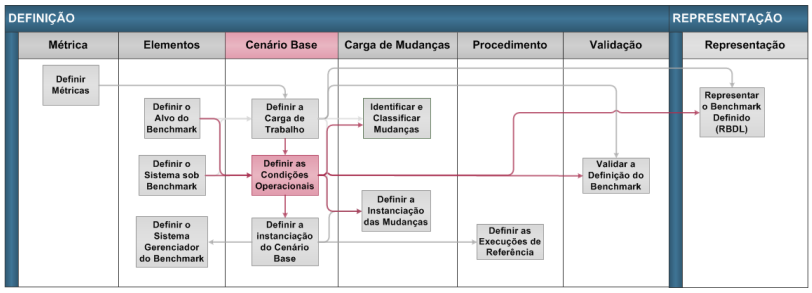
Elnaffar e Martin (2002) apresentaram um levantamento abrangente de técnicas de caracterização de cargas de trabalho em alguns domínios, tais como aplicações cliente e servidor, OLTP, sistemas paralelos e aplicações Web. Em cada uma dessas áreas, a fim de determinar o comportamento estático e dinâmico das cargas de trabalho, foram abordadas diferentes técnicas, tais como técnicas estatísticas, de agrupamento, etc. Esse trabalho também definiu uma metodologia para a caracterização de cargas de trabalho que compreende as seguintes etapas: (i) *análise de requisitos*, na qual são definidos os níveis de abstração, os componentes básicos da carga de trabalho (p. ex. scripts, transações SQL, instruções de CPU, pacotes de rede), os parâmetros de carga de trabalho que podem ter impacto no desempenho do sistema (p. ex. o tamanho de um pacote de rede, o número de instruções de I/O) e critérios para avaliar a representatividade do modelo; e (ii) *construção do modelo*, que compreende a coleta de dados na qual os parâmetros selecionados são coletados por um determinado período de tempo, a criação de um banco de dados onde os dados coletados são armazenados e processados, e a análise estática e dinâmica dos dados usando ferramentas específicas (i.e., estatísticas, agrupamento, etc.).

Desta forma, neste passo, a carga de trabalho à qual o sistema estará sujeito deve ser definida ou utilizando contextos bem conhecidos e delimitados como, por exemplo, modelos em uma infraestrutura de simulação, ou utilizando metodologias como a descrita por Elnaffar e Martin (2002) quando se fizer necessário modelar comportamentos de uso do sistema, por exemplo, acessos de usuários ao sistema, tráfegos típicos de uma dada rede de computadores, etc.

É esperado como resultado deste passo a definição da arquitetura da carga de trabalho e a definição dos seus elementos e parâmetros principais. A Figura 4.11 resume as entradas e as saídas esperadas nesse passo.

Figura 4.11 - Definir as Cargas de Trabalho - Entradas e Saídas.

4.2.3.2. Definir as Condições Operacionais



Em um benchmark de resiliência, a execução de referência será representada por um *Cenário Base* caracterizado, para além da carga de trabalho, também pelas condições de operação do sistema. A definição das *Condições Operacionais* compreende a caracterização do BT e do SUB tanto no que tange a equipamentos e sistemas, quanto no que se refere a aspectos de configuração e uso típico dos seus elementos.

São esperados como saídas deste passo, uma lista dos elementos que definem a configuração de cada elemento do SUB, bem como a relação de itens a serem configurados relativamente ao sistema alvo. A Figura 4.12 resume as entradas e saídas esperadas para o passo.

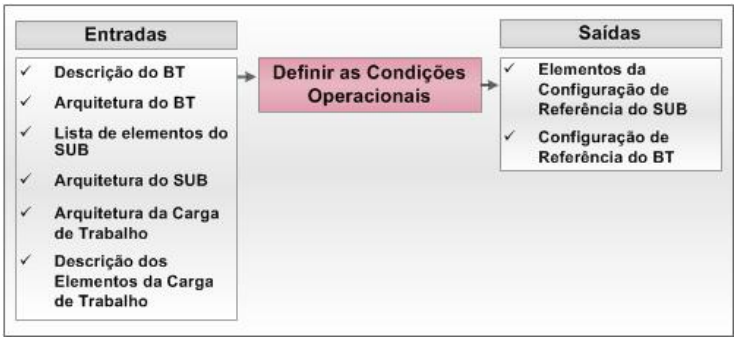
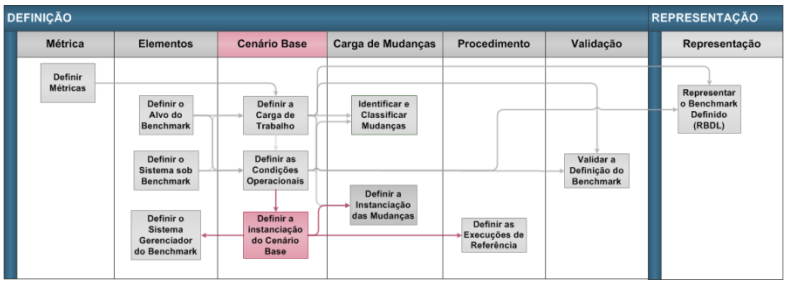


Figura 4.12 - Definir as Condições Operacionais - Entradas e Saídas.

4.2.3.3. Definir a Instanciação do Cenário Base



Neste passo devem ser especificados os métodos de geração da carga de trabalho de acordo com o modelo de carga definido anteriormente, bem como deve ser definida a forma como essa carga de trabalho será instanciada durante os experimentos. Este passo deve indicar, ainda, como serão geradas e implementadas as configurações do BT e do SUB.

Como saída esperada para este passo, devem-se obter as regras de geração e instanciação da carga de trabalho, os requisitos dos métodos de geração da carga de trabalho e das configurações das condições de operação de referência. A Figura 4.13 resume as entradas e saídas esperadas para o passo.

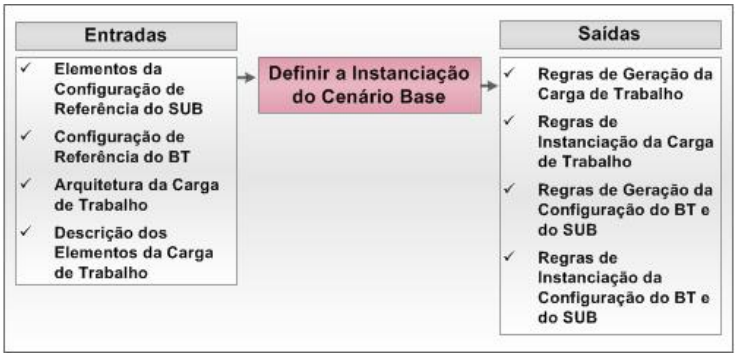


Figura 4.13 - Definir a Instanciação do Cenário Base - Entradas e Saídas.

4.2.4. Carga de Mudança

Nesta etapa é definida a carga de mudanças que é o componente mais complexo de um benchmark de resiliência, de um lado, por ser o componente adicional que o diferencia dos demais tipos de benchmark e, de outro, dada a dificuldade em se compreender a natureza, a complexidade e a imprevisibilidade das mudanças. Esta seção apresenta detalhadamente os passos necessários para definir a carga de mudanças. Estes passos estão descritos na Figura 4.14.

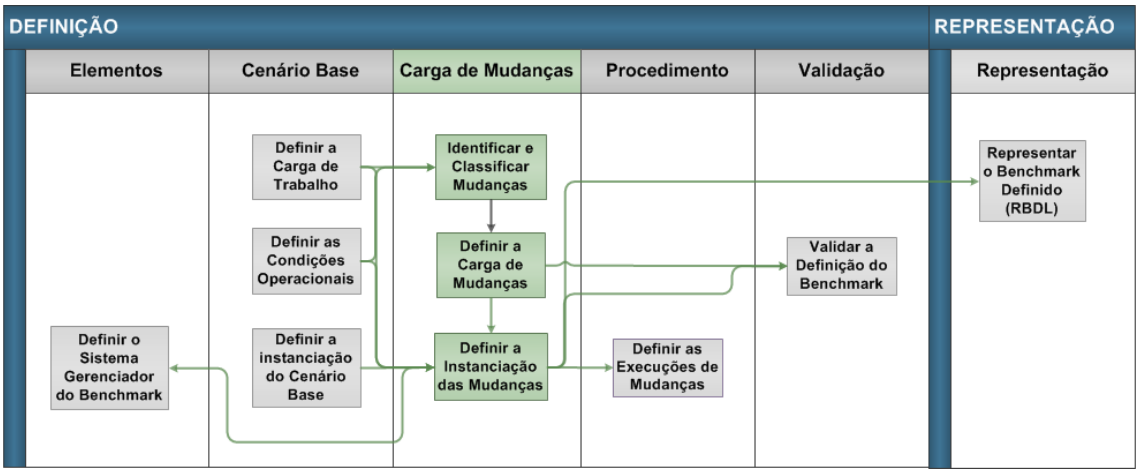
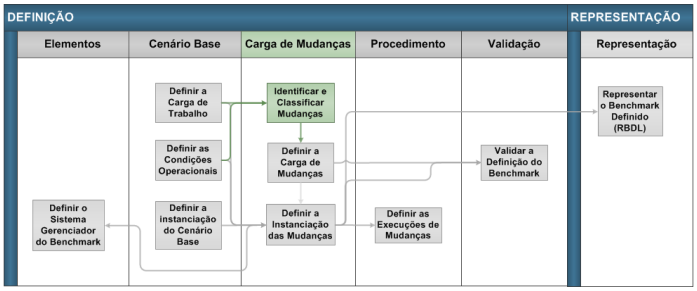


Figura 4.14 - Definição da Carga de Mudanças.

4.2.4.1. Identificar e Classificar Mudanças



Este passo compreende a definição do conjunto geral de mudanças às quais o sistema está potencialmente sujeito e pode ser subdividido nos seguintes subpassos: (i) definir as classes de mudanças a serem consideradas; (ii) identificar os elementos de mudança; e (iii) levantar o conjunto de mudanças.

Como saída deste passo espera-se obter a lista de classes e subclasses de mudanças, bem como uma lista abrangente das mudanças aplicáveis ao domínio em questão. Vale salientar que a lista de classes e subclasses apresentada nesta metodologia é genérica e pode ser utilizada em diferentes domínios. A Figura 4.15 resume as entradas e saídas esperadas para o passo.

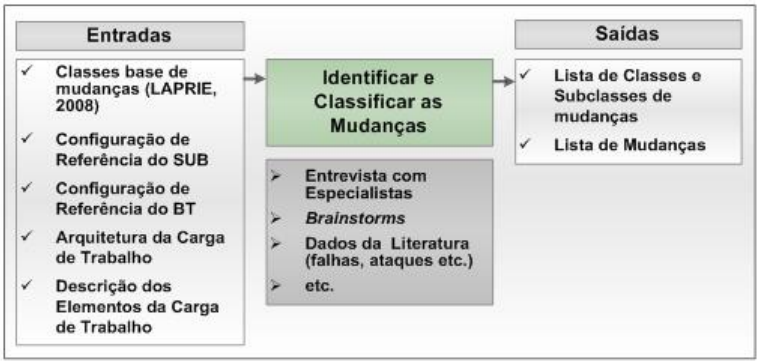


Figura 4.15 - Identificar e Classificar Mudanças - Entradas e Saídas.

1) Definir Classes de Mudanças

Segundo Laprie (2008), as mudanças nos sistemas computacionais podem ser consideradas sob três diferentes perspectivas: *natureza*, as mudanças podem ser funcionais, ambientais ou tecnológicas; *expectativa*, mudanças podem ser esperadas (p. ex. uma nova versão), previsíveis (p. ex. novas tecnologias ou plataformas) ou imprevisíveis (p. ex. mudanças nos requisitos, falhas ou novos tipos de ameaças); e *termo*, mudanças de curto prazo (de segundos a horas),

mudanças de médio prazo (de horas a meses) e mudanças de longo prazo (meses a anos).

As mudanças elencadas para um dado sistema poderiam ser classificadas por qualquer uma das três perspectivas definidas por Laprie. Neste trabalho nós utilizamos a perspectiva *natureza* para derivar as classes de mudanças, considerando principalmente os tipos e fontes de mudanças e usamos as perspectivas *expectativa* e *termo* no passo de avaliação da carga de mudanças que será apresentado na próxima seção. Assim, relativamente a sua natureza, as mudanças podem ser:

- ***Tecnológicas***: são as mudanças originadas por evoluções ou restrições tecnológicas que podem ter impactos positivos ou negativos no comportamento geral do sistema. São exemplos de mudanças tecnológicas uma nova versão de um sistema operacional, novas configurações de hardware, novas tecnologias de rede, etc.

- ***Requisitos***: são as mudanças relativas aos diferentes requisitos do sistema, ou seja, mudanças nos objetivos do sistema, na arquitetura, nos requisitos de projeto, etc.

- ***Ambientais***: são as mudanças originadas pelo ambiente, por exemplo, mudanças no padrão do uso de um sistema por seus usuários, falhas não esperadas em elementos que interagem com o sistema, mudanças na disponibilidade de recursos, ataques aos sistemas, etc.

Relativamente à natureza das mudanças, pode haver alguma sobreposição entre as classes, ou mesmo, alguma relação de causa e efeito entre elas. Por exemplo, requisitos funcionais podem condicionar uma mudança tecnológica: um requisito de tempo real pode exigir um sistema operacional de tempo real, ou a complexidade do sistema, em uma arquitetura distribuída, pode determinar o padrão de distribuição e o número de máquinas. A natureza das mudanças também pode ser mais ou ser menos dependente do domínio. Por exemplo, as mudanças tecnológicas são mais gerais e podem ser estendidas a

outros domínios, enquanto mudanças nos requisitos são dependentes do domínio.

A partir das classes de mudança estabelecidas (tecnológicas, requisitos e ambientais), devem-se definir subclasses que expressem com maior granularidade as mudanças a serem definidas. Por exemplo, mudanças tecnológicas podem estar relacionadas às versões dos elementos do SUB (p. ex. nova versão de um sistema operacional) ou às implementações (elemento implementado usando uma linguagem de programação diferente). Da mesma forma, em termos de mudanças ambientais, um sistema poderia ter o seu perfil de utilização modificado (p. ex. alterações no tráfego da rede) ou a sua configuração modificada. A Tabela 4.1 define as classes e subclasses de mudanças a serem consideradas na definição de uma carga de mudanças.

Tabela 4.1 - Classes de Mudanças.

Classes de Mudança	Subclasses	Descrição
Tecnológicas	Versões	Novas versões do software, bibliotecas, processadores.
	Implementação	Mudanças na implementação de um dos elementos: fornecedor, linguagem de programação, sistema operacional, etc.
	Classes	Classes de tecnologia: tempo real, uso de canal de comunicação alternativo, novas categorias de equipamento, etc.
	Recursos	Mudanças nos recursos oferecidos, normalmente associada a recursos físicos: novas máquinas, novo canal de comunicação, etc.
Requisitos	Objetivos	Mudanças nos requisitos funcionais do sistema.
	Escala	Mudanças em requisitos funcionais ou não funcionais associados à escala: número de modelos, número de máquinas, complexidade, etc.
	Projeto	Mudanças no projeto do sistema que se traduza em novas arquiteturas físicas ou lógicas.
Ambientais	Configuração	Mudanças associadas à configuração dos elementos do SUB ou BT.
	Perfil de uso	Mudanças no uso do sistema, p. ex., número de usuário, carga de uso, acessos, tráfego de rede, etc.
	Falhas Operacionais	Falhas causadas por intervenção humana (VIEIRA, 2005; DURÃES et al., 2008).
	Falhas de Software	Falhas originadas por erros de software (VIEIRA, 2005; DURÃES et al., 2008).
	Falhas de Hardware	Falhas originadas por erros hardware (VIEIRA, 2005; DURÃES et al., 2008).
	Ataques	Ataques intencionais ao sistema.

2) Identificar Elementos de Mudança

Outra perspectiva da mudança é a fonte da mesma, ou seja, qual elemento poderá ser modificado afetando o alvo do benchmarking (BT). Exemplos de elementos de mudança são o hardware onde o sistema é executado, o sistema operacional empregado, os padrões de distribuição do sistema, etc.

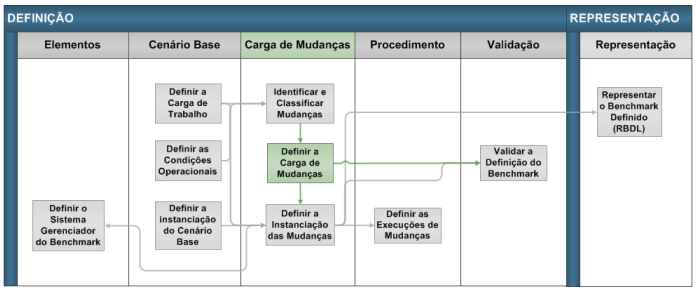
Neste subpasso devem-se identificar e elencar os potenciais elementos passíveis de mudança, já que os mesmos servirão como base, em conjunto com as classes, para a identificação do conjunto completo de mudanças. Em um benchmark de resiliência, todos os elementos que compõem o SUB e o *Cenário Base* estão sujeitos a mudanças sendo as potenciais fontes originadoras das mesmas.

3) Elencar Mudanças Potenciais

Neste subpasso deve-se realizar um levantamento exaustivo das mudanças que podem afetar o alvo do benchmark, tomando como base as classes de mudanças definidas e os elementos fontes de mudanças.

Entrevistas e *brainstorms* com especialistas da área são técnicas que podem ser utilizadas para prospectar o conjunto de mudanças do domínio. É importante que neste passo nenhuma mudança esperada ou expectável e que possa ser importante para a avaliação e comparação de sistemas ou para a determinação da resiliência dos mesmos seja esquecida ou negligenciada. Os especialistas devem analisar cada elemento identificado no passo anterior em relação a cada classe e subclasse de mudança, visando especificar as mudanças que poderiam afetá-los. Neste passo não deve ser efetuado nenhum tipo de seleção, buscando a elaboração de uma lista de mudanças que seja a mais abrangente possível, já que nos passos subsequentes essas mudanças devem ser normalizadas e avaliadas para efetivamente compor a carga de mudanças.

4.2.4.2. Compor a Carga de Mudanças



A carga de mudanças de um benchmark de resiliência será composta pelo conjunto de mudanças que posteriormente comporão os *Cenários de Mudança* na fase de instanciação do benchmark. Neste passo devem ser definidas as mudanças que farão parte da carga de mudanças por meio da execução dos seguintes subpassos: (i) normalizar as mudanças; e (ii) avaliar as mudanças.

Neste passo, as saídas esperadas são a lista normalizada de mudanças e a carga final de mudanças avaliada conforme ilustra a Figura 4.16.

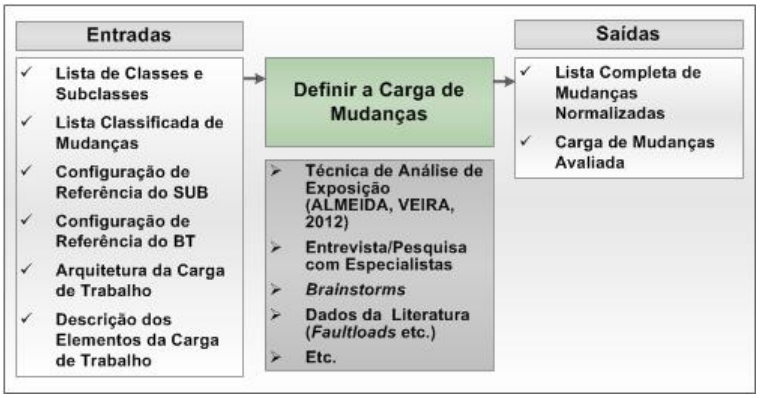


Figura 4.16 - Definir a Carga de Mudanças - Entradas e Saídas.

1) Normalizar as Mudanças

O objetivo do passo de *Identificação e Classificação das Mudanças* apresentado anteriormente é definir um conjunto detalhado de mudanças. No entanto, normalmente, como nem todas as mudanças identificadas têm a mesma granularidade, elas devem ser normalizadas com o objetivo de compor uma carga de mudanças mais sucinta e representativa, facilitando a sua implementação e instanciação durante os experimentos. A normalização das mudanças é feita por meio de um mapeamento para as mudanças finais, que pode ser um mapeamento um para um ou muitos para um.

Várias mudanças podem ser mapeadas para uma única mudança por meio de um parâmetro da mesma ou por meio da criação de um novo parâmetro para este fim. Por exemplo, as mudanças "*aumento no número de máquinas durante a simulação*" e "*diminuição no número de máquinas durante a simulação*" podem ser transformadas em uma única mudança usando o parâmetro "número de máquinas". A Figura 4.17 apresenta exemplos de mapeamento muitos para um utilizando parâmetros.

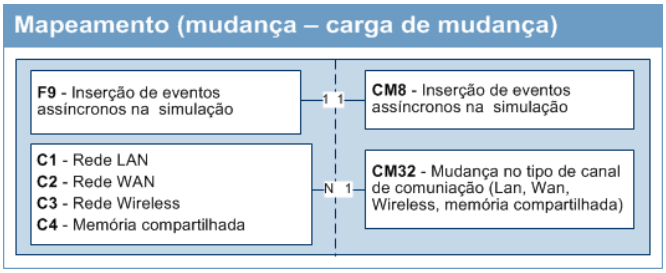


Figura 4.17 - Mapeamento de mudanças - Parâmetros.

O mapeamento também pode ser feito analisando o efeito das mudanças sobre o sistema, de forma análoga a abordagens da área de injeção de falhas nas quais, visando a avaliação de um sistema, muitas vezes é suficiente injetar somente os efeitos de uma falha (injeção de erros) (CHILLAREGE; BASSIN, 1998). Assim, uma única mudança que indica a indisponibilidade de uma máquina em uma simulação distribuída poderia, por exemplo, representar diferentes mudanças originais - o desligamento físico da máquina, reinicialização da máquina, etc. A Figura 4.18 mostra exemplos de mapeamento utilizando o efeito da mudança.

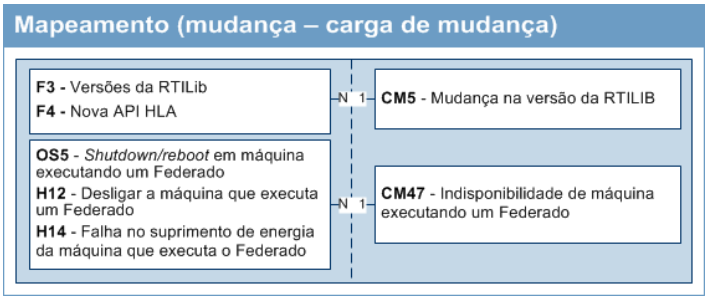


Figura 4.18 - Mapeamento de mudanças - Impacto.

2) Avaliar as mudanças

A carga de mudanças em um benchmark de resiliência deve garantir as propriedades de representatividade, portabilidade e simplicidade associadas a benchmarks. Entretanto, de um lado há um grande conjunto de mudanças possíveis dentro das diferentes classes estabelecidas e, de outro, há a incerteza relativa às mudanças, dificultando a definição de uma carga de mudanças que seja a um só tempo simples e representativa.

Diferentemente do que ocorre no âmbito de falhas, no qual dados históricos ou estatísticos podem ser utilizados para definir a representatividade de uma carga de falhas, no caso de mudanças, dados históricos de mudanças no perfil de uso de um sistema podem ser difíceis de ser encontrados, afora o fato de não necessariamente virem a representar o comportamento futuro. Ademais, as mudanças não estão restritas aos perfis de uso e às falhas, mas também se referem a novas tecnologias, arquiteturas e objetivos.

Seguindo a perspectiva *expectativa* proposta por Laprie (2008), na qual as mudanças podem ser esperadas, previsíveis ou imprevisíveis; observa-se uma semelhança com as expectativas de riscos em projetos. Assim, a análise da representatividade das mudanças em uma carga de mudanças pode ser realizada usando abordagens análogas às abordagens usadas em análises de riscos.

Neste trabalho nós seguimos a metodologia proposta em Almeida e Vieira (2012a) na qual a especificação de uma carga de mudanças representativa é feita por meio da análise da exposição do sistema ao conjunto de mudanças. A exposição do sistema a uma dada mudança é expressa pela probabilidade de dessa mudança ocorrer versus o impacto que a mesma teria sobre os requisitos de resiliência do alvo do benchmark.

Para essa análise, nós consideramos, dada a heterogeneidade da natureza das mudanças, também a perspectiva *termo* proposta por Laprie (2008) como mais um elemento para a definição da representatividade ou importância de

uma mudança. Por exemplo, uma nova versão de um sistema é uma mudança tecnológica esperada e tem probabilidade de média a alta de que venha a ocorrer, o impacto pode ser alto, entretanto, normalmente é um cenário de mudança que aparece no longo ou médio prazo, o que poderia torná-lo não menos representativo, mas menos importante quando comparado, por exemplo, a uma falha operacional que desestabilizasse um sistema em operação e afetasse a resiliência.

Visando avaliar a exposição do sistema a uma mudança, neste trabalho, adaptando Almeida e Vieira (2012a), nós consideramos:

- Probabilidade: define a probabilidade de ocorrência de uma mudança que afete o alvo do benchmark (BT). As escalas consideradas foram: *Muito Alta* quando a ocorrência da mudança é praticamente certa; *Alta,* quando a probabilidade de ocorrência da mudança é bastante grande; *Baixa,* para pequena probabilidade de ocorrência da mudança; e *Muito Baixa,* quando é muito pouco provável que a mudança venha a ocorrer.

- Impacto: define o impacto que uma determinada mudança teria nos principais atributos de dependabilidade e resiliência do sistema. As escalas consideradas para o impacto foram: *Catastrófico*, quando a mudança trouxer queda e indisponibilidade do sistema em questão; *Crítico*, quando for observado um impacto severo sobre os requisitos considerados; *Marginal*, quando houver um pequeno impacto nos requisitos avaliados; e *Negligenciável*, quando o impacto não trouxer prejuízos ao sistema.

- Prazo: define o período esperado de ocorrência de uma mudança e pode ser combinado com o impacto de sua ocorrência. As escalas utilizadas são: *Longo* (anos), *Médio* (dias a meses) e *Curto* (segundos a dias).

A exposição de um sistema a uma mudança é expressa, então, pela probabilidade da sua ocorrência versus o seu impacto e é modelada pelo aspecto prazo. O nível de exposição pode determinar a representatividade das mudanças em uma carga de mudanças e foi classificado como: *Mandatório,*

(mudanças que serão forçosamente incluídas na carga de mudanças*), Muito Alto, Alto, Médio, Baixo, Muito Baixo e Negligenciável* (mudanças que podem ser ignoradas).

Vale salientar que as escalas adotadas para *probabilidade* e *impacto* evitaram a tendência ao centro, são escalas pares, e estão baseadas nas escalas apresentadas no trabalho de referência, assim como a escala de *exposição* à mudança (ALMEIDA; VIEIRA, 2012a). Já a escala adotada para *prazo* segue definições do trabalho de Laprie (2008). A Tabela 4.2 apresenta a matriz de exposição.

Tabela 4.2 - Matriz de Exposição

Prazo / Impacto			Probabilidade			
			Muito Alta	Alta	Baixa	Muito Baixa
Prazo / Impacto	Curto	Catastrófico	Mandatório	Mandatório	Muito Alto	Alto
		Crítico	Mandatório	Muito Alto	Alto	Médio
		Marginal	Muito Alto	Alto	Médio	Baixo
		Negligenciável	Alto	Médio	Baixo	Muito Baixo
	Médio	Catastrófico	Mandatório	Muito Alto	Alto	Médio
		Crítico	Muito Alto	Alto	Médio	Baixo
		Marginal	Alto	Médio	Baixo	Muito Baixo
		Negligenciável	Médio	Baixo	Muito Baixo	Negligenciável
	Longo	Catastrófico	Muito Alto	Alto	Médio	Baixo
		Crítico	Alto	Médio	Baixo	Muito Baixo
		Marginal	Médio	Baixo	Muito Baixo	Negligenciável
		Negligenciável	Baixo	Muito Baixo	Negligenciável	Negligenciável

Propomos que para a definição da representatividade da carga de mudança deva ser definindo um nível de corte, por exemplo, a carga será composta por mudanças com nível de exposição igual ou acima de *Alto*.

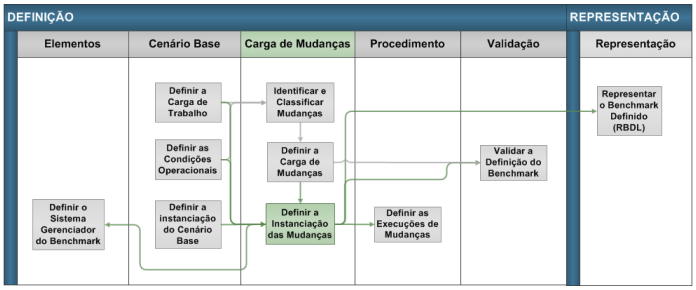
É importante frisar que é difícil atestar a representatividade das mudanças devido à sua complexidade e imprevisibilidade. A técnica proposta, análoga à análise de risco, é apenas uma das diferentes abordagens que podem ser adotadas com o objetivo de criar cargas de mudanças mais concisas e representativas.

Entretanto, devemos levar em consideração que um benchmark concreto é uma instância da definição do benchmark de resiliência e, assim, em uma dada instância pode-se ter interesse em avaliar apenas o impacto de um dado conjunto de mudanças ou de uma determinada classe ou subclasse de

mudanças. Além disso, fatores externos podem conduzir a composição dos cenários de mudança em uma determinada instanciação do benchmark, por exemplo, alterações na previsão do prazo de uma mudança, como no caso da perspectiva do uso de uma nova tecnologia. Para além disso, um benchmark de resiliência pode ser utilizado para verificar a continuidade da estabilidade de um sistema em face de uma mudança específica e já em andamento, por exemplo, atualização do sistema operacional.

Como estamos tratando de mudanças, esperadas, mas não completamente dominadas, ao longo do tempo a carga de mudanças e/ou a representatividade das mudanças dentro da mesma pode se alterar. Assim, é esperado que a definição do benchmark tenha a capacidade de acomodar essas alterações.

4.2.4.3. Definir a Instanciação das Mudanças



Neste passo devem ser definidos os atributos de definição das mudanças e os requisitos dos métodos de geração e instanciação das mudanças. Os métodos definidos devem ser implementados na ferramenta de benchmark.

Os atributos que definem uma mudança são:

a) *Fonte*: expressa a origem de um dado tipo de mudança. Por exemplo, o canal de comunicação é a origem da mudança “*alterações no perfil de uso da rede*”;

b) *Parâmetro*: define a geração e a aplicação de uma mudança. Por exemplo, na mudança "*aumento do número de modelos da simulação*", o parâmetro será o número de modelos. Os parâmetros podem ser categorizados em: *parâmetros de geração*, utilizados na fase de geração da mudança; *parâmetros de instanciação*, utilizados durante a instanciação dos cenários de mudança e parâmetros *de recuperação*. Vale salientar que na

instanciação os parâmetros podem assumir valores fixos ou podem obedecer a uma função de variação, por exemplo, o parâmetro que expressa o aumento na utilização de uma rede de dados pode ser modelado por uma função que modela a variação no tráfego da mesma ao longo do tempo. Neste passo é definida a lista de *parâmetros* da mudança que só terão os seus valores especificados na fase de instanciação do benchmark concreto;

c) *Métodos*: relativamente à implementação da carga de mudanças, três tipos de métodos podem estar associados a uma mudança: (i) *método de geração*, que define a forma como uma mudança será gerada (associado aos parâmetros de geração); (ii) *método de instanciação*, serviço pelo qual a mudança será instanciada durante o experimento de benchmark (associado aos parâmetros de instanciação); e (iii) *método de recuperação*, procedimentos de recuperação do sistema exigidos por alguns tipos de mudanças.

O resultado deste passo é a definição de uma carga de mudanças onde cada mudança está associada à sua *fonte*, aos seus parâmetros, bem como à lista de métodos de geração, instanciação e recuperação. Os requisitos funcionais dos métodos associados às mudanças também devem ser estabelecidos.

Vale salientar que técnicas de injeção de falhas, conforme apresentadas no Capítulo 3, podem ser utilizadas na definição de requisitos dos métodos de geração e aplicação de mudanças relacionadas a falhas de hardware, software e canal de comunicação.

A Figura 4.19 apresenta as entradas e as saídas esperadas neste passo.

Figura 4.19 - Definir a Instanciação da Carga de Mudanças - Entradas e Saídas.

4.2.5. Procedimento

Nesta etapa devem ser definidos os procedimentos e as regras de execução dos experimentos de benchmark. A execução de um benchmark de resiliência consiste em duas fases: a primeira, refere-se à execução de referência, onde as medidas obtidas serão utilizadas como base de comparação; enquanto a segunda, refere-se à execução do benchmark em presença dos cenários de mudança.

Importa salientar que nesta fase são definidas as regras de execução por meio da definição do projeto de execução e dos parâmetros de execução que nortearão os experimentos, no entanto, esses parâmetros de execução só se materialização por meio de valores durante a instanciação dos benchmarks concretos.

A Figura 4.20 representa as etapas desta fase.

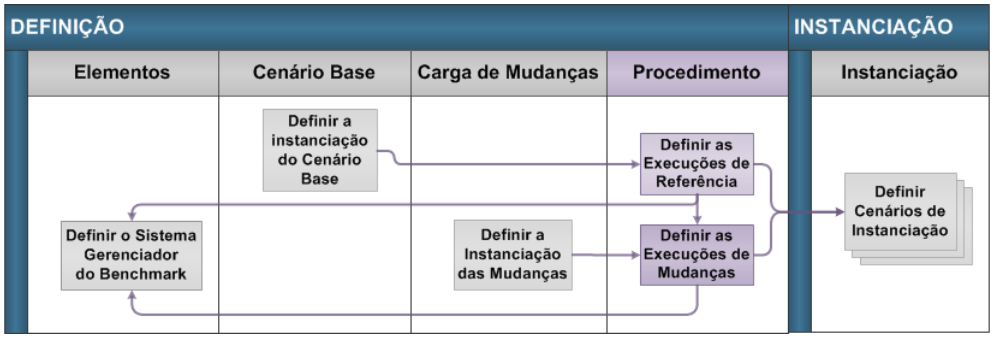
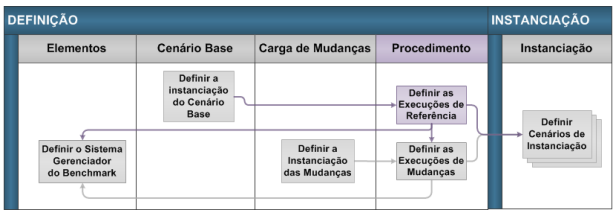


Figura 4.20 - Definição dos Procedimentos e Regras.

4.2.5.1. Definir a Execução de Referência



Uma vez definidas as cargas de trabalho, seus métodos de instanciação e as condições operacionais, deve-se definir o projeto de execução do benchmark. Neste projeto define-se como os experimentos de referência serão executados.

Assim, devem ser definidos os parâmetros que caracterizam a execução do experimento, em que ponto de tempo serão coletados os resultados, como os resultados serão armazenados, etc. Os parâmetros de execução são definidos neste passo e assumirão valores específicos nos benchmarks concretos.

Vale salientar que nesta etapa é feito o projeto geral de execução do benchmark, no entanto, benchmarks concretos podem determinar pequenas variações relativamente ao projeto geral.

Neste passo, além do projeto da execução do experimento de referência do benchmark, definem-se também o conjunto de parâmetros e as regras de aplicação do mesmo. A Figura 4.21 apresenta as entradas e saídas do passo.

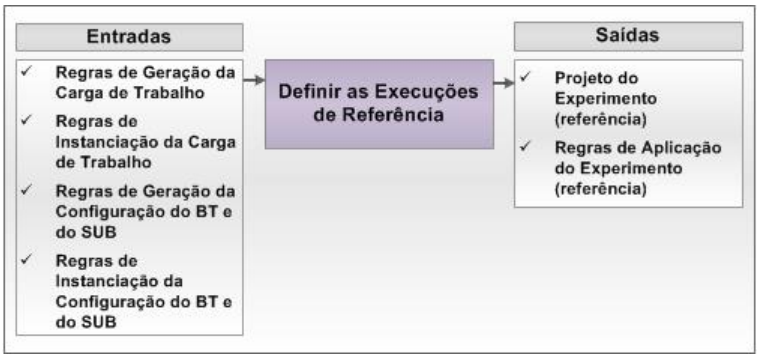
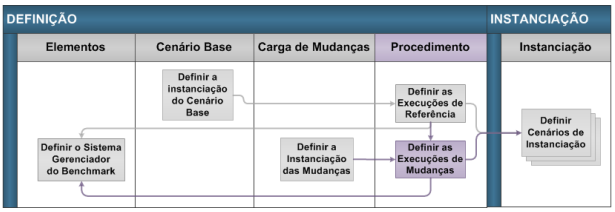


Figura 4.21 - Definir as Execuções de Referência - Entradas e Saídas.

4.2.5.2. Definir as Execuções de Mudança



O projeto de execução com mudanças é, normalmente, representado pelo projeto de execução de referência definido no passo anterior associado às regras de aplicação das mudanças definidas por meio de seus atributos de aplicação. Os atributos relacionados à aplicação das mudanças durante a execução do experimento são:

a) *Gatilho*: determina a ação ou fator que irá iniciar a aplicação de uma mudança durante o experimento do benchmark. O gatilho pode ser orientado ao tempo, por exemplo, a mudança é aplicada a cada 3 minutos,

pode ser orientado a estado, por exemplo, uma mudança é aplicada após a estabilização do sistema, etc.;

b) *Duração*: o tempo pelo qual uma mudança permanece ativa no sistema. Uma mudança pode estar ativa durante uma rodada inteira de execução do benchmark, ou ficar ativa apenas durante um determinado período de tempo;

c) *Cardinalidade*: número de repetições da aplicação da mudança dentro de uma rodada de execução do benchmark;

d) *Tempo de recuperação*: tempo necessário para que o sistema se recupere da aplicação de uma dada mudança. Mudanças que podem tornar o sistema instável ou indisponível, como mudanças que estressem o sistema ou falhas, podem exigir um período para reestabilização do sistema.

Esses atributos representam as regras de aplicação gerais de uma mudança e devem assumir valores específicos durante a instanciação de um dado benchmark concreto.

Caso haja alguma divergência entre as execuções de referência e de mudança, por exemplo, algum parâmetro adicional, pontos específicos de coleta de resultados, etc., essa especificidade deve estar refletida no projeto de execução de mudanças, caso contrário, este passo é opcional.

Espera-se como resultado deste passo a definição do projeto de execução com mudanças e parâmetros adicionais de execução quando houver. A Figura 4.22 apresenta as entradas e saídas do passo.

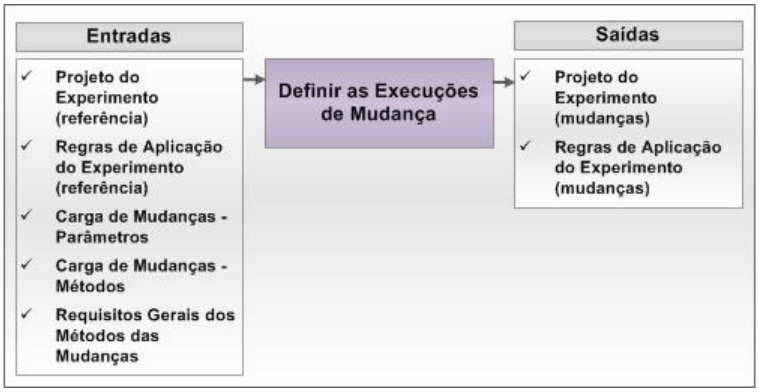


Figura 4.22 - Definir as Execuções de Mudanças - Entradas e Saídas.

4.2.6. Validação

A característica que distingue um benchmark de outras ferramentas de comparação e avaliação *ad hoc* é a credibilidade do mesmo junto à indústria ou à comunidade de usuários. Assim, para que um benchmark seja útil e aceito como uma ferramenta padronizada, ele deverá atender a um conjunto de propriedades pré-estabelecidas.

A metodologia proposta adota a maioria das propriedades usadas em benchmarks de dependabilidade, conforme apresentando no Capítulo 3. Assim, um benchmark de resiliência deve ser *relevante*, *representativo*, *repetível*, *não-intrusivo, portável, escalável* e *simples*.

Nesta etapa, deve-se atestar que os elementos do benchmark definidos nas etapas anteriores atendem a estas propriedades e são válidos. A Figura 4.23 apresenta os passos desta etapa.

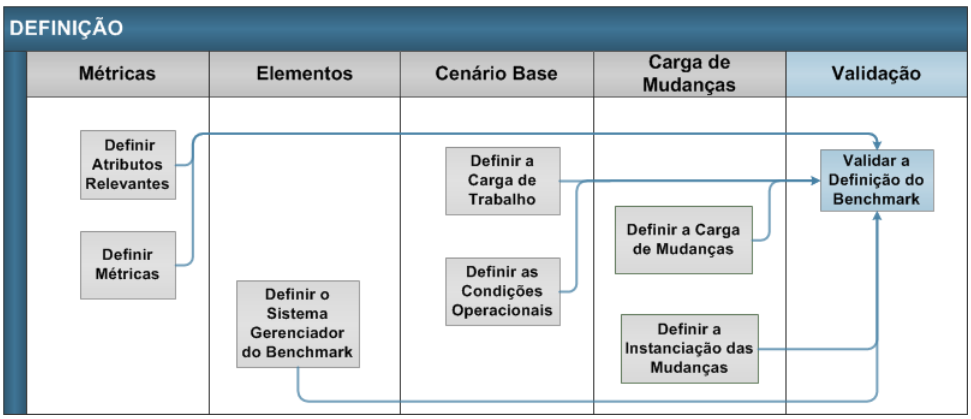


Figura 4.23 - Validação do Benchmark.

Neste trabalho nós consideramos os elementos definidos para um benchmark de resiliência como um superconjunto de diferentes benchmarks concretos que podem ser derivados e instanciados. Uma parte da validação está diretamente relacionada à definição do benchmark, ou seja, garantindo que os elementos definidos são válidos, garante-se automaticamente que os benchmarks derivados dos mesmos também o são.

Nos itens seguintes serão apresentados os aspectos de validação no que tange aos elementos especificados na fase de definição.

a) *Relevância:* para que um benchmark seja relevante é preciso que tenham sido avaliados os atributos mais importantes do domínio específico. A avaliação de um atributo pouco importante pode fazer com que um benchmark tenha pouca utilidade como ferramenta de suporte à escolha de produtos, já que o atendimento do atributo por um dado produto não lhe daria vantagens reais. Portanto, a escolha dos atributos a serem avaliados determina a relevância do benchmark e estes são os elementos a serem validados relativamente a essa propriedade.

b) *Representatividade:* a definição de elementos que não representem adequadamente o uso de um sistema pode levar um benchmark a resultados que atestem que um produto apresenta um determinado nível de atendimento em relação a um atributo específico, ou que um produto é melhor que outro, fatos que podem não se confirmar durante a utilização do sistema. Por exemplo, se em um benchmark de desempenho no qual se está a avaliar o tempo de resposta, o produto é submetido a uma carga de trabalho que fica aquém da carga real, vários produtos poderiam apresentar desempenho semelhante e que não seria mantido quando uma carga real e maior fosse aplicada, isso invalidaria a comparação. Por outro lado, se uma carga de trabalho excessiva e não realista fosse aplicada a vários produtos, a avaliação poderia resultar em casos nos quais poucos produtos ou nenhum produto conseguiria atender ao atributo de forma satisfatória, isso poderia levar ao descarte de produtos viáveis. A representatividade de um benchmark está intimamente relacionada aos seus principais elementos. No

caso de benchmark de resiliência, não apenas a carga de trabalho deve ser representativa, mas também a carga de mudanças deve representar falhas e mudanças reais às quais o sistema estaria sujeito, e as métricas escolhidas devem representar adequadamente os atributos a serem medidos.

c) *Simplicidade*: relativamente a esta propriedade, o benchmark de resiliência deve ser fácil de ser implementado, utilizado e aplicado, e deve ser economicamente viável. As primeiras características estão diretamente ligadas à especificação do benchmark e dos requisitos de implementação das ferramentas. Assim, a previsão de automação das tarefas de geração de cargas de trabalho e mudanças, bem como a execução automática do experimento de benchmark devem garantir a simplicidade do seu uso.

Em resumo, o passo de validação da definição do benchmark deve ter como saída a demonstração de que os elementos definidos atendem às propriedades elencadas. A Figura 4.24 resume esse passo mostrando as entradas requeridas e a saída esperada.

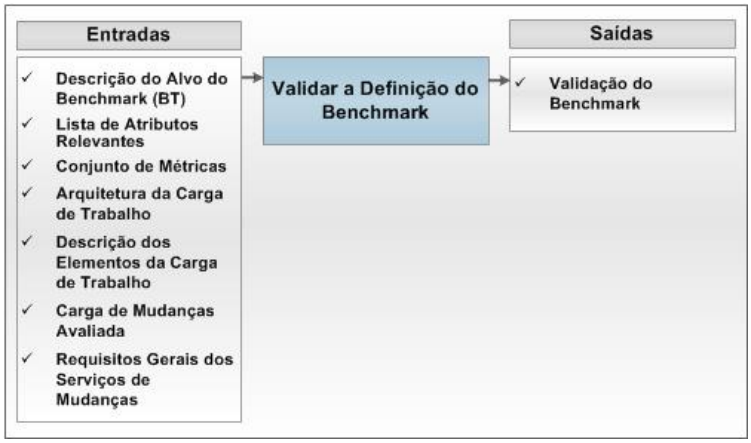


Figura 4.24 - Validar a Definição do Benchmark - Entradas e Saídas.

4.3. Benchmark de Resiliência: Instanciação

Nesta etapa devem ser instanciados e validados os benchmarks concretos. Um benchmark de resiliência concreto é uma instância do processo de definição e

seus elementos representam um subconjunto dos elementos previamente definidos.

Os benchmarks concretos são instanciados por meio de *Cenários de Instanciação* nos quais são definidos os atributos e métricas a serem avaliadas, o *Cenário Base* a ser utilizado, o conjunto de mudanças a serem aplicadas e os parâmetros de execução do benchmark. Um experimento de benchmark será sempre realizado a partir de um benchmark concreto, ou seja, a partir de um *Cenário de Instanciação*.

A Figura 4.25 mostra o fluxo de trabalho para a instanciação de um benchmark e execução de experimentos de benchmark.

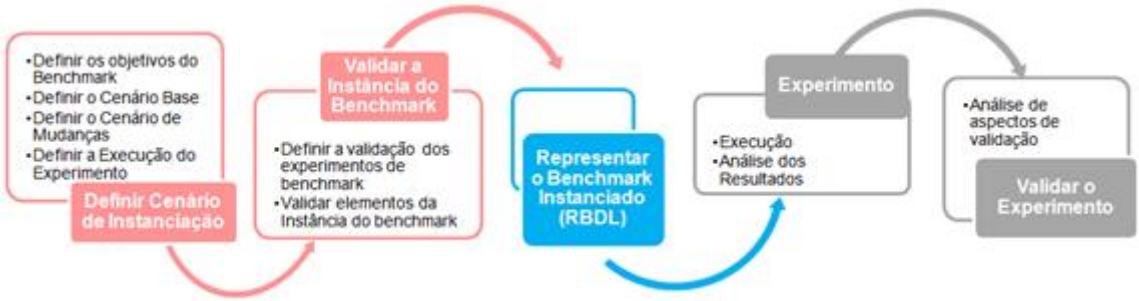


Figura 4.25 - Fluxo de trabalho - Instanciação dos benchmarks concretos e execução dos experimentos de benchmark.

Em um benchmark de resiliência concreto, representado por um *Cenário de Instanciação*, o conjunto de mudanças a serem aplicadas compõe um *Cenário de Mudança*.

Vale salientar que mudanças em um sistema nem sempre ocorrem de forma isolada, sendo esperado que ocorram de forma simultânea ou sequencial. Por exemplo, um sistema pode ter um contexto no qual se diminui a capacidade do processador da máquina onde o BT está sendo executado e avalia-se o efeito dessa mudança na resiliência do mesmo, e outro no qual se diminui a capacidade de memória da máquina e, da mesma forma, avalia-se o efeito da mudança. É importante avaliar os cenários de forma isolada, já que tal avaliação permitiria verificar, por exemplo, qual restrição afetou de maneira mais significativa a resiliência do sistema; o que possibilitaria configurar melhor

a máquina de execução ou escolher um sistema mais resiliente a um ou outro fator. No entanto, seria razoável imaginar que uma nova máquina tivesse maior capacidade de memória e processamento simultaneamente, ou que poderia haver a contingência representada pela obrigatoriedade do uso de uma máquina com menor capacidade, que tivesse esses dois elementos diminuídos ao mesmo tempo. A concomitância nas mudanças é ainda mais crítica para as classes de mudanças de ambiente, tais como variações no perfil de uso do sistema, falhas ou ataques.

Assim, no que se refere à aplicação das mudanças, a metodologia considera dois tipos de *Cenários de Mudança*: (i) *Cenários Primários*, formados por mudanças independentes umas das outras; (ii) *Cenários Compostos*, formados pela aplicação de duas ou mais mudanças de forma associada. Os Cenários Compostos podem ser: *aleatórios*, no qual as mudanças são aplicadas alternadamente e de forma aleatória, por exemplo, um conjunto de falhas; *sequenciais*, no qual as mudanças são aplicadas em uma sequência conhecida e pré-determinada; e em *cascata*, no qual a aplicação de uma mudança guia a aplicação das demais, por exemplo, um cenário composto pelo cenários de mudanças "*mudança no número de máquinas do ambiente distribuído*" e "*mudança no número de modelos em uma simulação*", neste caso, cada variação no número de máquinas exercitaria cada possível variação no número de modelos.

A Figura 4.26 apresenta os passos desta etapa.

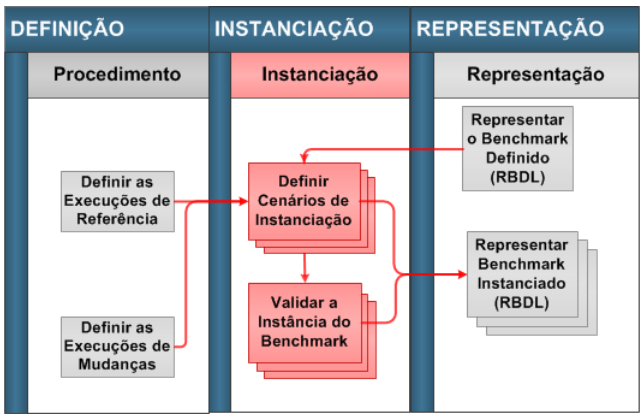
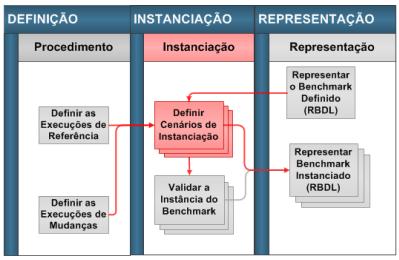


Figura 4.26 - Instanciação do Benchmark.

4.3.1. Definir os Cenários de Instanciação

Neste passo, de acordo com o objetivo específico do benchmark concreto, devem ser escolhidos, dentre os elementos previamente definidos na fase de definição do benchmark, os elementos que comporão os Cenários de Instanciação.



Subpassos que devem ser seguidos para a definição de um cenário de instanciação:

1) ***Definir o objetivo do benchmark:*** neste subpasso, a partir do conjunto de atributos e métricas previamente definidos, são selecionados os atributos e as métricas a serem avaliadas de acordo com o objetivo do benchmark em questão. Em benchmarks de resiliência, esse passo compreende também a escolha das métricas específicas de resiliência a serem utilizadas.

2) **Definir o *Cenário Base:*** neste subpasso são selecionadas a carga de trabalho e as condições operacionais a serem utilizadas na instância do benchmark. Este passo compreende a escolha de uma das cargas de trabalho especificadas anteriormente, bem como uma das configurações do sistema e do SUB definidas.

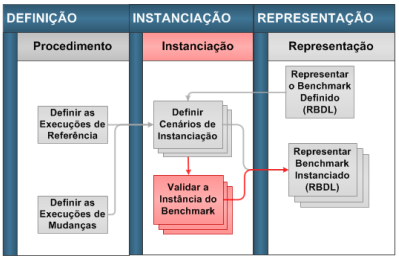
3) **Definir o *Cenário de Mudanças:*** neste subpasso, para benchmarks de resiliência, são escolhidos os tipos de mudanças que comporão o *Cenário de Mudanças* naquela instância do benchmark e de que forma o conjunto de mudanças será instanciado, ou seja, deve ser indicado o tipo do cenário: primário, aleatório, sequencial ou em cascata. Devem ser definidos, também, valores para os atributos de aplicação de cada mudança do cenário: parâmetros, cardinalidade, duração, gatilho e o tempo de recuperação, quando pertinente.

4) ***Definir a Execução***: neste subpasso devem-se definir valores para os parâmetros de execução previamente especificados na fase de definição de regras e procedimentos.

Espera-se como resultado deste passo obter uma lista dos elementos escolhidos a partir do conjunto de elementos previamente definidos (atributos, métricas, mudanças, etc.), bem como a definição de todos os valores de atributos e parâmetros que serão usados no benchmark concreto. A Figura 4.27 representa esse passo.

Figura 4.27 - Definir Cenários de Instanciação - Entradas e Saídas.

4.3.2. Validar a Instância do Benchmark



Segundo a metodologia, alguns aspectos da validação dos benchmarks de resiliência já estão cobertos por meio da validação da definição de seus elementos. Isso satisfaz, por exemplo, a validação da relevância do benchmark. Entretanto, algumas propriedades só podem ser validadas ou atestadas a partir da materialização de um dado benchmark.

As propriedades requeridas para validação de uma instância benchmark são:

a) *Portabilidade/Escalabilidade*: um benchmark é mais útil quando se pode portá-lo para diferentes sistemas e arquiteturas e aplicá-lo considerando diferentes escalas, já que um benchmark estanque para um sistema específico terá sua utilidade bastante reduzida. Em um benchmark de resiliência, a portabilidade e a escalabilidade estão incorporadas à carga de mudanças por meio da classe de mudanças *Tecnológicas* e por meio da classe de mudanças *Escala*. Assim, usando cenários de mudanças

compostos será possível definir que um dado experimento será portável para outros sistemas e escalável. Por exemplo, a combinação da mudança "*diferente sistemas operacionais*" com as demais mudanças já garante e, também mede, a portabilidade do benchmark. Da mesma forma, mudanças previstas da classe *Escala*, já podem incorporar regras de escalabilidade tanto da carga de trabalho, quanto das condições operacionais, por meio dos valores de seus parâmetros. Em um benchmark concreto, devem ser estabelecidos os critérios de composição de cenários visando atestar as propriedades de portabilidade e escalabilidade.

b) *Representatividade*: está associada à definição da carga de trabalho, da carga de mudanças e das métricas e já é parcialmente validada na definição do benchmark. Entretanto, para o caso da carga de mudanças, não basta que a mudança em si seja representativa, é preciso que os valores dos parâmetros da mudança sejam também válidos, bem como os valores de seus demais atributos. Além disso, a composição das mudanças em cenários de mudança compostos também deve ser representativa. Como esses aspectos só se materializam durante a instanciação do benchmark, essa validação deverá ser apresentada para cada benchmark concreto.

c) *Repetibilidade*: um benchmark deve poder ser repetido no mesmo contexto de aplicação apresentando resultados estatísticos semelhantes. A repetibilidade é um termo estatístico que descreve a estabilidade ou o grau de concordância entre os resultados de medições sucessivas do mesmo atributo quando efetuadas sob as mesmas condições de medição, sendo qualquer desvio atribuído ao instrumento medidor (ANDRIOTTI, 2005). O desvio padrão dos resultados do teste obtidos em condições de repetibilidade é chamado de *desvio padrão de repetibilidade* e é uma medida de dispersão da distribuição dos resultados de teste. Nesta tese nós utilizamos o *desvio padrão de repetibilidade relativo*, aqui chamado de *índice de repetibilidade*, para obter o percentual de variação entre os diferentes resultados replicados. O número vezes que o experimento deverá

ser reaplicado, bem como o máximo índice de repetibilidade aceito em um experimento devem ser especificados no benchmark concreto. Já o relatório de resultados deve atestar o índice de repetibilidade efetivamente obtido nos experimentos.

d) *Simplicidade de Uso*: o dimensionamento dos *cenários de instanciação* e das regras de aplicação do benchmark deve garantir a suficiência do experimento em termos estatísticos sem afetar a sua viabilidade. Uma instância concreta do benchmark de resiliência deve demonstrar a propriedade simplicidade especificando a expectativa de duração do experimento e atestando a facilidade de aplicação do mesmo. O relatório de resultados deve atestar tanto o período em que o benchmark foi aplicado, quanto o tempo efetivo de aplicação.

e) *Não-intrusividade*: relativamente a uma instância de benchmark, a não-intrusividade está relacionada à implementação das cargas de mudança e configurações do sistema. A não-intrusividade deve ser atestada pela instância do benchmark concreto.

O passo de *validação da instância do benchmark* se divide nos seguintes subpassos:

1) Definir a validação dos experimentos de benchmark

Neste subpasso são definidos os aspectos de validação que devem ser atestados pelos relatórios de execução dos experimentos. As principais propriedades a serem atestas são: (i) simplicidade, no que tange à duração do experimento; (ii) repetibilidade, aqui devem-se definir o número mínimo de vezes que o experimento deve ser reaplicado e o índice aceitável de repetibilidade; e (iii) não-intrusividade, aqui, caso seja possível, deve-se indicar a forma como a não-intrusividade deve ser atestada em um experimento.

2) Validar a Instância de Benchmark

Neste subpasso os elementos da instância do benchmark devem ser validados, relativamente às propriedades listadas: não intrusão, simplicidade de execução, portabilidade/escalabilidade, representatividade. Além disso, sempre que pertinente, devem-se validar os resultados dos experimentos do benchmark em relação aos valores definidos previamente (repetibilidade e duração do experimento).

Espera-se obter neste passo a lista dos elementos validados, bem como uma lista de elementos que devem ser validados por meio do relatório de resultados do experimento de benchmark. A Figura 4.28 resume o passo.

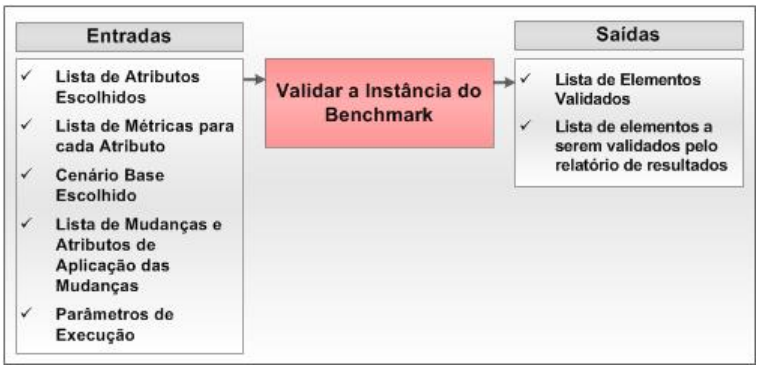


Figura 4.28 - Definir a Validação da Instância do Benchmark - Entradas e Saídas.

4.4. Benchmarks de Resiliência: Representação

De acordo com a metodologia, os elementos definidos para o benchmark de resiliência, bem como os benchmarks concretos derivados da definição devem ser representados usando a sintaxe e semântica estabelecida pela Linguagem de Descrição de Benchmark de Resiliência (RBDL) apresentada no Capítulo 6. A Figura 4.29 apresenta os passos desta fase. Vale salientar que a RBDL é uma alternativa para representação e disseminação de benchmarks, mas outras abordagens podem ser utilizadas na execução dos passos desta etapa.

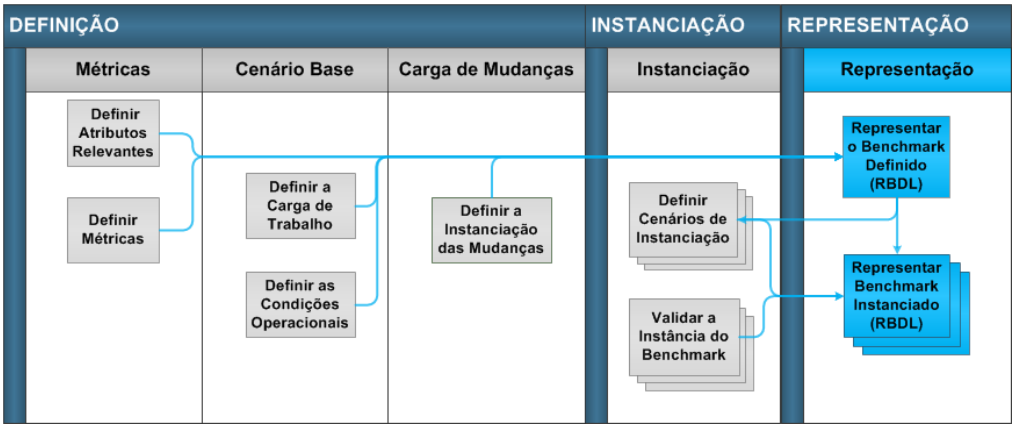
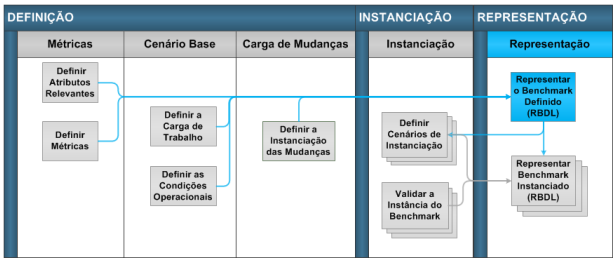


Figura 4.29 - Representação da Definição do Benchmark.

4.4.1. Representar a Definição do Benchmark



Os elementos especificados durante a fase de definição do benchmark de resiliência devem ser descritos em um formato padronizado e a linguagem de descrição do benchmark RBDL apresenta seções específicas que possibilitam a representação do conjunto de elementos definidos pela metodologia. Esses elementos serão representados uma única vez, em um arquivo de definição, mas poderão ser utilizados e instanciados em diferentes benchmarks. Embora essa seja a parte fixa definida pela metodologia, ela pode evoluir e vir a incorporar novos atributos, métricas, cargas de trabalho e mudanças.

Espera-se como resultado deste passo obter um arquivo de representação de benchmark (RBDL) no qual os elementos definidos nas etapas anteriores estejam representados e possam ser referenciados na instanciação de benchmarks concretos. A Figura 4.30 resume o passo.

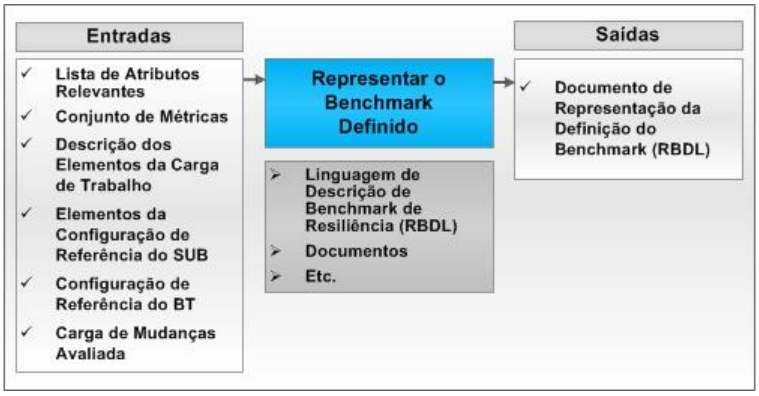
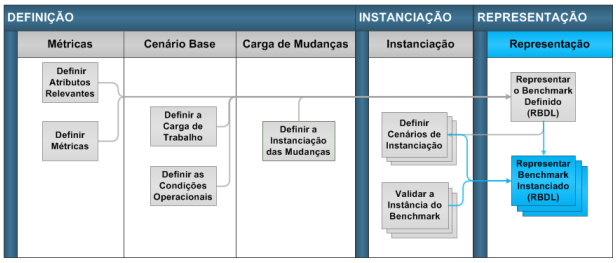


Figura 4.30 - Representar o Benchmark Definido - Entradas e Saídas.

4.4.2. Representar o Benchmark Instanciado



Neste passo deve ser representado o benchmark concreto. Desta forma, cada elemento especificado durante a instanciação do benchmark deve ser mapeado nas seções correspondentes da RBDL visando definir o arquivo de representação do Benchmark.

O objetivo deste passo é gerar o documento que representa uma instância de benchmark e que será utilizado para guiar a sua implementação e execução. O documento resultante deste mapeamento deve incorporar ou referenciar a definição do benchmark e deve ser representado usando a linguagem RBDL.

A Figura 4.31 representa as entradas e saídas do passo.

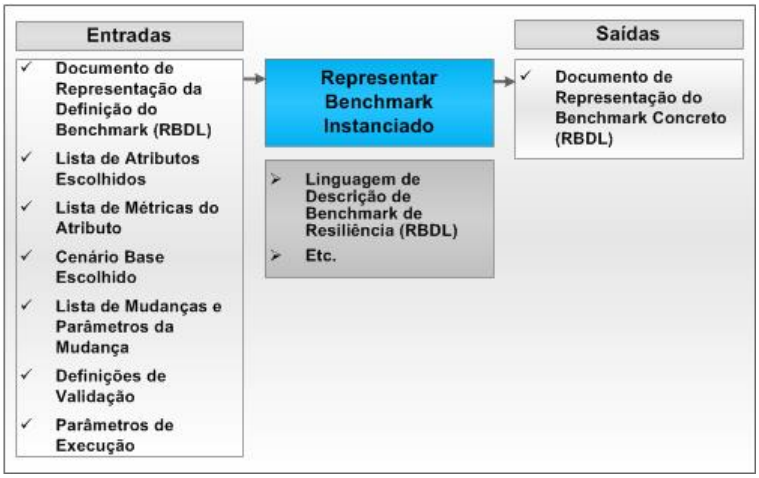


Figura 4.31 - Representar o Benchmark Instanciado - Entradas e Saídas.

4.5. Considerações Finais

Este capítulo apresentou uma metodologia para a definição, instanciação e representação de benchmark de resiliência. A definição da metodologia se baseou no conjunto de trabalhos de especificação de benchmarks de desempenho e dependabilidade apresentados no capítulo anterior. Por meio da metodologia, buscou-se, de um lado, sistematizar o processo de especificação de benchmarks e, de outro, estendê-lo para incorporar aspectos específicos de um benchmark de resiliência.

Embora a metodologia tenha sido definida com o objetivo de sistematizar a especificação de abordagens de benchmarking de resiliência para o domínio de infraestruturas de simulação baseadas em HLA, as diretrizes aqui apresentadas são gerais e permitem o seu uso ou adaptação a outros domínios.

5 BENCHMARKING DE RESILIÊNCIA PARA INFRAESTRUTURAS DE SIMULADORES DE SATÉLITES BASEADAS EM HLA

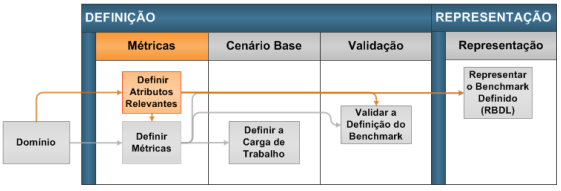
Este capítulo apresenta a nossa abordagem para a realização dos passos recomendados pela metodologia para a definição de um benchmark de resiliência para infraestruturas de simuladores de satélites baseadas em HLA. Cada seção apresenta as etapas e passos executados e exemplos dos elementos resultantes de cada um deles.

Há uma correspondência direta entre as seções deste capítulo e as seções do Capítulo 4 relacionadas à fase de definição de abordagens de benchmarking. Além disso, este capítulo é complementado pelo Apêndice B que apresenta o conjunto mais completo de elementos de benchmark especificados para o domínio de infraestruturas HLA.

5.1. Métricas no Contexto de Simuladores de Satélite

As subseções a seguir mostram a execução dos passos para a fase de definição de métricas e apresentam os atributos e métricas especificados para o domínio de infraestruturas de simulação.

5.1.1. Atributos Relevantes



O uso de simuladores nas diferentes fases de uma missão espacial traz variações nos seus requisitos de dependabilidade. Por exemplo, simuladores usados na análise da missão requerem a correção dos resultados fornecidos pelos modelos, uma vez que decisões acerca da missão serão tomadas a partir dos mesmos. Simuladores para verificação e validação, quando utilizam hardware na malha de simulação, têm requisitos temporais de tempo real restrito e, neste caso, a estabilidade e latência são também importantes. Já simuladores operacionais, utilizados no diagnóstico de anomalias no satélite ou na validação de procedimentos operacionais críticos, exigem modelos com grande fidelidade e confiabilidade e apresentam, ainda,

altos requisitos de disponibilidade, uma vez que os procedimentos operacionais devem ser realizados em momentos específicos no tempo. Ao avaliar infraestruturas de simulação que serão usadas por diferentes classes de simuladores deve-se ter em mente esses diferentes requisitos.

No levantamento dos atributos mais relevantes, em um primeiro momento, foi elencado um conjunto abrangente de atributos de dependabilidade relacionados ao domínio de simuladores. Além dos atributos apresentados no Capítulo 3, foram considerados, por sua relevância, dois atributos listados por Firesmith (2003) e Boehm et al. (2004): (i) o desempenho e seus subatributos, correlacionados principalmente aos requisitos de tempo de um simulador de satélite; e (ii) a correção e seus subatributos, relacionados aos resultados dos modelos que utilizam a infraestrutura de simulação.

A Tabela 5.1 apresenta o conjunto de atributos usados neste trabalho para a definição dos atributos mais relevantes no domínio de infraestruturas HLA. Para a composição dessa tabela foram utilizados os seguintes critérios de escolha: (i) atributos mais comumente aceitos no âmbito da dependabilidade; (ii) atributos relacionados a aspectos de uso do sistema, já que é esse é o foco de um benchmark; e (iii) atributos de dependabilidade relacionados ao domínio do problema (p. ex. desempenho, acurácia). A definição usada no contexto deste trabalho para os atributos apresentados encontra-se no Glossário.

Tabela 5.1 - Atributos e Subatributos de Dependabilidade.

Atributos			
Confiabilidade			
Interoperabilidade			
Disponibilidade			
Segurança (*Safety)*			
Robustez			
Segurança (*Security*)			
	Confidencialidade	Integridade	
Correção			
	Acurácia	Estabilidade	*Currency*
Desempenho			
	Jitter	Latência	Agendabilidade
	Rendimento	Tempo de Resposta	

Uma vez definido o conjunto geral de atributos a ser analisado, passou-se a escolha dos atributos mais relevantes. Neste trabalho, nós adotamos duas abordagens para a escolha dos atributos mais relevantes para o domínio de simuladores de satélite: pesquisa com especialistas e análise dos requisitos funcionais das infraestruturas de simulação.

Na primeira abordagem, a pesquisa com especialistas consistiu em apresentar a cada participante uma *Matriz de Avaliação* composta pelos atributos de dependabilidade elencados (Tabela 5.1) e pelas classes e tipos de simuladores previstos no memorando técnico ECSS (2010) e detalhados no Capítulo 2. Na avaliação, para cada célula da matriz, o especialista atribuiu uma nota relativa ao grau de exigência daquele atributo para aquele tipo ou classe de simulador. Assim, foi avaliado o quão importante era o atendimento daquele atributo para o simulador em questão. Relativamente à nota, foi utilizada uma escala de quatro níveis, balanceada em termos de nível de exigência, sem nível intermediário, visando dirimir qualquer efeito que pudesse advir de uma tendência ao centro. Foi utilizada a seguinte escala: 1 (muito baixo), 2 (baixo), 3 (alto) e 4 (muito alto). Obtivemos como resultado dessa pesquisa, um índice do grau de exigência para cada atributo relativamente às classes gerais de simuladores que foi calculado usando a média das notas atribuídas pelos especialistas. O Apêndice B apresenta detalhes dessa pesquisa e os resultados obtidos.

Na segunda abordagem, foi realizada a análise dos requisitos funcionais de infraestruturas de simulador definidos pelo memorando técnico ECSS (2010) em relação aos atributos de dependabilidade exigidos pelas diferentes classes de simuladores de satélite. Para essa análise, nós seguimos o trabalho de Mostert e Von Solms (1995) que apresenta uma metodologia na qual os requisitos não funcionais são elicitados a partir de requisitos funcionais por meio da definição de contramedidas; e que define uma matriz tridimensional na qual a primeira dimensão representa os requisitos funcionais, a segunda os requisitos não funcionais analisados (atributos) e a terceira os *stakeholders* e tecnologias. No nosso trabalho, a abordagem foi utilizada para avaliar o

possível impacto dos atributos de dependabilidade nos requisitos funcionais da infraestrutura de simulação, segundo a opinião de dois *stakeholders*. Essa análise foi realizada tanto para os requisitos gerais de uma infraestrutura de simulação, quanto para os requisitos específicos da infraestrutura de simulação que são cobertos por serviços do padrão HLA. Cada requisito funcional foi avaliado em relação aos atributos elencados na Tabela 5.1 e para cada atributo foi atribuída uma nota indicando o nível de observância desejado. Para essa avaliação foi aplicada a mesma escala usada na primeira abordagem. O Apêndice B apresenta o resultado da avaliação dos atributos relativamente à infraestrutura.

A definição do conjunto final dos atributos mais relevantes foi realizada levando em consideração as duas abordagens. Uma vez coletados os resultados das pesquisas, foi determinado o conjunto de atributos relevantes que era a intersecção entre os resultados observados sob o ponto de vista da infraestrutura e os resultados observados sob o ponto de vista das diferentes classes de simuladores. A Figura 5.1 apresenta o resultado geral das duas abordagens (em azul a perspectiva dos tipos de simuladores, em laranja a perspectiva da infraestrutura de simulação) indicando os sete atributos mais relevantes (50% da lista inicial).

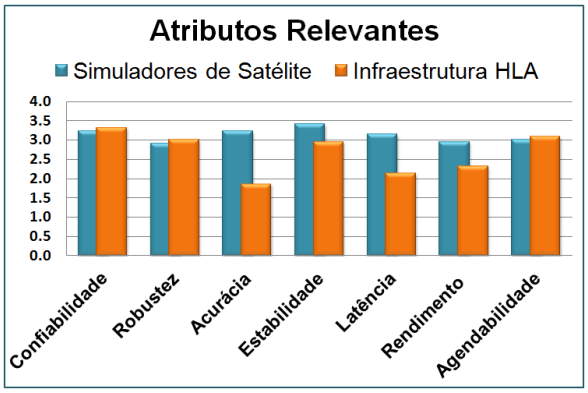


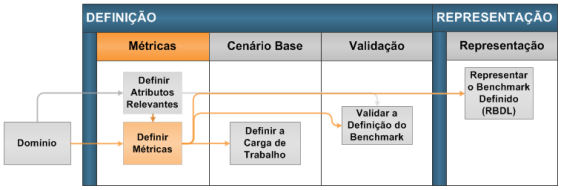
Figura 5.1 - Atributos de Dependabilidade Relevantes.

Nos benchmarks definidos com base nesta tese, os atributos relevantes serão avaliados sempre sob a perspectiva da infraestrutura de simulação, i.e., não serão avaliados modelos simulados. Por exemplo, quando se fala de forma genérica da *acurácia* de um modelo, fala-se de quanto os resultados deste

modelo estão de acordo com os resultados previstos para o seu equivalente físico. No nosso contexto, avaliar a *acurácia* significa verificar se um modelo mantém os resultados durante o intercâmbio de dados em presença da infraestrutura de simulação e em presença de mudanças às quais essa infraestrutura estará submetida.

Vale ressaltar que um benchmark deve ter por objetivo a avaliação de poucos atributos e poucas métricas. Assim, o conjunto de atributos identificados como mais relevantes para o domínio de infraestrutura de simuladores de satélite é um conjunto ainda demasiado abrangente e não é uma lista completamente fechada, mas visa ser um guia no processo de escolha dos atributos a serem avaliados durante a instanciação de benchmarks de resiliência concretos. A escolha dos atributos a serem medidos dependerá dos objetivos específicos dos usuários do benchmark em um dado contexto ou uso da infraestrutura HLA.

5.1.2. Métricas



Os resultados das medições feitas nos experimentos devem possibilitar a comparação de infraestruturas de simuladores que utilizam o padrão HLA ou permitir a avaliação de uma dada infraestrutura relativamente a um atributo de interesse. Seguindo a metodologia, serão apenas consideradas medidas empíricas que permitem comparar sistemas relativamente ao conjunto de atributos relevantes e que podem ser obtidas diretamente por meio dos experimentos conduzidos durante o benchmark.

Para a execução deste passo, nós utilizamos uma simplificação da metodologia GQM com o intuito de sistematizar a definição de métricas. O GQM foi utilizado da seguinte forma: o benchmark foi considerado como objetivo geral (avaliação) e como subobjetivo foi considerado cada atributo relevante e, quando era o caso, a meta em relação ao mesmo (*goal*); em seguida, foram levantadas questões acerca dos fatores do sistema que poderiam estar

relacionados ou ter impacto sobre esses atributos (*questions*); e por fim, foram derivadas métricas que seriam capazes de quantificar as respostas das questões (*metric*). Buscou-se, ainda, definir de forma clara o objetivo de cada métrica, as fórmulas a serem utilizadas, a interpretação da medida e a unidade a ser adotada. É importante ressaltar que esteve fora do escopo deste trabalho apresentar novas métricas para os atributos avaliados. Assim, as métricas sistematizadas são métricas já amplamente usadas e validadas.

A Tabela 5.2 apresenta exemplos de métricas para os atributos Latência e Robustez, ilustrando a aplicação da GQM. O conjunto completo de métricas está apresentado no Apêndice B.

Tabela 5.2 - Métricas de Dependabilidade - Latência e Robustez.

Atributo: LATÊNCIA	**Objetivo: Avaliar**	**Ponto de Vista: Federação**
Q1 Qual o tempo entre a atualização de atributos por um federado (emissor) e o reflexo dessa operação em outro federado (receptor)?		
M1.1	Média de latência direta de atualização de atributos (*LUpdAvg*) (DRAKE et al., 2003)	
	Objetivo	Comparar as infraestruturas de simulação relativamente à latência no intercâmbio de dados entre modelos (federados).
	Descrição	Média do tempo de latência direta, ou seja, diferença entre o tempo de atualização de atributos no federado emissor e o tempo de reflexo da operação no federado receptor.
	Fórmula	$LUpdAvg = \frac{1}{P} \sum_{i=1}^{P} (tR - tU)$ tR: tempo do recebimento dos dados pelo receptor tU: tempo na atualização do dado pelo federado (emissor) P: número de atualizações de atributos
	Interpretação	0 < LUpdAvg (quanto menor, melhor)
	Unidade	ms (milissegundos)
Atributo: ROBUSTEZ	**Objetivo: Avaliar**	**Ponto de Vista: Federação**
Q1 Com qual frequência a infraestrutura de simulação falha de forma generalizada (catastrófica) em presença de perturbações? (ISO/IEC, 2002)		
M1.1	Taxa de Falhas catastróficas (*RFTCas*) (FERNSLER; KOOPMAN, 1999)	
	Objetivo	Comparar a robustez das infraestruturas de simulação relativamente a falhas catastróficas.
	Descrição	Taxa de falhas catastróficas em presença de perturbações. Falhas catastróficas são aquelas que terminam a simulação, parando a infraestrutura de simulação (processo *RTIExec*) ou o sistema como um todo.
	Fórmula	$RFTCas = \frac{FCas}{P}$ FCas: número de falhas catastróficas P: número de falhas injetadas
	Interpretação	0 < $RTFCas$ (quanto menor, melhor)

A metodologia GQM também foi usada para sistematizar as métricas do atributo geral *Resiliência*. A Tabela 5.3 apresenta exemplos de métricas de resiliência definidas e utilizadas no contexto deste trabalho.

Tabela 5.3 - Métricas de Resiliência.

Atributo: RESILIÊNCIA	**Objetivo: Avaliar**	**Ponto de Vista: Federação**
Q2 Qual o impacto das *mudanças* nas diferentes métricas dos atributos de dependabilidade?		
M1.1	Índice de resiliência - estabilidade relativa (*RRindex*)	
	Objetivo	Comparar a resiliência das infraestruturas de simulação relativamente a uma dada métrica de dependabilidade.
	Descrição	Índice de resiliência relativo à estabilidade da infraestrutura em face de diferentes *mudanças*. O índice se aplica ao conjunto de métricas de dependabilidade e é calculado com base no coeficiente de variabilidade para todo o experimento. Dessa forma, considera-se a medida de referência como valor esperado e calcula-se o desvio padrão relativo usando as medidas das execuções com *mudança*.
	Fórmula	$RRindex = \frac{\sqrt{\frac{1}{T}\sum_{i=1}^{T}(EC_i - ER)^2}}{\overline{ER}}$ T: Número de instâncias da mudança aplicada ER: Medida de execução de referência EC_i: Medida da execução da i-ésima mudança
	Interpretação	0< $RSIndex$ (quanto menor, melhor)
Q2 Qual a perda de serviço observada nas medidas dos atributos de dependabilidade quando em face de mudanças?		
M2.1	Índice de resiliência - perda de serviço (*RLIndex*) (ALMEIDA; VIEIRA, 2013)	
	Objetivo	Comparar a resiliência das infraestruturas de simulação relativamente a uma dada métrica de dependabilidade.
	Descrição	Índice de resiliência em relação à oferta de serviços da infraestrutura em face de diferentes perturbações. O índice se aplica ao conjunto de métricas de dependabilidade e é calculado usando a média da relação entre a medida obtida na execução de referência e a medida obtida quando em presença de mudanças.
	Fórmula	$RLIndex = \frac{1}{T}\sum_{i=1}^{T}\left(\frac{EC_i}{ER}\right)$ T: Número de instâncias da mudança aplicada *ER*: Medida de execução de referência EC_i: Medida da execução da i-ésima mudança Obs.: quando o objetivo da métrica de dependabilidade é a diminuição em seus valores, a seguinte fórmula deve ser usada: $RLIndex = \frac{1}{T}\sum_{i=1}^{T}\left(\frac{EC_i}{ER}\right)^{-1}$
	Interpretação	$\text{RLIndex} \leq 1.0 \leq \text{RLIndex}$ (quanto maior, melhor)

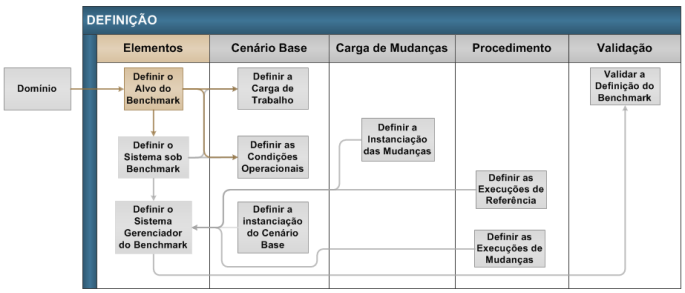
É importante salientar que as métricas de resiliência referem-se à variabilidade de cada uma das métricas de dependabilidade definidas no Apêndice B. Por exemplo, temos a *LUpdAvg* (Tabela 5.2), que é a latência média de atualização de atributos, como potencial métrica de Latência e o índice de resiliência

RRIndex aplicado a essa métrica será o coeficiente de variação das medidas obtidas quando em face de diferentes mudanças.

5.2. Elementos do Benchmark para Infraestruturas HLA

Conforme indicado na metodologia, os elementos do benchmark devem ser definidos a partir da análise do domínio de interesse, neste caso os componentes de uma *Infraestrutura de Execução HLA* (RTI). As subseções a seguir apresentam a delimitação do Alvo do Benchmark, a definição dos elementos que interagem com a RTI e a proposta de uma arquitetura para o Sistema de Gerenciamento de Benchmark.

5.2.1. Alvo do Benchmark



O objetivo da abordagem de benchmarking proposta é a avaliação de infraestruturas de simulação HLA que possibilitam a comunicação e o intercâmbio de dados entre modelos da simulação, a execução de serviços e a gestão da sequência dessas operações no tempo. No padrão HLA essas tarefas estão concentradas na *Infraestrutura de Execução* (RTI) que é composta pelo processo central da RTI (*RTIExec*), pela biblioteca RTI (*RTILib*) utilizada pelos federados para se comunicar com o processo central e demais federados, e pelo canal de comunicação. Esses elementos compõem o Alvo do benchmark conforme apresentado na Figura 5.2.

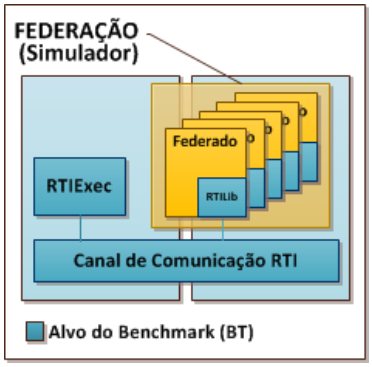
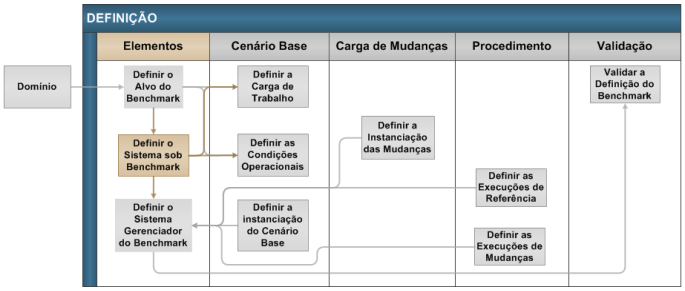


Figura 5.2 - Benchmark de Resiliência - Alvo do Benchmark (BT).

5.2.2. Sistema Sob Benchmark (SUB)



O sistema sob benchmark incorpora os elementos que interagem com a Infraestrutura de Execução (RTI), neste caso, a federação como um todo (modelos do simulador), os elementos que assistem o benchmark, por exemplo, o injetor mudanças ou falhas, e os elementos que apoiam a operação do BT, tais como o sistema operacional, o hardware e o canal de comunicação.

A Figura 5.3 ilustra os elementos SUB para o domínio de infraestrutura de simuladores HLA.

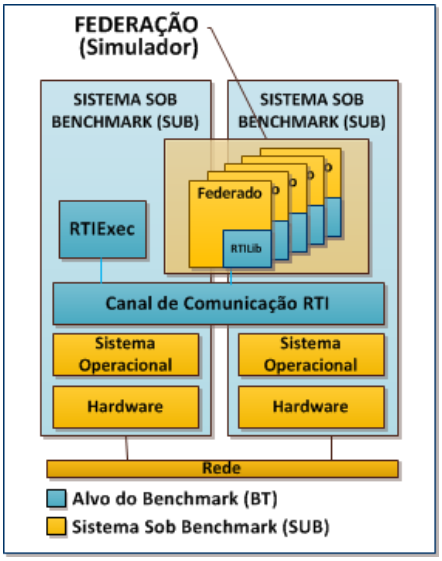
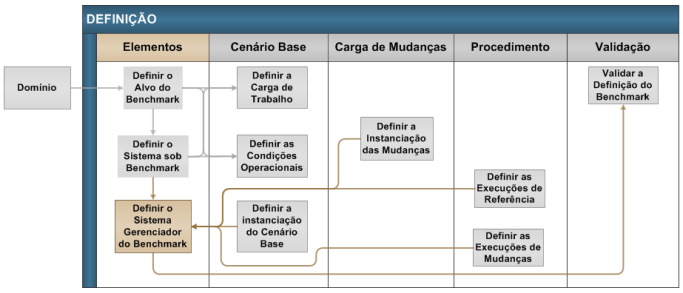


Figura 5.3 - Benchmark de Resiliência - Sistema Sob Benchmark (SUB).

5.2.3. Sistema Gerenciador do Benchmark (SGB)



No domínio de infraestruturas de simuladores baseadas em HLA, o SGB é o responsável por: (i) gerar as federações de referência que compõe a carga de trabalho; (ii) gerar a configuração da RTI HLA e do SUB; (iii) gerar as federações modificadas; (iv) iniciar, monitorar e finalizar a execução tanto do experimento de referência, quando dos experimentos com mudanças,; e (v) coletar e processar os dados de execução armazenados no SUB. A partir dos dados de execução coletados, a ferramenta de análise deve ter a capacidade de gerar os resultados do benchmark para os produtos avaliados. Os requisitos gerais do SGB para infraestruturas HLA estão descritos no Apêndice B.

A Figura 5.4 apresenta uma visão geral da arquitetura do SGB.

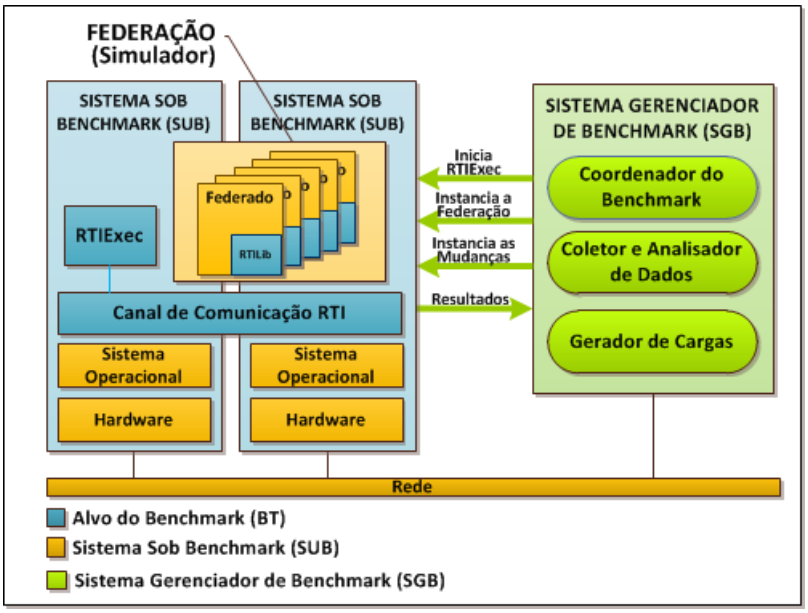
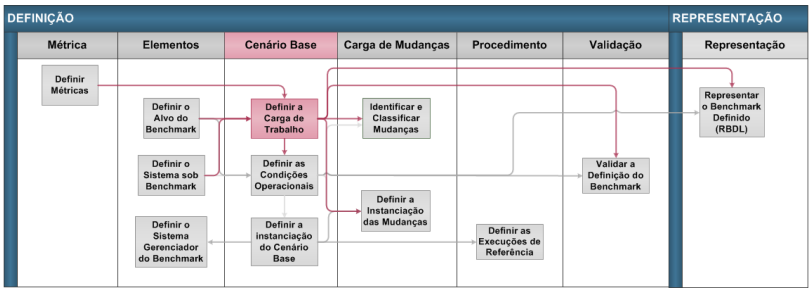


Figura 5.4 - Benchmark de Resiliência - Sistema Gerenciador de Benchmark (SGB).

5.3. Cenário Base

As subseções seguintes apresentam a definição das cargas de trabalho e das condições operacionais típicas que compõem o *Cenário Base* no domínio de *Infraestrutura de Execução* HLA.

5.3.1. Carga de Trabalho



Do ponto de vista da infraestrutura de simuladores de satélites, os modelos simulados (federações) que representem satélites típicos no contexto do INPE e os tipos de simuladores apresentados pelo memorando técnico ECSS (2010) são considerados como cargas de trabalho.

Vale ressaltar que o benchmark de resiliência para infraestrutura simuladores de satélite pode ser instanciado em diferentes estudos, com base em diferentes tipos de simuladores, com o objetivo de medir os atributos mais relevantes para cada tipo. Por esses motivos, não é definida uma carga de trabalho específica, mas sim a arquitetura da carga de trabalho e os elementos e parâmetros que caracterizam diferentes cargas de trabalho em relação às suas principais propriedades.

A Figura 5.5 apresenta a arquitetura de uma federação usada como carga de trabalho neste domínio, a qual é composta por:

- Federado de Monitoramento e Controle (M&C): responsável por construir a federação, iniciar e finalizar um experimento específico (conjunto de rodadas), iniciar e finalizar cada rodada de simulação;

- Federados Típicos: modelos sintéticos que representam modelos reais (modelo de ambiente, térmica, suprimento de energia, etc.). Esses modelos

podem executar serviços de outros modelos e publicam e subscrevem a dados (telemetrias), e podem estar submetidos a uma carga de processamento. Esses federados são modelos adaptados para a estrutura HLA e seguem o *template* geral apresentado no Apêndice A. Vale observar que o *template* pode mudar de acordo com as características do benchmark concreto, por isso, o tipo de *template* utilizado é um dos parâmetros explicitamente definido na carga de trabalho.

- Federados Injetores: responsáveis por injetar federados modificados, federados com falhas ou instanciar mudanças específicas (são federados opcionais).

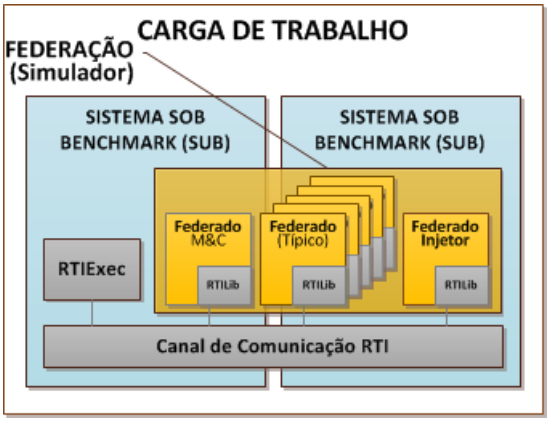


Figura 5.5 - Benchmark de Resiliência - Carga de Trabalho (*workload*).

Uma vez definida a arquitetura, devem-se definir os elementos e parâmetros que representam a carga de trabalho. No caso da infraestrutura de simulação HLA os elementos da carga de trabalho foram definidos considerando as seguintes perspectivas:

- Objeto Simulado: características gerais do satélite a ser simulado que está representado na carga de trabalho pelo número de modelos, volume de dados intercambiados, ligação entre modelos, carga dos modelos, etc.;

- Tipo de simulador: características do tipo de simulador escolhido usando como referência os tipos de simuladores ECSS (ECSS, 2010), por exemplo, repetibilidade exigida, frequência do passo de simulação (ver Capítulo 2, Tabela 2.1);

- Projeto do Federado HLA: características de implementação do federado em relação ao padrão HLA, por exemplo, número *Classes de Objetos* utilizados no intercâmbio de dados, número de *Classes de Interação*, serviços HLA utilizados, modelo de *callback,* API utilizada (linguagem e versão).

A Tabela 5.4 apresenta os parâmetros considerados na composição de uma carga de trabalho para infraestruturas de simuladores de satélites baseadas em HLA.

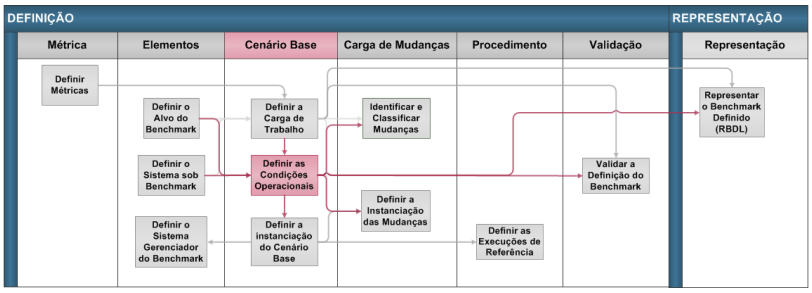
Tabela 5.4 - Elementos da Carga de Trabalho.

Elementos da Carga de Trabalho	
Elementos	**Descrição**
Características Gerais - Simulador	
Número de modelos (federados)	Número de modelos que compõe a federação.
Dados intercambiados	Volume de dados intercambiados, expressos pelo número de atributos.
Tipo de dados intercambiados	Tipo de dado de cada atributo.
Serviços assíncronos	Chamadas assíncronas a serviços dos federados (*classes de interação*).
Parâmetros dos serviços assíncronos	Parâmetros das chamadas assíncronas (número de parâmetros).
Tipo de dados dos parâmetros	Tipo de dado de cada parâmetro do serviço assíncrono.
Ligação entre modelos	Arquitetura da interligação entre os modelos da simulação (aos pares, circular, etc.).
Repetibilidade	Indicação da necessidade de repetibilidade no resultado da simulação.
Frequência do passo de simulação	Frequência do passo de simulação medida em Hz.
Comportamento do modelo	Carga que simula o comportamento de um modelo.
Características - HLA	
Classes de Objetos - tipos	Número de tipos de *Classes de Objetos* para intercambiar dados.
Classes de Interação - tipos	Número de tipos de *Classes de Interação* usadas para chamadas a serviços.
Classes de Interação - número	Número de *classes de interação* chamada de serviços
Classes de Objetos - instâncias	Número de instâncias de *classes de objetos* para intercambiar dados (instâncias de cada tipo).
Modelo de *callback*	Modelo que indica como a RTI HLA chama serviços dos federados (ver Tabela 2.2)
Serviços HLA	Lista do grupo de serviços HLA usados pelos federados (Gerenciamento de Tempo, etc.).
Linguagem de programação da RTI	Linguagem de programação da API (C++, Java, etc.).
Versão da biblioteca RTI	Versão da RTILib utilizada (p. ex. RTI 1.3, RTI 1516).

Cargas de trabalho especificadas por meio da definição de valores para cada um dos parâmetros definidos e que podem ser diretamente utilizadas em benchmarks concretos são apresentadas no Apêndice B.

O HLA é utilizado em diferentes domínios: aeronáutico, militar, etc. e, na área espacial, em sistemas espaciais como sondas, lançadores, etc. A análise de cargas de trabalho para esses domínios esteve fora do escopo deste trabalho. No entanto, vários parâmetros definidos para a carga de trabalho de infraestruturas de simuladores de satélites baseadas em HLA podem ser adaptados, especialmente em termos de escala, a domínios correlatos e outros mais podem ser adicionados para representar novos objetos simulados ou domínios.

5.3.2. Condições Operacionais



As condições operacionais para infraestruturas baseadas em HLA são definidas por meio da configuração da Infraestrutura de Execução (RTI), das federações e dos elementos do SUB de referência.

Vale salientar que, assim como ocorre com as características da carga de trabalho, as condições operacionais também compõem uma fonte de mudanças. Além disso, pode-se definir mais de uma condição operacional para posterior escolha na composição do *Cenário Base* durante a instanciação dos benchmarks concretos, permitindo que se avalie o BT em diferentes contextos.

5.3.2.1. Configuração do BT - Padrão HLA

Neste subpasso são definidos os parâmetros de configuração relevantes para a execução de uma federação HLA.

A configuração dos federados será feita por meio dos arquivos de configuração da federação (FOM), já a configuração geral da RTI será realizada por meio dos arquivos de configuração de cada implementação (RID). Enquanto os arquivos FOM serão gerados automaticamente para cada federação, os arquivos RIDs de base serão fixos para cada produto avaliado, já que o formato destes arquivos é proprietário. A Tabela 5.5 apresenta os parâmetros HLA a serem configurados.

Tabela 5.5 - Condições Operacionais - Configuração HLA.

Configuração HLA		
Parâmetro	**Configuração**	**Descrição**
Taxa de Atualização	Arquivo FOM	Representa a taxa na qual os dados são atualizados e os serviços são chamados.
Chaves de configuração (*Advisory Keys*)	Arquivo FOM	Configura as mensagens informativas que são trocadas entre a RTI e os federados.
Tipo de transporte	Arquivo FOM	Configura o tipo de transporte usado para troca de dados ou chamada a serviços (confiável ou não-confiável).
Diretório do arquivo de salvamento do estado da federação	Arquivo RID	Configura o diretório de salvamento do arquivo de estado da federação gerado pela RTI e pelos federados.
Configurações da rede - *bundling*	Arquivo RID	Configura a rede em relação ao uso de empacotamento (*bundling*).
Configurações da rede - *multicast*	Arquivo RID	Configura a rede em relação ao uso de UDP multicast.

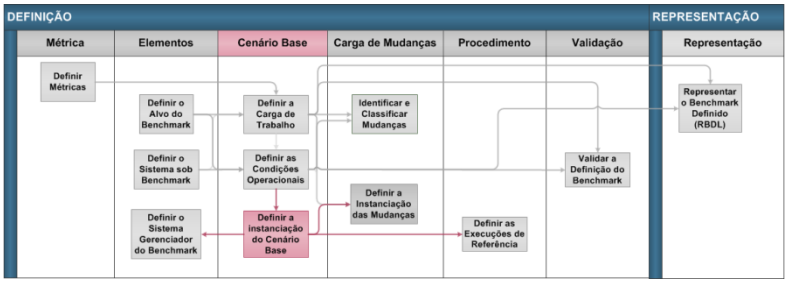
5.3.2.2. Configuração de Referência do SUB

Neste passo são especificados os parâmetros que definem as arquiteturas de referência para o SUB. No caso da infraestrutura de simuladores, a configuração do SUB inclui a definição do sistema operacional, do hardware e do canal de comunicação utilizados como referência. A Tabela 5.6 descreve os elementos que devem ser definidos como base para as condições operacionais.

Tabela 5.6 - Condições Operacionais - Configuração SUB.

Configuração do SUB	
Item	**Descrição**
Hardware	
Cardinalidade	Número de equipamentos utilizados.
Processador	Informações sobre o processador (modelo, número de núcleos, tamanho da memória *cache*).
Memória	Capacidade.
Disco rígido	Capacidade.
Categoria	Categoria do hardware (PC, servidor, estação de trabalho, etc.).
Sistema Operacional	
Classe	Classe do sistema operacional (tempo real, etc.).
Nome	Definição do sistema operacional.
Versão	Versão do sistema operacional.
Atualização	Versão da atualização ou correção de falhas instalada (*patches*).
Canal de Comunicação	
Tipo	Tipo de rede (LAN, WAN).
Infraestrutura/ Arquitetura	Tipo de infraestrutura utilizada (ethernet, wireless).
Banda	Capacidade da banda.

5.3.3. Instanciação do Cenário Base



O Sistema de Gerenciamento do Benchmark (SGB) deve ser capaz de gerar uma federação HLA automaticamente a partir dos parâmetros definidos para a carga de trabalho, tais como número de modelos, número de atributos intercambiados, etc. O sistema deverá, também, gerar o arquivo FOM que representa a federação e considerar que a federação será instanciada nas máquinas estabelecidas na configuração do SUB, definindo assim a distribuição dos federados instanciados. A federação será gerada a partir dos elementos definidos nos dois passos anteriores e representados por meio da linguagem RBDL descrita no Capítulo 6.

Relativamente à instanciação da carga de trabalho, o SGB deve ser capaz de instanciá-la remotamente. A Figura 5.6 ilustra a dinâmica de instanciação da federação.

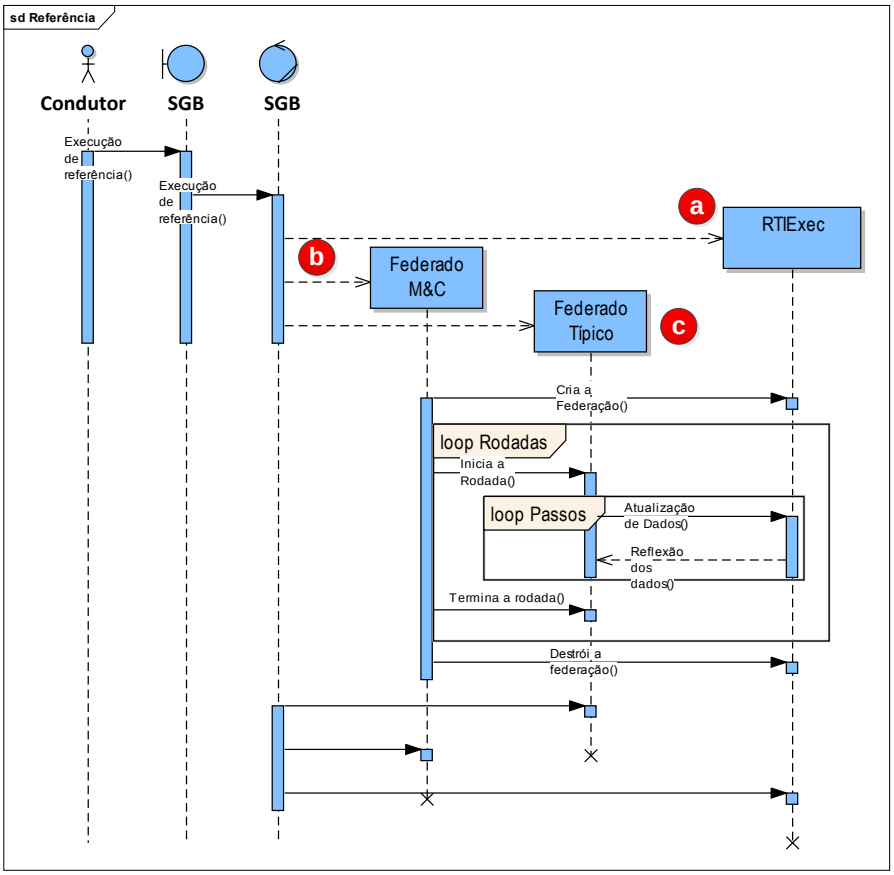


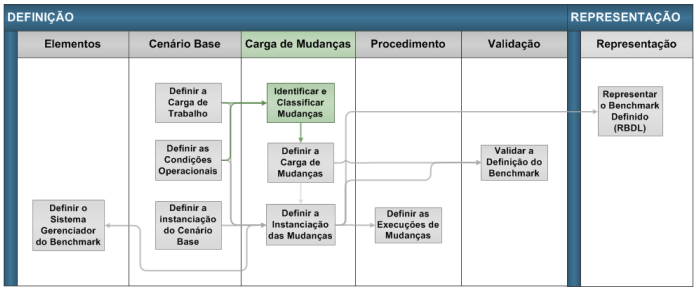
Figura 5.6 - Instanciação do Cenário Base

Conforme mostra a Figura 5.6, o SGB deverá iniciar a RTI (*RTIExec*) da implementação HLA em avaliação (item (a)); em seguida, deverá instanciar o federado de Monitoramento e Controle (M&C) responsável por iniciar e terminar as rodadas de simulação (item (b)); por fim, deverá instanciar sequencialmente os demais federados nas máquinas definidas por meio da configuração do SUB (item (c)).

5.4. Carga de Mudanças no Contexto de Infraestruturas HLA

As subseções seguintes apresentam os passos realizados na definição da carga de mudanças para benchmarks de resiliência de uma infraestrutura de simuladores de satélite baseada em HLA, bem como apresenta uma carga de mudanças para esse domínio.

5.4.1. Mudanças



Neste passo nós utilizamos os potenciais elementos de mudança, tais como os elementos que caracterizam a carga de trabalho (Tabela 5.4), as condições operacionais (Tabelas 5.5 e 5.6) e os elementos do SUB e analisamos cada um deles relativamente às classes de mudanças definidas pela metodologia.

Relativamente à infraestrutura de simulação baseada em HLA, os elementos que podem originar mudanças estão marcados em vermelho na Figura 5.7.

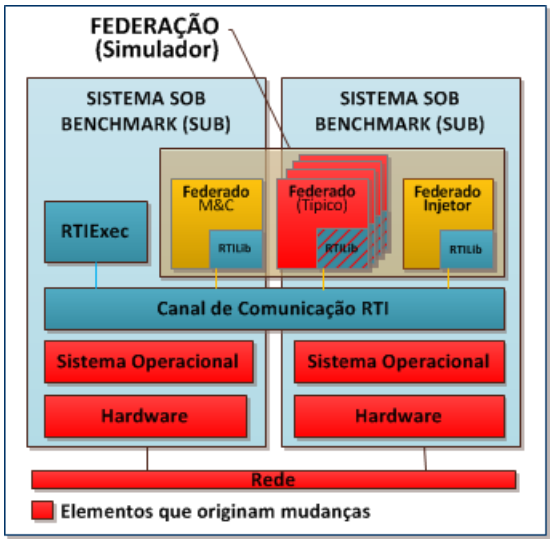


Figura 5.7 - Benchmark de Resiliência - Elementos Origem de Mudanças.

A partir das classes de mudanças previamente estabelecidas (Tabela 4.1) e dos elementos originadores de mudança apresentados na Figura 5.7, foi definido um conjunto de 109 mudanças. As Tabelas 5.7 a 5.9 apresentam exemplos de mudanças identificadas para esses elementos em diferentes classes e subclasses de mudanças.

Tabela 5.7 - Mudanças Tecnológicas.

MUDANÇAS TECNOLÓGICAS			
Id	**Mudança**	**Descrição**	**Subclasse**
Federação (RTILib)			
F2	Federados desenvolvidos em diferentes linguagens de programação	Inserção de federados desenvolvidos em diferentes linguagens de programação (API da linguagem).	Implementação
F3	Versões da RTILib	Inserção de federados que utilizam diferentes versões da RTILib (federados legados).	Versão
Canal de Comunicação			
C3	Rede Wireless	Federação distribuída através de rede wireless.	Implementação
C5	Aumento da banda da rede	Aumento na capacidade da banda de rede utilizada pela federação.	Recurso
Sistema Operacional			
OS1	Atualização no Sistema Operacional utilizado - RTIExec	Patches de atualização do sistema operacional utilizado pela processo RTIExec.	Versão
OS6	Mudança nos sistemas operacionais - RTILib	Mudança no sistema operacional utilizado pelos federados - RTILib (Linux, OSMac e Windows).	Implementação
Hardware			
H1	Aumento na capacidade do processador	Uso de máquinas com processadores de maior desempenho.	Recurso

Tabela 5.8 - Mudanças de Requisitos.

MUDANÇAS de REQUISITOS			
Id	**Mudança**	**Descrição**	**Subclasse**
Federação (RTILib)			
F8	Passo de simulação *as-fast-as-possible*	Simuladores com requisitos do tempo de passo de simulação *as-fast-as-possible*.	Objetivo
F19	*Threads* em federados para *callback*	Mudanças no projeto de federados para uso de *callback* em *thread* separada.	Projeto
Hardware			
H8	Aumento no número de máquinas na simulação	A complexidade do simulador como um todo ou a complexidade de alguns modelos pode exigir um aumento no número de máquinas usadas na simulação distribuída.	Escala

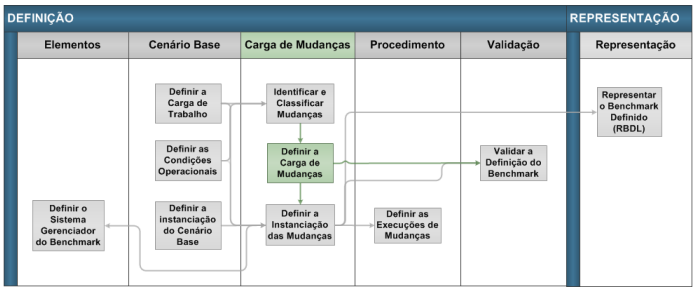
Tabela 5.9 - Mudanças Ambientais.

MUDANÇAS AMBIENTAIS			
Id	**Mudança**	**Descrição**	**Subclasse**
Federação (RTILib)			
F28	Variação na frequência do passo de simulação	Variação na frequência do passo de simulação dos federados (aceleração).	Perfil de uso
F30	Tipo de transporte confiável (*reliable*)	Troca de mensagens configuradas como *reliable*. Prioriza a entrega da mensagens de forma confiável.	Configuração
Canal de Comunicação			
C11	Falha na interface da rede	Falha ocorrida nas interfaces de rede das máquinas do SUB.	Falhas de Hardware
Sistema Operacional			
OS12	Falha de atribuição - MVAV	Falha de software ODC (tipo atribuição) injetada no sistema operacional por meio do operador MVAV (DURÃES; MADEIRA, 2006).	Falhas de Software
Hardware			
H10	Variação no padrão de uso dos processadores	Uso do sistema com variações na carga dos processadores das máquinas.	Perfil de uso
H15	Falha no suprimento de energia - RTIExec	Problema no suprimento de energia da máquina onde é executado o RTIExec.	Falhas de Hardware

Para o levantamento de mudanças relacionadas a falhas, foram consideradas falhas de operação, falhas de hardware de alto nível e falhas de software. Relativamente às falhas de software, tanto as aplicáveis ao sistema operacional, quanto aos federados, foram consideradas como falhas de interesse as falhas remanescentes nos sistemas entregues, ou seja, o conjunto das falhas não detectadas previamente pelos testes. Os trabalhos de Durães e Madeira (2003, 2006) mostraram que aproximadamente 50% das falhas latentes nos softwares entregues estão relacionadas a um conjunto de treze tipos de falhas que nesses trabalhos foram representados por um conjunto de operadores. Esse conjunto de operadores, considerado na composição da carga da carga de falhas em vários trabalhos de benchmark de dependabilidade (VIEIRA, 2005; VIEIRA et al., 2008; DURÃES et al., 2008), também fazem parte a carga de mudanças definida nesta tese.

5.4.2. Carga de Mudanças

Este passo compreende a normalização e a avaliação da lista completa de mudanças.



A normalização foi feita usando o critério de definição de parâmetros adicionais e a análise de efeitos da mudança, conforme estabelecido na metodologia. Depois da aplicação deste subpasso a carga de mudanças ficou reduzida a um conjunto de 61 mudanças.

Relativamente à representatividade da carga de mudanças, nós utilizamos a avaliação de especialistas para realizar a análise do nível de exposição do sistema às mudanças normalizadas e, sempre que disponível, como no caso de falhas de software, nós utilizamos dados da literatura (DURÃES; MADEIRA, 2003, 2006). Assim, os trabalhos de Durães e Madeira (2003, 2006), já utilizados na fase de levantamento de mudanças, também foram utilizados na avaliação da probabilidade de ocorrência de uma falha, já que esses trabalhos

definem, para cada tipo de falha considerada (operadores), a sua porcentagem de ocorrência.

Importante frisar que, embora o trabalho de Moraes et al. (2006) tenha concluído que falhas de interface não representam de forma adequada as falhas residuais nos softwares, uma vez que o padrão HLA fornece uma API extensa e com peculiaridades de uso, nós consideramos as falhas de interface isoladamente e atribuímos um peso maior que a probabilidade definida na literatura por considerar que o uso incorreto da API é um importante fator de risco na integração e uso de novos modelos simulados.

A Tabela 5.10 apresenta exemplos de mudanças classificadas com diferentes índices de exposição.

Tabela 5.10 - Avaliação das Mudanças.

RESULTADOS - Análise de Exposição					
ID	**SubClasse**	**Prazo**	**Probabilidade**	**Impacto**	**Exposição**
Federação (RTILib)					
CM11	Aumento no número de parâmetros nas chamadas a serviços de outros federados	médio	baixa	marginal	**Baixo**
Canal de Comunicação					
CM42	Variação no tráfego da rede	curto	alta	crítico	**Muito Alto**
Sistema Operacional					
CM49	Atualização no Sistema Operacional - RTILib	curto	alta	marginal	**Alto**
CM54	Falhas de software ODC (operadores) Obs.: operadores de falha de software definidos em Durães e Madeira (2006)	curto	baixa	marginal	**Médio**
Hardware					
CM55	Mudança na capacidade do processador (aumento, diminuição)	curto	muito alta	crítico	**Mandatório**
CM60	Variação no padrão de uso dos processadores.	curto	alta	crítico	**Muito Alto**

A Figura 5.8 apresenta a evolução da carga de mudanças à medida que foram executados os subpassos indicados pela metodologia para a sua composição. O nível de corte escolhido para exposição do sistema foi “Alto”.

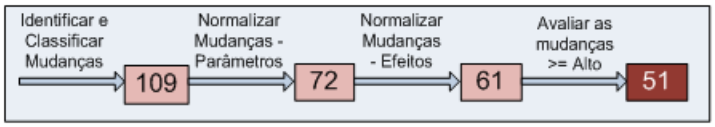
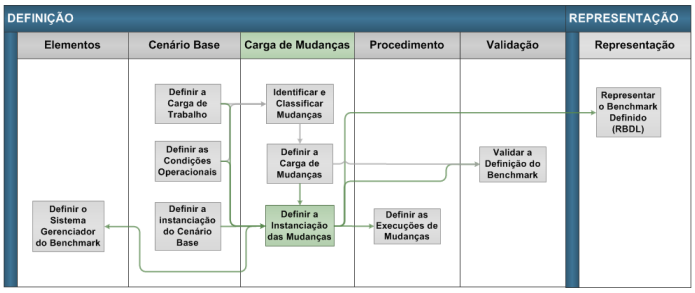


Figura 5.8 - Composição da Carga de Mudanças.

5.4.3. Instanciação das Mudanças



Neste passo, para cada mudança que compõe a carga de mudanças, foram especificados o elemento originador e os parâmetros de geração e instanciação da mesma. A Tabela 5.11 apresenta exemplos de atributos de mudança no domínio de infraestruturas HLA.

Tabela 5.11 - Atributos das Mudanças.

ATRIBUTOS						
Id	Operador	Mudança	Fonte	Parâmetros		
				Geração	Instanciação	Ambos
CM3	FRTIVer	Mudança na versão da RTILib	Versão da biblioteca RTI		numDeModelos	versao
CM8	FObjects	Aumento no volume de dados trocados entre federados - número de classes	Classes de Objetos		numDeModelos	numObjetos
CM16	FCallback	*Threads* de federados para *callback*	Template HLA	template	numDeModelos	tipoCallback

Da mesma forma, a cada mudança, foi associado um método de *Geração*, um método de *Instanciação* e, quando pertinente, um método de *Recuperação*. A Tabela 5.12 apresenta exemplos de métodos.

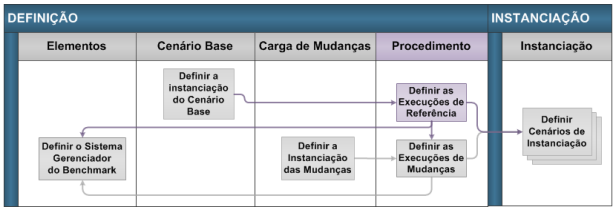
Tabela 5.12 - Métodos das Mudanças.

MÉTODOS					
Id	Operador	Mudança	Geração	Instanciação	Recuperação
CM9	FAttrib	Aumento no volume de dados trocados entre federados - número de atributos	FEDModification(operador, tipo, mudança)	Substituion (operador, tipo, mudança, numDeModelos)	NA
CM29	FErrSaveFile	Erro na configuração do diretório do arquivo de salvamento de estado.		RIDSubstitution(nameArq)	NA
CM38	FIntFault	Falhas de Interface (operador)	InterfaceFaultInjection(arqtipo)	InjectFailModel(operador, numRodada)	KillProccess()

É importante observar que um único método pode ser utilizado para a geração e/ou execução de uma ou mais mudanças.

O Apêndice B apresenta a carga de mudanças, bem como os requisitos dos métodos de geração e instanciação de mudanças que foram implementados para a execução dos benchmarks apresentados no Capítulo 7.

5.5. Procedimentos de Benchmark



As subseções seguintes apresentam a etapa de especificação das regras e procedimentos de execução do experimento de benchmark no domínio de infraestrutura HLA.

No caso das infraestruturas HLA, a execução de referência se dará por meio de rodadas similares a rodadas de simulação. Os experimentos são normalmente executados em várias rodadas com um intervalo entre elas e um período de estabilização no início e fim de cada uma. Para tanto, foram definidos os seguintes parâmetros de execução:

- Número de rodadas: número de repetições das rodadas de simulação a serem consideradas no experimento de benchmark. A repetição de rodadas visa reproduzir o uso da simulação em várias iterações, permitindo exercitar o reinício da comunicação entre os federados. Esse valor deve ser o mesmo tanto na execução de referência, quanto na execução com mudanças;

- Duração de uma rodada: tempo de execução de cada uma das rodadas de simulação;

- Intervalo: tempo de intervalo entre uma rodada de simulação e outra. Neste período os resultados obtidos durante a execução da rodada de simulação são armazenados nas máquinas locais;

- Período de estabilização: tempo, no início e fim de cada rodada, utilizado para estabilização do sistema. Os resultados não são coletados durante esse período.

No caso das simulações HLA, no início de cada execução é iniciado o processo *RTIExec* e criada a federação e, no final da execução, a federação é destruída. Os resultados dos experimentos são armazenados nas máquinas locais entre as rodadas de simulação e são coletados pelo SGB no final do experimento. A Figura 5.9 apresenta detalhes da Execução de Referência.

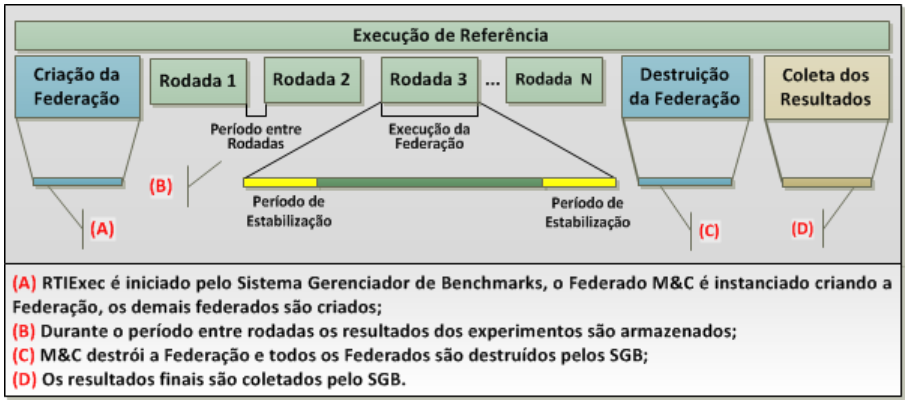


Figura 5.9 - Execução de Referência.

No caso da infraestrutura HLA, as rodadas de mudança seguem a mesma definição das rodadas de referência e a execução de cada mudança será representada somente pelos seus atributos de aplicação. Assim, nas execuções dos experimentos, as rodadas de mudança dependerão dos valores dos parâmetros e atributos de cada mudança que definem: a cardinalidade da mesma (Figura 5.10 item (E)), a duração de aplicação de cada mudança (Figura 5.10 item (F)) e o período de recuperação (Figura 5.10 item (G)).

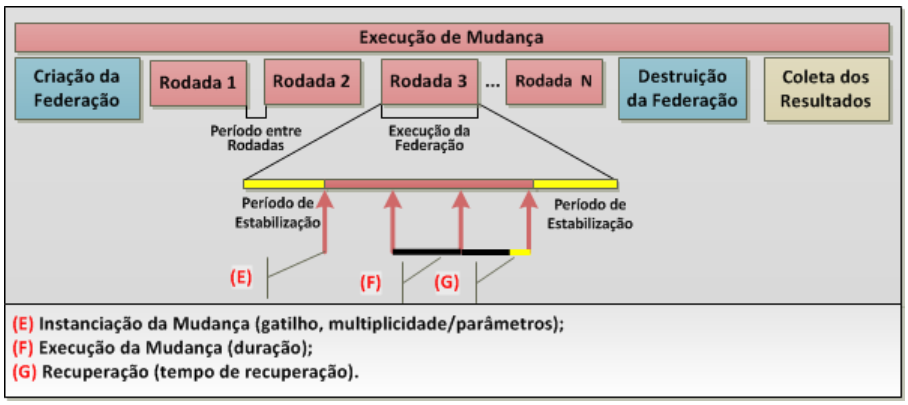
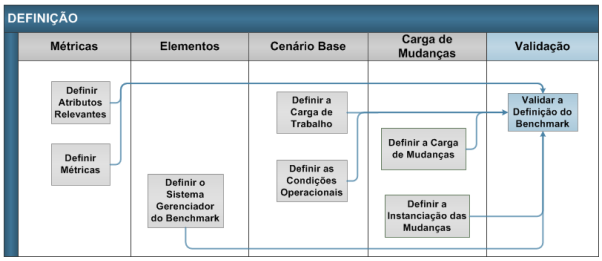


Figura 5.10 - Execução de Mudança.

5.6. Validação do Benchmark

Nesta seção são apresentados aspectos de validação dos elementos propostos na definição do benchmarking de resiliência para infraestruturas HLA.



A análise dos elementos de benchmark de resiliência definidos relativamente a cada uma das propriedades requeridas pelo benchmark é apresentada a seguir.

5.6.1. Relevância

No caso do benchmark de resiliência definido para infraestruturas de simuladores de satélite, foi realizada uma pesquisa com especialistas da área para classificar um conjunto de atributos de acordo com o seu grau de relevância para diferentes classes de simuladores. Realizou-se, ainda, uma avaliação da importância e do impacto desses atributos sobre o conjunto de requisitos funcionais de uma infraestrutura de simulação. Considera-se que as instâncias de benchmark que forem avaliadas utilizando um subconjunto dos atributos definidos como mais relevantes no domínio, também serão consideradas relevantes.

5.6.2. Representatividade

Conforme prescrito na metodologia, a representatividade geral de um benchmark de resiliência é atestada mediante a demonstração da representatividade da carga de trabalho, da carga de mudanças e das métricas definidas.

5.6.2.1. Carga de Trabalho

Embora o padrão HLA possa ser utilizado em diferentes contextos, a carga de trabalho considerada neste estudo está relacionada ao domínio de simuladores de satélites e as propriedades de carga de trabalho foram definidas com base

nas classes de simuladores apresentados no memorando técnico ECSS (2010). Na especificação do benchmark não foi definida uma carga de trabalho única, mas sim quais características definem a carga de trabalho em um determinado contexto, já que consideramos que a especificação pode ser usada em diferentes instâncias de um benchmark de resiliência. Assim, uma carga de trabalho será considerada representativa sempre que houver um valor válido representando cada uma das características identificadas. Serão considerados valores válidos todos os valores que representem satélites típicos no contexto do INPE e classes de simuladores de acordo com tipos de simuladores definidos pelo memorando técnico ECSS (2010). O Apêndice B apresenta exemplos de cargas de trabalho definidas.

5.6.2.2. Carga de Mudanças

A metodologia definida propôs avaliar a representatividade da carga de mudanças por meio da análise da exposição do sistema a cada mudança especificada e essa foi a abordagem usada para compor a carga de mudanças para o benchmark no domínio de infraestruturas HLA. No entanto, poderiam ter sido utilizadas outras abordagens.

O levantamento das mudanças às quais uma infraestrutura de simulação estaria exposta e a avaliação de representatividade da carga de mudanças foram realizadas levando em consideração os seguintes fatores: (i) arquitetura do SUB e características do Cenário Base, (ii) as classes de mudança definidas pela metodologia (Capítulo 4); e (iii) o resultado da entrevista com 4 especialistas da área que auxiliaram tanto na formulação da lista de mudança, quanto na avaliação do nível de exposição do sistema às mesmas.

Neste contexto, o conjunto de mudanças usado em um benchmark concreto será considerado representativo sempre que for composto por um subconjunto das mudanças especificadas durante a fase de definição do benchmark. Esse subconjunto pode ser selecionado usando um nível de corte de exposição do sistema a essas mudanças, classes de mudanças de interesse, ou uma única mudança que exprima um cenário desejado.

5.6.2.3. Métricas

Foi especificado um conjunto de métricas para o domínio de simuladores de satélite que atendem às seguintes propriedades: (i) podem ser diretamente obtidas de forma empírica; (ii) medem diretamente características dos produtos relativamente aos atributos relevantes; (iii) são objetivas e representadas por uma fórmula; (iv) permitem a comparação de produtos; (v) são simples de serem aplicadas e analisadas; e (vi) foram definidas métricas específicas para o atributo resiliência.

Relativamente a um benchmark concreto, considera-se que um conjunto representativo de métricas deve ser um subconjunto das métricas definidas na especificação do benchmark.

5.6.3. Simplicidade

Sob o ponto de vista da definição dos elementos do benchmark de resiliência, a validação da propriedade simplicidade se dá pela automação na geração de cargas de trabalho e mudanças e na execução do experimento. Os requisitos do Sistema Gerenciador de Benchmark para infraestruturas de simuladores baseadas em HLA cobrem os seguintes aspectos: (i) os federados que compõe a carga de trabalho são gerados automaticamente; (ii) os federados modificados são gerados automaticamente; (iii) sempre que necessário, o federado instanciador de mudanças é também gerado automaticamente como membro da federação; e (iv) ferramentas automatizadas tanto para a execução, quando para a análise dos resultados dos experimentos do benchmark estão previstas.

Os requisitos citados garantem que a facilidade e simplicidade da aplicação do benchmark estão previstas, mas não podem atestar a simplicidade na implementação da ferramenta, esse será um fator a ser atestado pelos executores do benchmark. No Capítulo 6 são apresentadas alternativas para o desenvolvimento de uma ferramenta de benchmark, incluindo um *framework* para benchmark de resiliência que contempla parte da implementação.

5.7. Considerações Finais

Este capítulo apresentou a especificação de uma abordagem de benchmarking de resiliência para infraestruturas de satélite baseadas no padrão HLA, demonstrando o uso da metodologia proposta no capítulo anterior. Foram também definidos o conjunto de elementos a serem utilizados nos benchmarks que podem ser efetivamente aplicados na avaliação de produtos RTI HLA.

É importante salientar que a especificação do benchmark de resiliência apresentado neste capítulo compreende as fases de definição conforme previsto na metodologia. Os estudos de caso do Capítulo 7 apresentam o uso dos elementos definidos por meio da instanciação, representação e aplicação de benchmarks concretos com diferentes objetivos.

6 LINGUAGEM DE DESCRIÇÃO PARA BENCHMARKS DE RESILIÊNCIA

Este capítulo apresenta uma linguagem para representação de benchmarks de resiliência no domínio de infraestruturas de simuladores de satélite baseadas no padrão HLA. Essa linguagem, chamada *Resilience Benchmarking Description Language* (RBDL), foi definida com base na *eXtensive Markup Language* (XML) e em Esquemas XML (*XML Schema*) e pode ser usada tanto para especificação de benchmarks de resiliência no domínio de infraestruturas HLA, como estendida a outros domínios e outros tipos de benchmark.

Além disso, uma vez que documentos XML são diretamente processáveis por programas computacionais, benchmarks definidos em RBDL podem ser processados por ferramentas de benchmarking tanto para a geração automática de elementos (carga de trabalho, mudanças etc.), quanto na execução dos experimentos.

6.1. A RBDL

A metodologia apresentada nos capítulos anteriores orientou as definições da RBDL. Desta forma, temos que os principais elementos que definem um benchmark de resiliência são: os *Objetivos de Dependabilidade e Resiliência* a serem avaliados, o *Cenário Base* e a *Carga de Mudanças*. A Figura 6.1 apresenta a definição dos elementos de um benchmark de resiliência, adaptado de Almeida e Vieira (2012a).

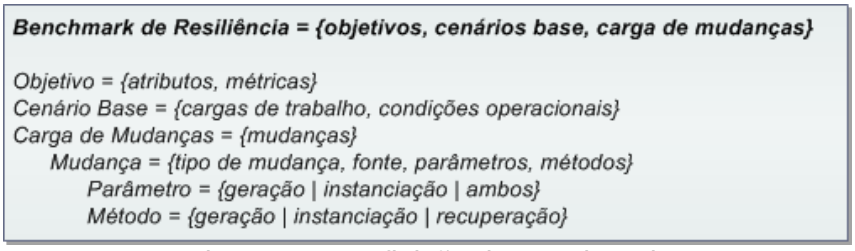


Figura 6.1 - Definição do Benchmark

A execução de um benchmark de resiliência pode ser expressa por *Cenários de Instanciação* nos quais um *Cenário Base* é submetido a um *Cenário de Mudança* de acordo com Regras de Execução. A Figura 6.2 apresenta os elementos da instanciação de um benchmark de resiliência.

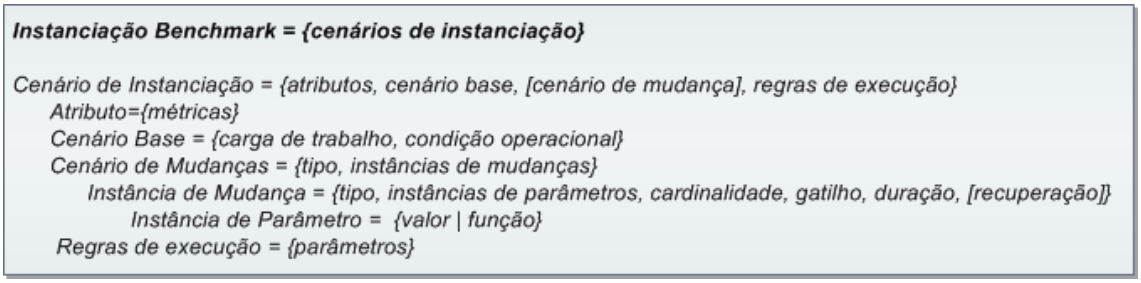


Figura 6.2 - Instanciação do Benchmark

A definição dos elementos de um benchmark conforme mostrado nas Figuras 6.1 e 6.2 sugere a possibilidade de formalização da especificação. Entretanto, um benchmark está sempre fortemente associado a um domínio de aplicação e, normalmente, os domínios são complexos e têm uma série de especificidades que dificultam a sua representação por meio de uma linguagem formal.

Notações que incorporam maior formalismo como a *Backus-Naur Form* (BNF) ou *Extended Backus-Naur Form* (EBNF) (ISO, 1996) foram avaliadas como alternativas para a especificação da RBDL. No entanto, nós optamos por especificar a RBDL por meio de uma linguagem semiformal, Esquemas XML, que auxilia na padronização, é simples e se beneficia da disponibilidade de um conjunto de ferramentas, comerciais e não comerciais, para a geração e visualização de benchmarks e construção de ferramentas.

6.1.1. Esquemas XML como Base para a RBDL

A XML é uma linguagem extensível e hierárquica que permite a representação de diferentes tipos de dados por meio da criação de etiquetas específicas (*tags*) e é amplamente utilizada para armazenamento e troca de informações. Um documento XML pode ter a sua estrutura definida e ser validado por meio de arquivos do tipo *Document Type Definition* (DTD) ou *XML Schema Definition* (XSD).

Os XSDs podem também ser vistos como uma metalinguagem por meio da qual novas linguagens derivadas podem ser especificadas para diferentes domínios. Esses esquemas permitem a especificação de *tags* para representar elementos, a criação de tipos simples e complexos que caracterizam esses

elementos, bem como a definição da hierarquia e da ordem a ser seguida na criação de um documento XML. A característica de extensibilidade da linguagem XML e a possibilidade de se utilizar esquemas XSD para a sua definição e validação fazem com que essa linguagem seja frequentemente usada para descrever domínios e como base de outras linguagens derivadas, como a *eXtensible Business Reporting Language* (XBRL) (BOVEE et al., 2005) e a *Election Markup Language* (EML) (WIKIPEDIA, 2013), entre outras.

Há na literatura um conjunto de melhores práticas e padrões de projeto amplamente usados para a definição de Esquemas XML, incluindo: *Boneca Russa (Russian Doll), Salame Fatiado (Salami Slice), Persiana (Venetian Blind), Jardim do Éden (Garden of Eden) e Camaleão (Chameleon).* Esses padrões têm nomes que sugerem a sua estrutura e diferem entre si principalmente em relação à localização dos elementos (locais ou globais) e à forma como esses elementos e os tipos da linguagem são utilizados. Diferentes padrões podem ser adotados de acordo com as necessidades e usos dos esquemas, por exemplo, alguns padrões priorizam o encapsulamento dos elementos, enquanto outros enfatizam a reutilização e a extensibilidade (HEWITT, 2009).

No padrão Boneca Russa é criado um elemento raiz único que contém todos os demais elementos de forma aninhada. Esse padrão produz Esquemas fáceis de serem escritos e lidos e privilegia o encapsulamento. No padrão *Salame Fatiado* todos os elementos são declarados de forma independente e global e os tipos de dados são locais. Esse padrão privilegia o reúso dos elementos, entretanto, como os elementos não são aninhados não há uma indicação de qual seja o elemento raiz e, embora os elementos possam ser reutilizados facilmente, o mesmo não ocorre com os tipos. O padrão *Persiana* reúne as vantagens dos dois anteriores e prevê um único elemento raiz global que é composto por tipos definidos externamente, misturando assim definições locais e globais. Este padrão maximiza o reúso e permite o uso de vários arquivos para compor o esquema. Já o padrão *Jardim do Éden* mistura os padrões *Salame Fatiado* e *Persiana*, nele todos os elementos e tipos são

definidos globalmente e referenciados quando necessário. Esse é o padrão que proporciona o máximo de reúso entre todos, já que tanto o elemento raiz, quanto os demais elementos e tipos podem ser reutilizados. Por fim, no padrão *Camaleão* os tipos comuns de dados são definidos em um esquema que não usa *namespace* e é normalmente incluído em outros Esquemas assumindo o *namespace* de destino. Esse é um padrão interessante para definir tipos básicos.

Na definição da RBDL foi utilizado o padrão de projeto *Camaleão* para definir os tipos genéricos. A Figura 6.3 mostra o uso desse padrão na linguagem por meio de um exemplo onde aparecem os tipos base da RBDL (a) e em destaque o tipo *formulaType* (b).

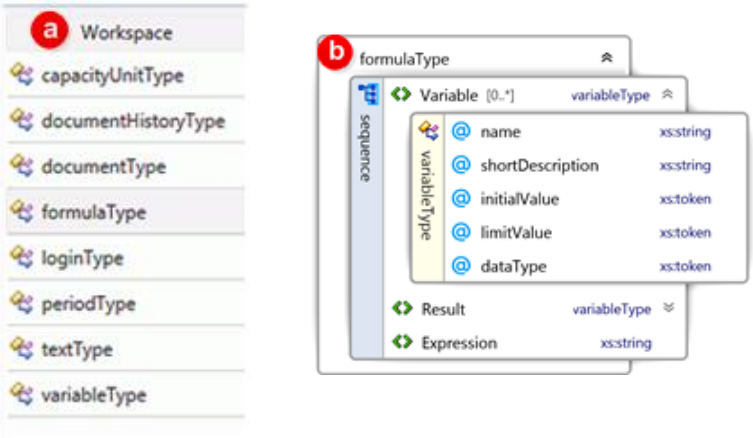


Figura 6.3 - Exemplo do padrão *Camaleão* na RBDL.

O padrão de projeto *Jardim do Éden* foi usado para definir os elementos dentro dos esquemas, buscando promover a reutilização tanto de tipos de dados, quanto de elementos. A escolha desse padrão permitiu que vários elementos e subelementos do benchmark fossem definidos de forma independente e depois referenciados quando necessário. Os elementos e tipos genéricos, aplicáveis a qualquer tipo de benchmark, permitem que alterações e extensões possam ser facilmente implementadas. Já em relação aos elementos específicos do domínio, o uso desse padrão permite que esses elementos sejam facilmente intercambiados para a utilização da linguagem em novos contextos. A Figura 6.4 exemplifica o uso do padrão de projeto *Jardim do Éden* na especificação da

RBDL, apresentando o elemento *Parameter* do tipo *parameterType* que pode ser referenciado em outros tipos (p. ex. *methodType e changeType*).

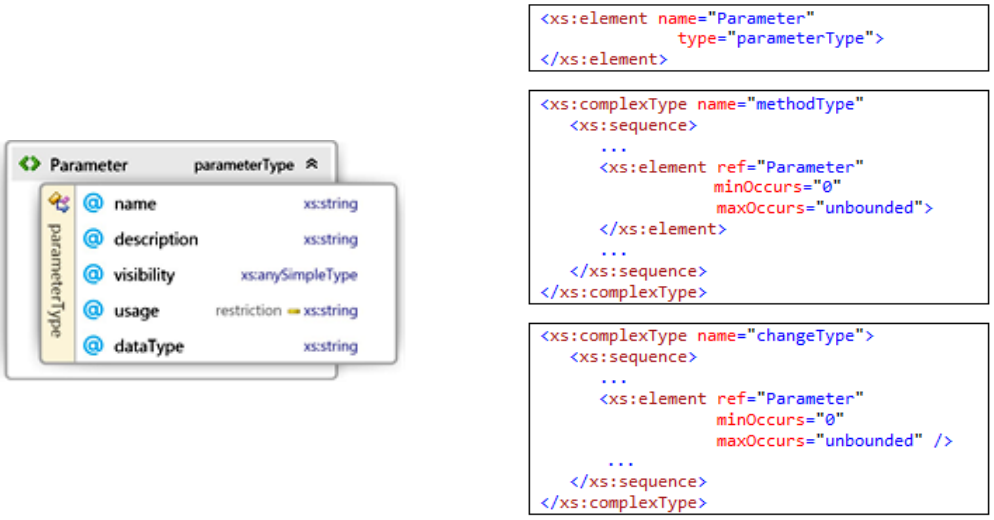


Figura 6.4 - Exemplo do padrão *Jardim do Éden*: *Parameter-parameterType.*

Além de padrões de projeto que promovem a reutilização, os próprios Esquemas XML têm características que facilitam a extensão da linguagem para outros domínios ou propósitos (*wildcards* e *Grupos de Substituição*), permitem o uso de herança, restrição de tipos e inclusão/importação de outros esquemas.

Na RBDL, as partes do esquema que são dependentes do domínio foram definidas em esquemas independentes, com *namespaces* específicos, que podem ser incluídos nos esquemas principais. O objetivo é tornar a linguagem extensível a outros domínios conforme detalhado na seção 6.1.4 deste capítulo.

A Figura 6.5 apresenta a hierarquia de esquemas utilizados para a RBDL no domínio de infraestrutura HLA. Nos níveis mais altos estão os esquemas que representam as grandes divisões da linguagem, por exemplo, *BENReport, BENInstantiation* e *BENDefinition*. Nos níveis intermediários tem-se os esquemas específicos e que são substituídos quando a linguagem é usada em um domínio diferente, por exemplo, *BENHLAType*, *BENSimulationType* e *BENDomainType*. No nível mais baixo estão os tipos básicos e gerais da linguagem *BENSubElement* e *BENTypes*.

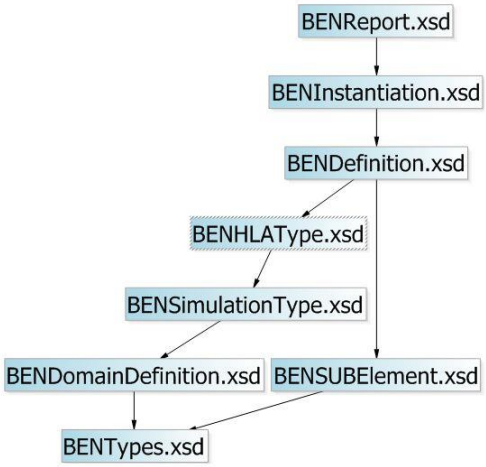


Figura 6.5 - Estrutura da RBDL.

6.1.2. Flexibilidade da Proposta da RBDL

Diferentes ferramentas comerciais e não comerciais permitem que a partir de XSDs sejam gerados automaticamente esqueletos de arquivos XML, bem como sejam validados arquivos XML já definidos. Desta forma, esqueletos gerados por meio da RDBL podem ser usados como guia na definição do benchmark, permitindo, também, que se possa naturalmente utilizar ferramentas de validação de sintaxe. Além disso, várias outras ferramentas de transformação podem ser utilizadas para a apresentação do benchmark e para auxiliar no processo de desenvolvimento do ferramental de geração e execução de experimentos.

A RBDL está dividida em duas partes com propósitos diferentes: (i) *especificação*, que é definida por meio de Esquemas XML (XSDs); e (ii) *publicação*, que é definida por meio de arquivos *EXtensible Stylesheet Language* (XSL). A Figura 6.6 apresenta a potencial aplicação da RBDL na geração de artefatos, mostrando a utilização de ferramentas de transformação.

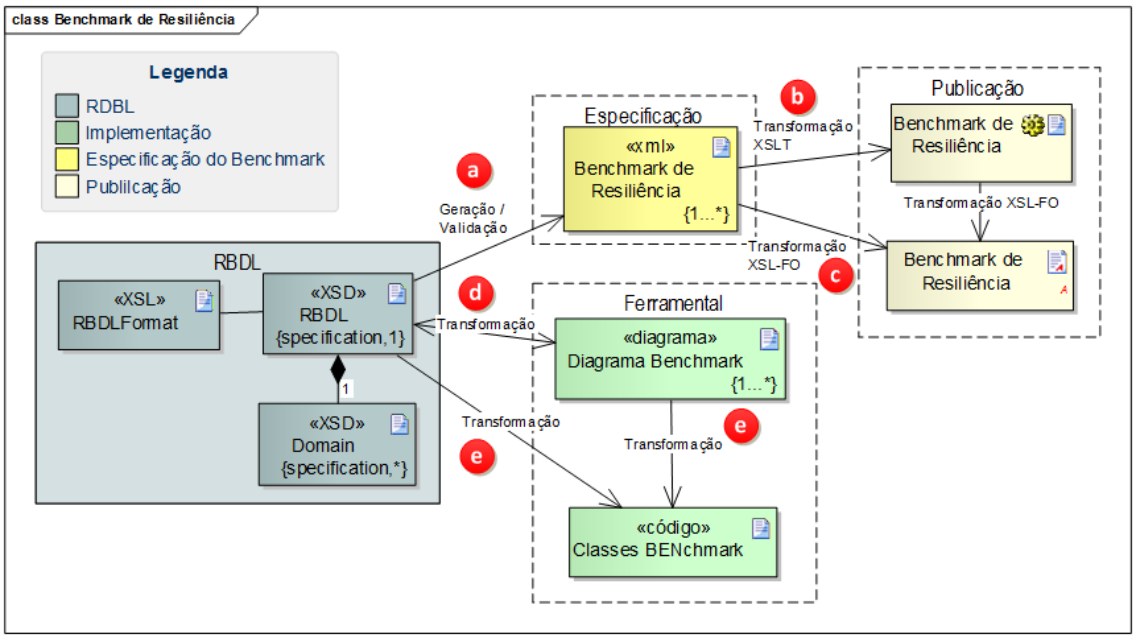


Figura 6.6 - RBDL - Uso de Ferramentas de Transformação.

A RBDL e as ferramentas de transformação permitem:

1) Gerar automaticamente arcabouços XML para a definição e instanciação de benchmarks de resiliência e validá-los (Figura 6.6, item (a));

2) Gerar páginas Web formatadas (Figura 6.6, item (b)) e/ou gerar documentos (Figura 6.6, item (c)) para publicação do benchmark a partir da ligação de um benchmark de resiliência expresso em XML a um arquivo de transformação XSL *Transformation* (XSLT);

3) Gerar Diagramas de Classe a partir da especificação XSD (e vice-versa) para a documentação da ferramenta de benchmark (Figura 6.6, item (d));

4) Gerar código em diferentes linguagens de programação a partir da especificação XSD ou a partir de diagramas de classe previamente gerados, acelerando o processo de desenvolvimento das ferramentas de benchmark (Figura 6.6, item (e)).

6.1.3. Descrição da RBDL

Como apresentado anteriormente, um benchmark pode ser logicamente dividido em: *Definição*, *Instanciação* e *Experimentos*. A RBDL, como linguagem

de representação, segue esta divisão e estabelece três *tags* principais, uma para cada parte do Benchmark: *BENchmarkDefinition*, representa os elementos fixos do benchmark; *BENchmarkInstantiation,* representa uma instância de benchmark concreto; e *BENchmarkReport,* expressa os resultados e a validação de um experimento de benchmark. As subseções a seguir descrevem cada uma dessas *tags*.

6.1.3.1. *BENchmarkDefinition:* Definição de Benchmarks usando a RBDL

A *definição* do benchmark é a parte fixa que especifica os elementos que não mudam entre várias instanciações de benchmarks concretos. A *tag BENchmarkDefinition* especifica os principais elementos do benchmark e pode ser dividida em seções lógicas representadas por *tags* aninhadas, conforme descrito a seguir:

1) A seção ***Descrição do Benchmark*** descreve, em linhas gerais, o Benchmark definido pelo documento RBDL. A *tag BENDefinitionHeader* permite que se defina o título do documento, seus autores, versão, identificador para referência (item (a) na Figura 6.7). A *tag BENchmarkDescription* permite indicar o tipo do benchmark (p. ex. resiliência, robustez, etc.), fazer a descrição textual do mesmo, descrever o domínio da aplicação e fazer referência a um documento de especificação quando pertinente (item (b) na Figura 6.7).

2) A seção ***Objetivos de Avaliação*** representa a lista de atributos relevantes que serão alvo de avaliação em benchmarks concretos. A *tag BenchmarkGoal* permite que se defina os atributos a serem avaliados: nome, definição adotada, tipo de atributo (dependabilidade, resiliência, robustez, etc.) (item (c) na Figura 6.8). Relativamente às métricas, a *tag Metric* permite que se represente: identificador, descrição, objetivo, fórmula a ser usada, unidade, forma de interpretação, etc. (item (d) na Figura 6.8).

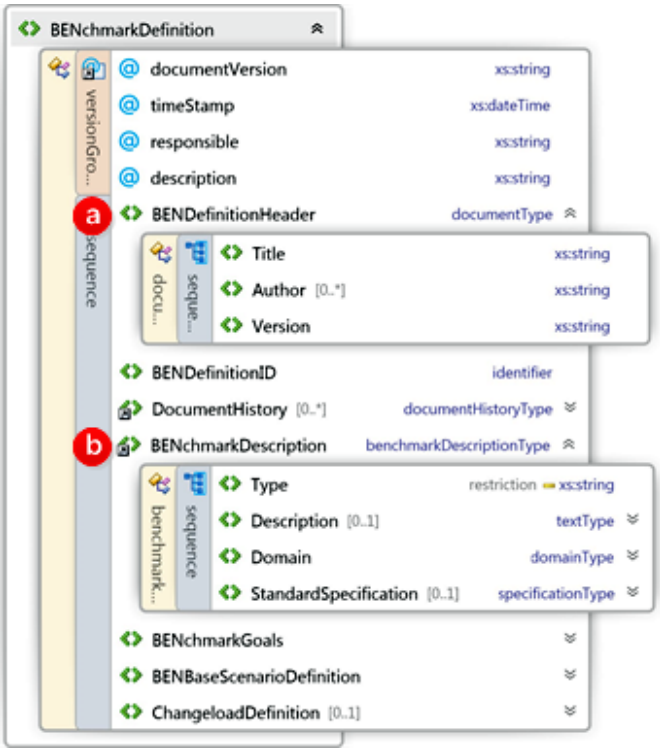


Figura 6.7 - RBDL - *BENchmarkDefinition* (Parte 1).

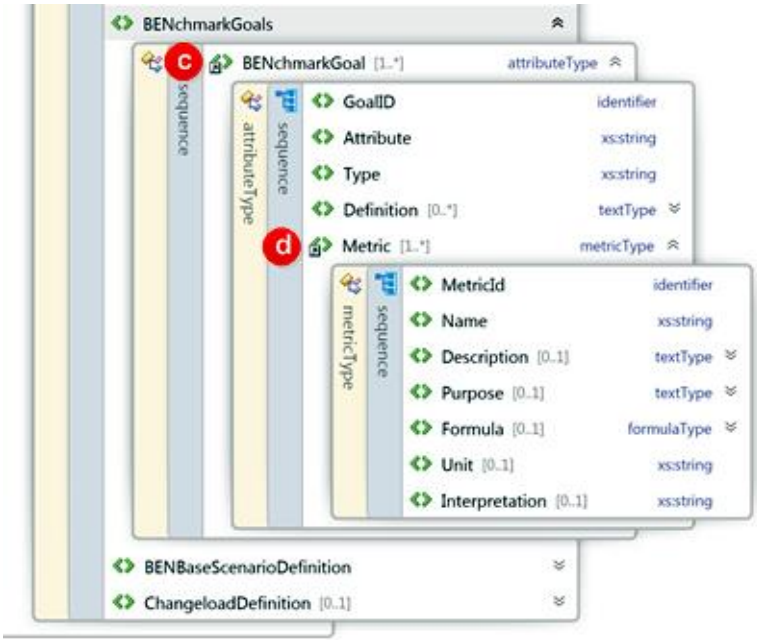


Figura 6.8 - RBDL - *BENchmarkDefinition* (Parte 2).

3) A seção ***Elementos do Cenário Base*** permite que se defina uma ou mais cargas de trabalho e condições operacionais, essa última dividida em configurações do domínio e configurações do sistema sob benchmark (SUB), conforme segue:

   i) A definição das *Cargas de trabalho* está dividida nas seguintes subseções: *DesignRequirements*, que representa as características de projeto da carga de trabalho; *TechnicalRequirements*, que representa as características tecnológicas da carga de trabalho, tais como linguagem de programação, versão, etc.; e *UsageRequirements*, que representa as características comportamentais da carga de trabalho, por exemplo, no domínio das infraestruturas de simulação, o comportamento dos modelos (item (e) Figura 6.9);

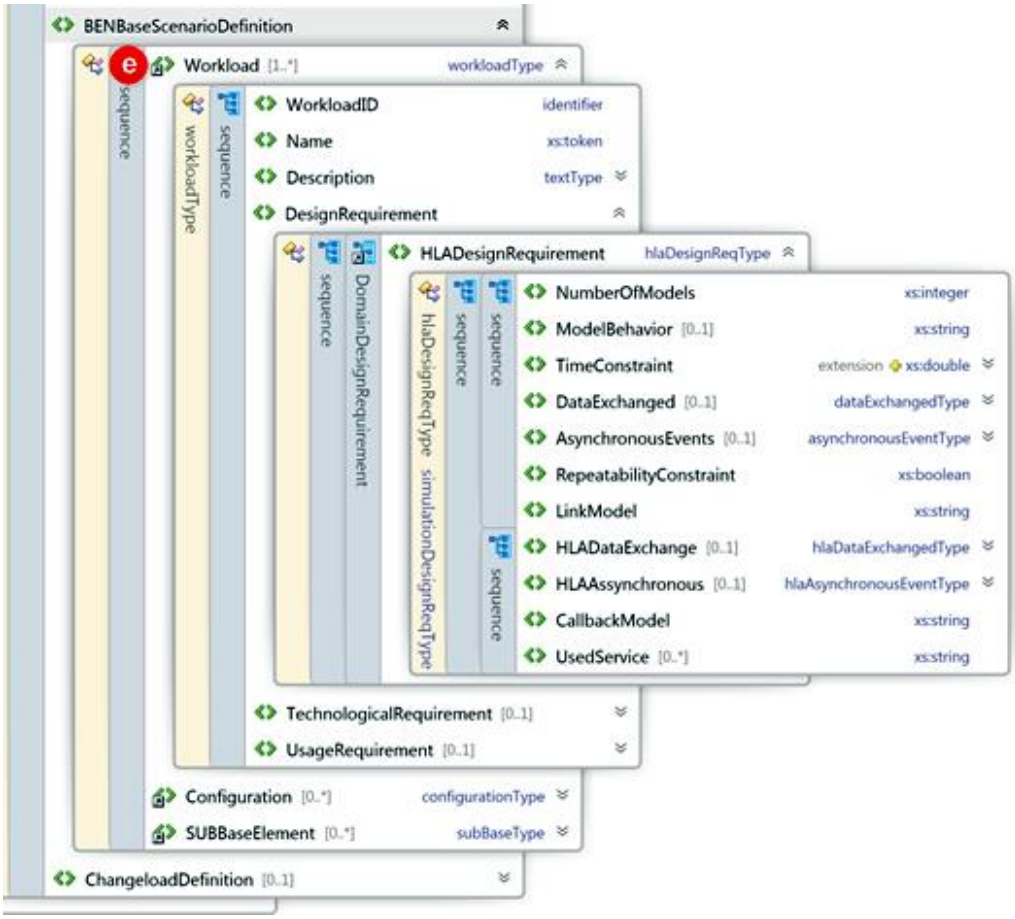


Figura 6.9 - RBDL - *BENchmarkDefinition* (Parte 3).

ii) *Configuração de domínio*: por meio desta subseção representam-se as configurações típicas dentro de cada domínio. O item (f) da Figura 6.10 apresenta a configuração HLA;

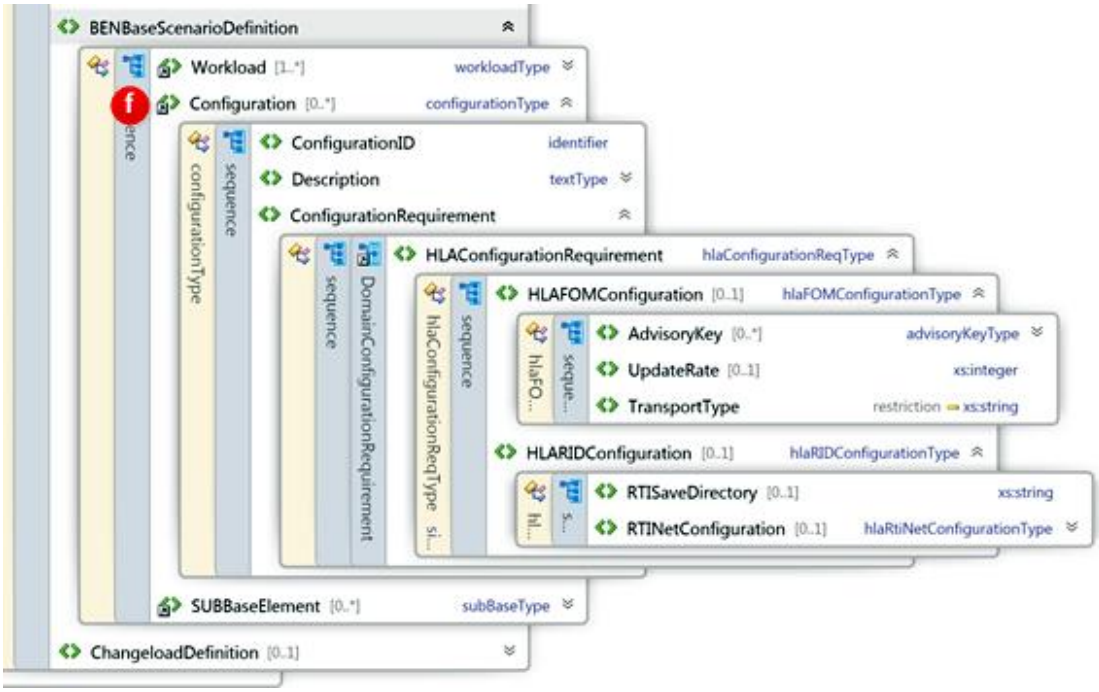


Figura 6.10 - RBDL - *BENchmarkDefinition* (Parte 4).

iii) *Configuração do sistema sob benchmark*: por meio desta subseção representam-se as arquiteturas base do SUB (item (g) Figura 6.11).

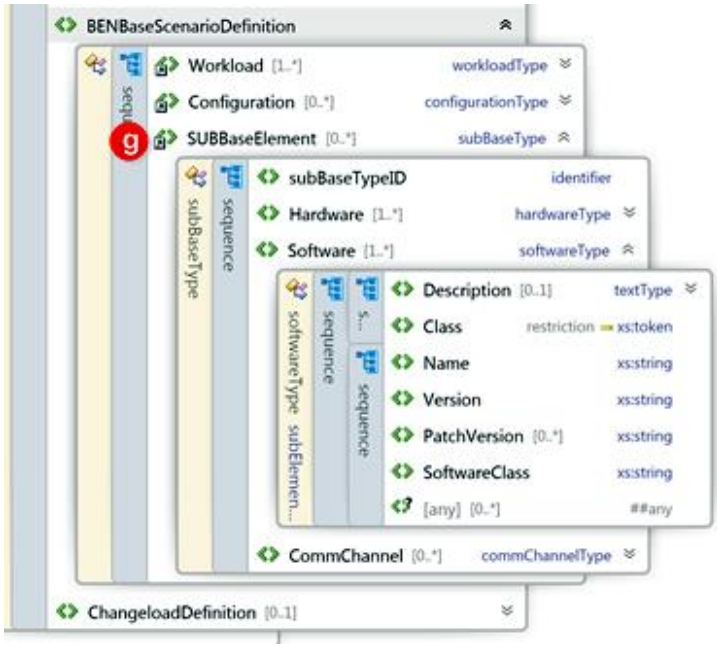


Figura 6.11 - RBDL - *BENchmarkDefinition* (Parte 5).

4) A seção ***Carga de Mudanças*** representa, para cada mudança, a parte fixa que a caracteriza: operador, tipo, conjunto de parâmetros e os métodos de geração, instanciação e recuperação associados a ela (item (h) Figura 6.12).

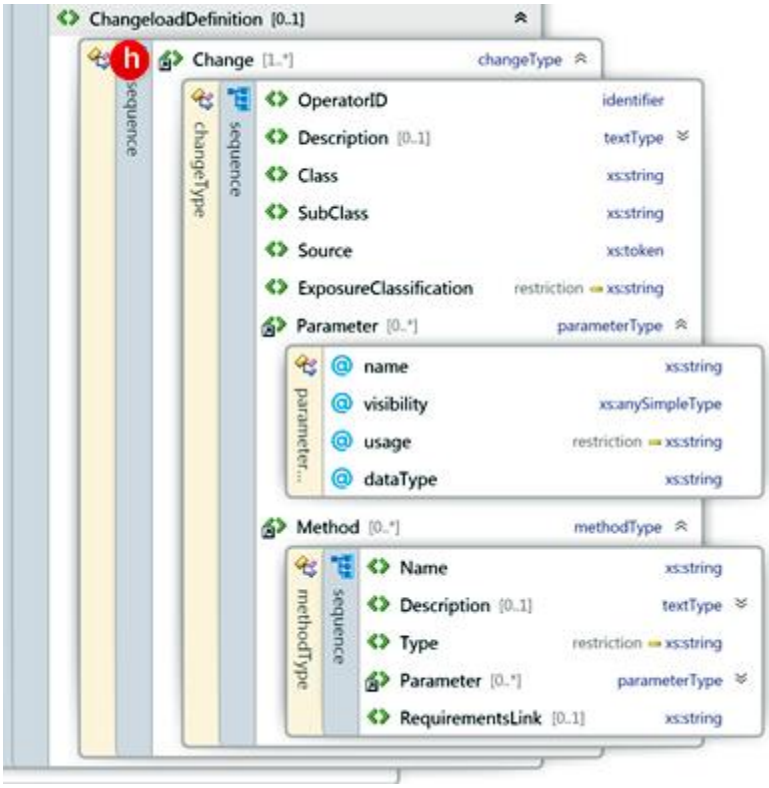


Figura 6.12 - RBDL - *BENchmarkDefinition* (Parte 6).

O documento RBDL de definição pode ser um documento único, separado e incluído nos documentos de instanciação de benchmarks concretos. Tal como acontece em documentos de especificação de benchmarks definidos por organizações de padronização como a TPC, sempre que novos elementos ou objetivos forem identificados, o documento de definição deverá ser atualizado para incorporar a evolução, permitindo que várias versões possam ser geradas.

Os documentos de definição tanto podem incorporar novos objetivos, novas cargas de trabalho, novas mudanças ou falhas, como excluir elementos obsoletos. A definição completa e atualizada deve estar representada pela última versão do mesmo. Importa salientar que os benchmarks concretos, bem

como os resultados obtidos na execução dos experimentos sempre estarão relacionados e farão referência a uma dada versão do documento de definição.

6.1.3.2. *BENchmarkInstantiation:* Instanciação de Benchmarks usando a RBDL

Ao longo da descrição da metodologia enfatizamos que diferentes benchmarks, com diferentes objetivos, poderiam ser derivados da definição completa do benchmark de resiliência. A *tag BENchmarkInstantiation* descreve esta instanciação de benchmarks e está dividida logicamente em cabeçalho e *Cenários de Instanciação*. Cada bloco é representado por *tags* específicas que estão descritas a seguir:

1) A seção ***Cabeçalho do Benchmark Instanciado*** descreve em linhas gerais o benchmark concreto a ser definido pelo documento RBDL: título do documento, autores, versão, identificador para futura referência. Todo benchmark concreto deriva de um benchmark previamente definido, assim nesta seção também é apresentada a referência ao documento de definição de benchmark utilizado (item (a) na Figura 6.13).

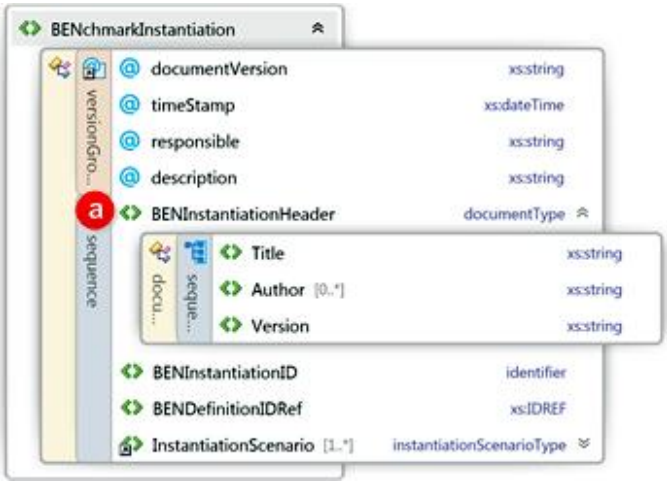


Figura 6.13 - RBDL - *BENchmarkInstantiation* (Parte 1).

2) A seção ***Cenários de Instanciação*** descreve os cenários que correspondem a um ou mais experimentos (dependendo do número de atributos e métricas que representem), mas sempre corresponderão a uma execução onde um *Cenário Base* é avaliado e, no caso dos benchmarks de

resiliência e dependabilidade, exposto a uma ou mais mudanças e/ou falhas. Esta seção está subdividida em:

i) *Descrição do cenário de instanciação*: por meio dessa subseção podem-se representar a identificação do cenário de instanciação, a sua descrição e, no caso de benchmark de resiliência, as métricas gerais de resiliência a serem utilizadas (item (b) na Figura 6.14);

ii) *Descrição de atributos e métricas*: neste ponto são referenciados os atributos a serem medidos e as métricas escolhidas (item (c) na Figura 6.14);

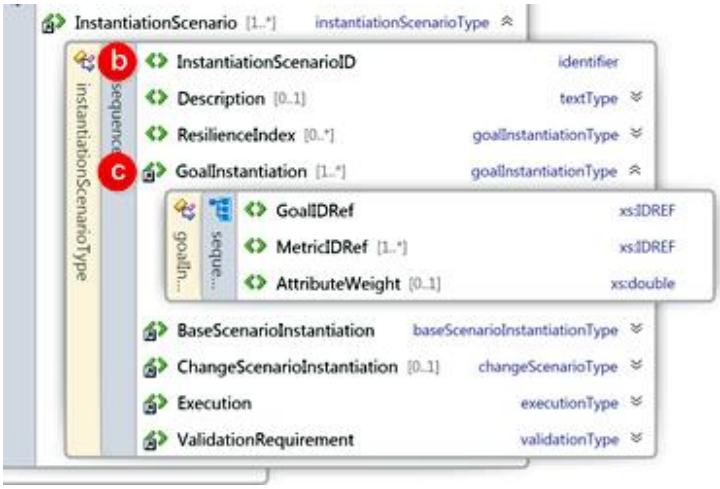


Figura 6.14 - RBDL - *BENchmarkInstantiation* (Parte 2).

iii) *Cenário Base*: por meio desta subseção pode-se representar a instanciação de um cenário base para o qual foram escolhidas a carga de trabalho e as condições operacionais - configuração e arquitetura - a serem usadas. Na definição da arquitetura escolhe-se qual SUB definido será usado, quais plataformas e como a carga de trabalho será instanciada nessas plataformas (*deployment*) (item (d) na Figura 6.15);

iv) *Cenário de Mudanças*: por meio desta subseção podem-se definir o tipo de cenário de mudança e uma ou mais mudanças a serem aplicadas ao *Cenário Base*, bem como os valores dos seus parâmetros e as regras de aplicação associadas (duração, gatilho, etc.) (item (e) na Figura 6.16);

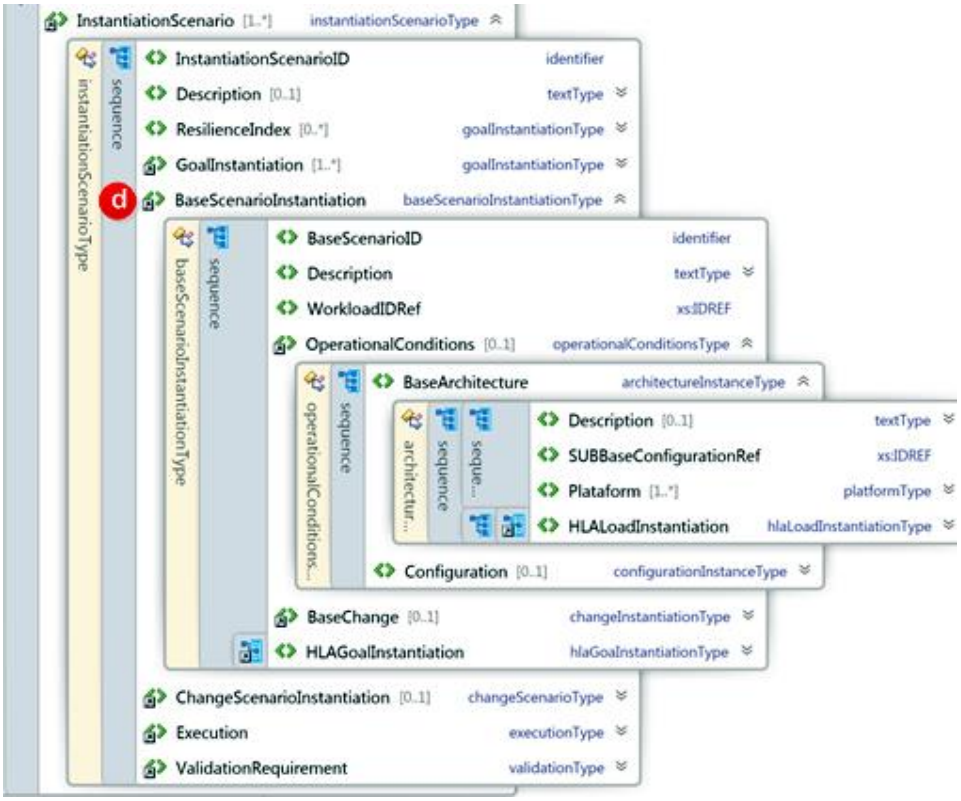


Figura 6.15 - RBDL - *BENchmarkInstantiation* (Parte 3).

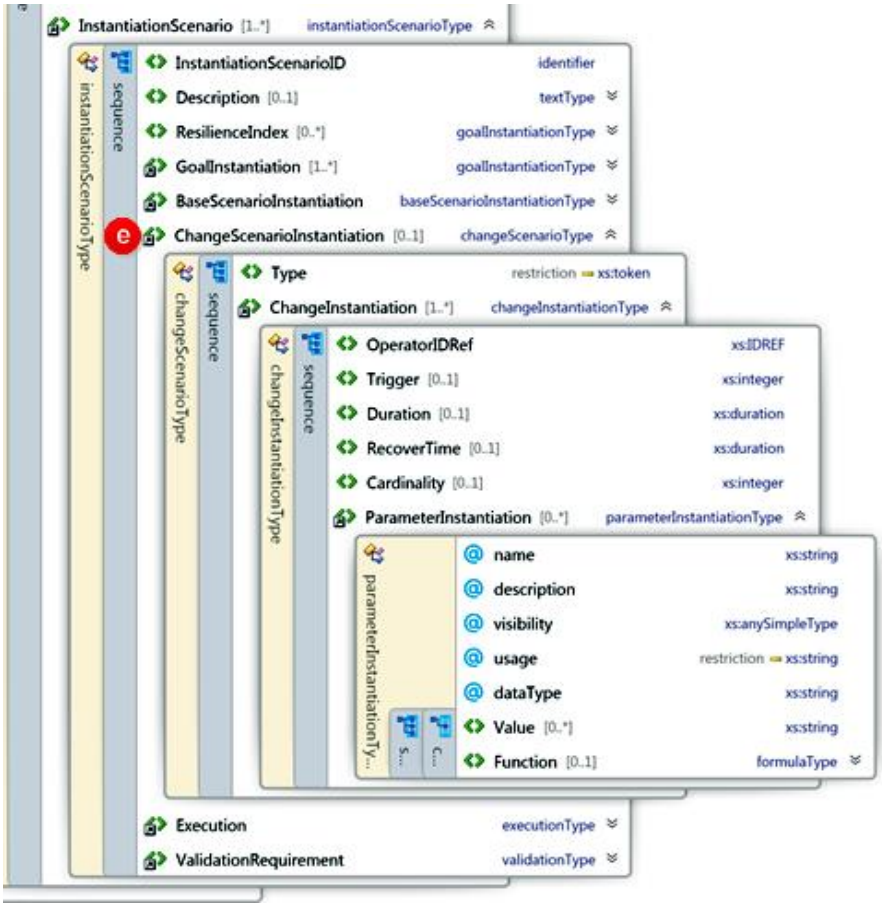


Figura 6.16 - RBDL - *BENchmarkInstantiation* (Parte 4).

v) *Execução*: por meio desta subseção podem-se definir as regras de execução do experimento naquele domínio (item (f) na Figura 6.17);

vi) *Validação*: usando esta subseção podem-se definir a duração prevista do experimento, o número de repetições e os níveis aceitáveis para a variabilidade dos resultados (*índice de repetibilidade*) (item (g) na Figura 6.17).

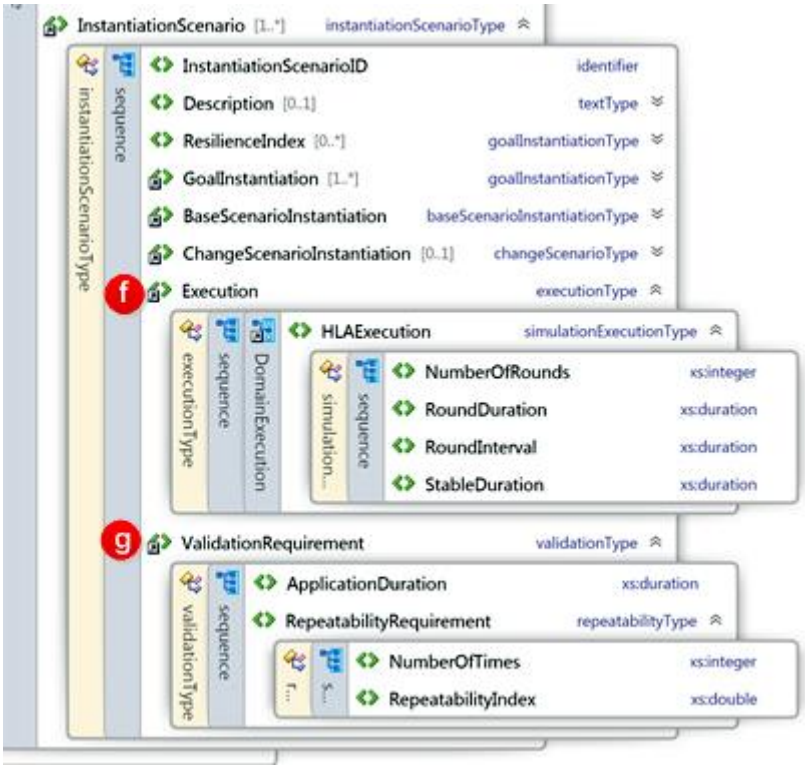


Figura 6.17 - RBDL - BENchmarkInstantiation (Parte 5).

6.1.3.3. *BENchmarkReport:* Relatórios de resultados usando a RBDL

Para que os resultados dos experimentos de um benchmark sejam aceitos é mandatório que se ateste que os mesmos foram realizados de acordo com a especificação e que se dê subsídios para que os resultados e condições de aplicação possam ser repetidos e auditados. Documentos de especificação de benchmarks de desempenho definidos pelo TPC (2010a, 2010b) e trabalhos de benchmark de dependabilidade (VEIRA, 2005), além de definirem os elementos próprios de cada tipo de benchmark, também definem o formato e conteúdo padrão dos relatórios de resultados que devem ser apresentados por quem

aplica os benchmarks. A especificação deve garantir que os resultados serão apresentados de forma semelhante, possibilitando a comparação dos mesmos.

A *tag BENchmarkReport* define o formato e o conteúdo a ser reportado relativamente aos resultados dos experimentos de benchmark. Normalmente os resultados são gerados em arquivos RBDL separados.

As seções definidas por esta *tag* são:

1) A seção ***Cabeçalho do Relatório de Resultados*** descreve o relatório e permite que se defina os dados gerais do documento, um identificador do conjunto de experimentos, uma referência à instância de benchmark que originou o experimento executado e a descrição geral do experimento (item (a) na Figura 6.16). Um relatório está sempre relacionado a um benchmark concreto.

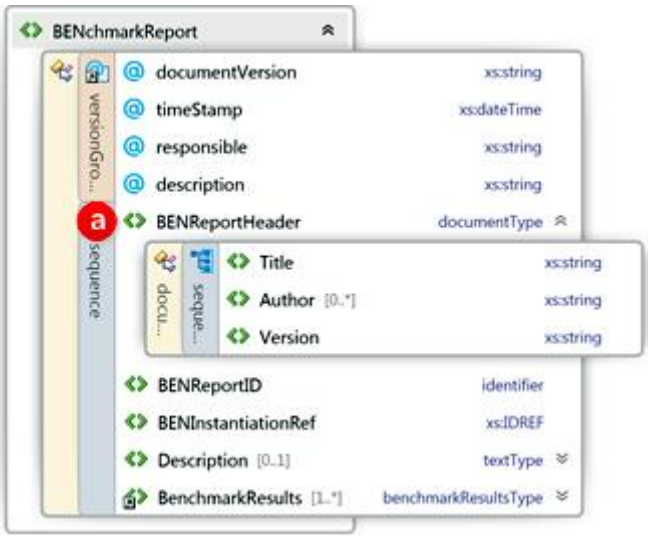


Figura 6.18 - RBDL - *BENchmarkReport* (Parte 1).

2) A seção ***Resultados do Benchmark*** permite que se descreva os resultados dos experimentos. Um relatório poderá apresentar um ou mais resultados, cada um deles relacionado a um cenário de instanciação. Por meio desta *tag* descreve-se a qual cenário de instanciação se refere o resultado, em qual período ele foi executado, qual a duração do experimento (em dias, horas, etc.), qual a arquitetura efetivamente utilizada

(item (b) na Figura 6.19) e os resultados obtidos na avaliação de cada produto, conforme segue:

i) *Resultados por Produto*: esta subseção permite que se apresente, para cada produto avaliado, as medidas obtidas relativamente aos atributos de interesse. Esta seção está dividida em:

- *Cabeçalho do produto*: por meio desta subseção descreve-se o produto e pode-se apresentar o resultado para o índice geral de resiliência (item (c) na Figura 6.19);

- *Objetivos*: por meio desta subseção apresenta-se a referência do atributo sendo avaliado e, para cada métrica, o resultado geral, o índice de resiliência medido (caso dos benchmarks de resiliência), o número de repetições daquele experimento e o *índice de repetibilidade* obtido para a métrica (item (d) na Figura 6.19). Esta subseção permite, ainda, representar as medidas obtidas no experimento: medida obtida na rodada de referência e as medidas obtidas em cada rodada de mudança (item (e) na Figura 6.19).

6.1.4. Generalização da RBDL

Vários elementos de um benchmark de resiliência podem ser especificados de forma genérica usando a linguagem de descrição, por exemplo, os atributos podem ser descritos com as mesmas propriedades para diferentes domínios ou tipos de benchmark, bem como as mudanças de uma carga de mudança, já que as mesmas podem ser genericamente caracterizadas por meio da definição de seu tipo, operador, elemento de origem e de seus parâmetros. No entanto, como mencionado anteriormente, os requisitos para a geração de carga de trabalho em um benchmark são completamente dependentes do domínio. Por exemplo, enquanto um simulador usando uma infraestrutura HLA tem federados como carga de trabalho, um benchmark de dependabilidade para sistemas OLTP usa como carga de trabalho transações (conforme especificado pelo TPC-C (VIEIRA, 2005; TPC, 2010a).

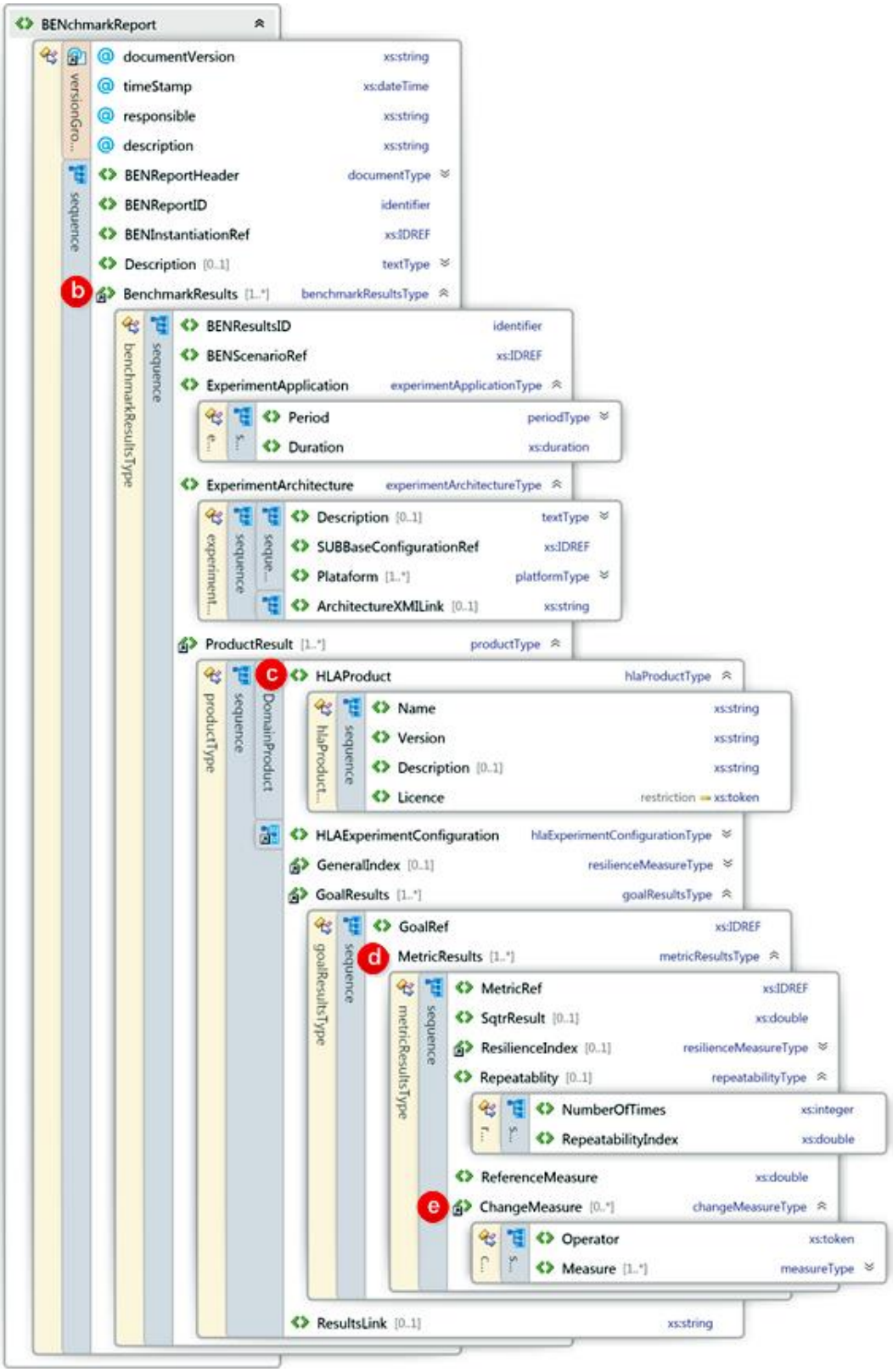


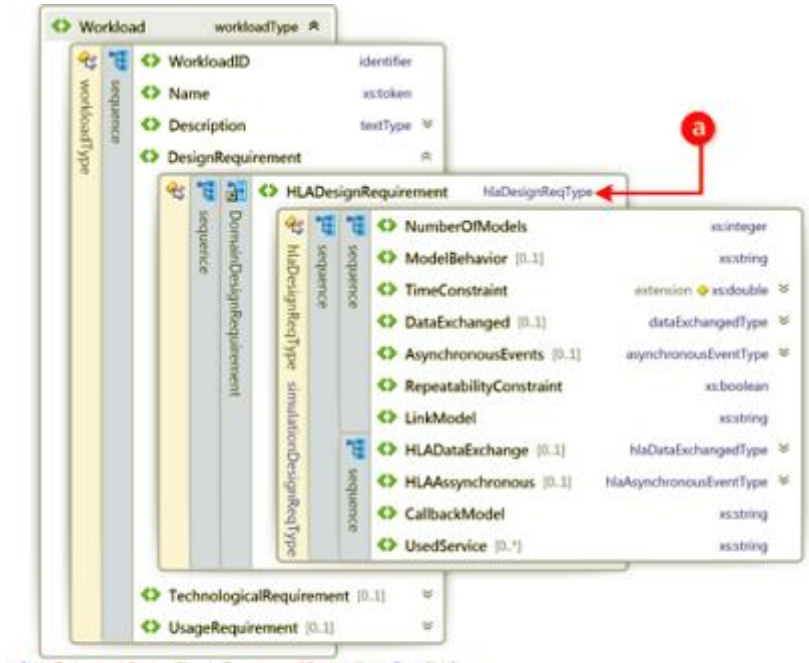
Figura 6.19 - RBDL - *BENchmarkReport* (Parte 2).

Visando generalizar a RBDL para que a mesma pudesse ser adaptada a diferentes domínios, foram analisadas as capacidades de extensão dos XSDs que são: (i) *inclusão de esquemas XML*, no qual os elementos XML genéricos são referenciados em pontos específicos da linguagem; (ii) *wildcards (xs:any)* que são pontos de extensão bem definidos que indicam a possibilidade do uso de *tags* não previamente especificadas; e (iii) *grupos de substituição* que definem elementos e tipos abstratos, que configuram pontos de extensão, e que podem ser substituídos e estendidos por elementos dos esquemas XML específicos (OBASANJO, 2004; WC3, 2004).

Na RBDL foram adotados *Grupos de Substituição* por ser uma característica intrínseca dos Esquemas XML e por utilizar conceitos análogos ao polimorfismo das linguagens Orientadas a Objetos (LOO). No entanto, para cada novo domínio, um esquema XML específico deve ser importado, ou seja, será necessária uma pequena alteração nos esquemas genéricos.

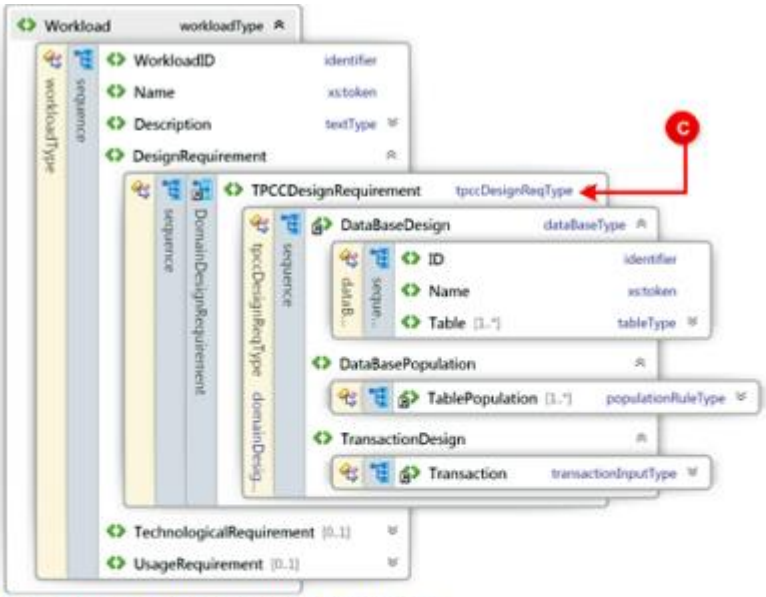
Para demonstrar a extensibilidade e adaptabilidade da RBDL foram definidos esquemas XML do domínio de infraestrutura de simuladores ((a) na Figura 6.20) e, também, esquemas XML que representam a especificação TPC-C ((c) na Figura 6.20). A linguagem de representação definida para a carga de trabalho TPC-C é apenas um exemplo simplificado que visa validar a abordagem escolhida. Usando as extensões propostas, definimos um exemplo da linguagem para dois domínios diferentes apenas mudando o nome do esquema de domínio importado ((b) e (d) na Figura 6.20).

A solução proposta especifica a RBDL como uma linguagem que fornece pontos de extensão que podem ser adaptados a diferentes domínios. Entretanto, a definição de esquemas específicos para um dado domínio pode ser um trabalho complexo que represente uma parte considerável da definição da linguagem de descrição.




```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
           targetNamespace="http://www.inpe.br/posgrad/Benchmark/2013"
           xmlns="http://www.inpe.br/posgrad/Benchmark/2013"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation=".\BENSUBElement.xsd" />
  <xs:include schemaLocation=".\HLA\BENHLAType.xsd" />
  <xs:element name="BENchmarkDefinition">
```

b




```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
           targetNamespace="http://www.inpe.br/posgrad/Benchmark/2013"
           xmlns="http://www.inpe.br/posgrad/Benchmark/2013"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation=".\BENSUBElement.xsd" />
  <xs:include schemaLocation=".\TPCC\TPCCType.xsd" />
  <xs:element name="BENchmarkDefinition">
```

d

Figura 6.20 - Extensão RBDL - HLA.

6.2. Ferramental para Benchmark de Resiliência Baseado em RBDL

O Sistema Gerenciador de Benchmark é composto por três subsistemas: (i) gerador de cargas de trabalho e mudanças, (ii) coordenador da execução do benchmark; e (iii) coletor de dados e analisador de resultados.

A Figura 6.21 apresenta uma arquitetura geral para a ferramenta de benchmark de resiliência no domínio de infraestruturas HLA.

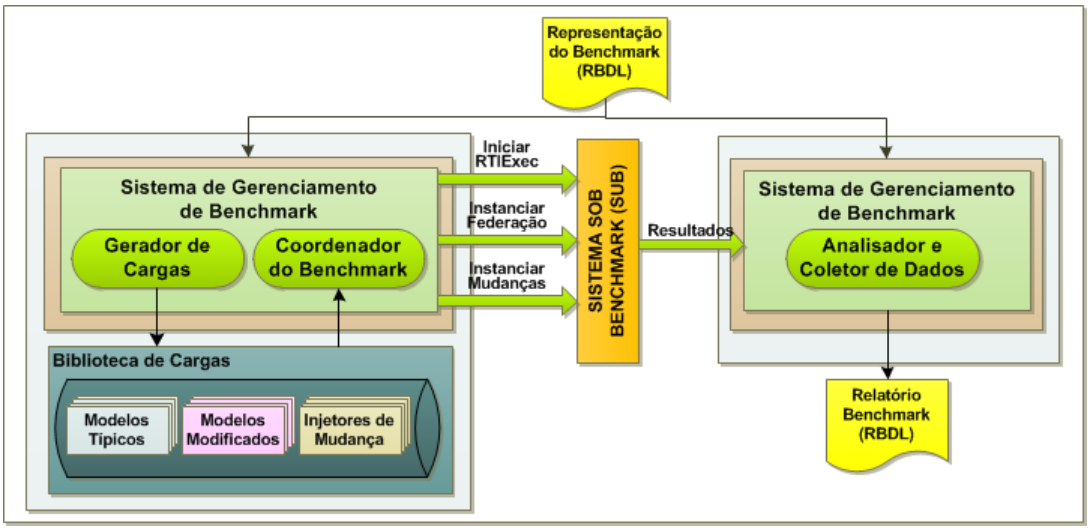


Figura 6.21 - Arquitetura Geral - SGB.

A ferramenta de benchmark de resiliência no domínio infraestruturas HLA deve ser capaz de gerar automaticamente os modelos de referência (federados típicos) e os modelos modificados de acordo com as mudanças definidas no arquivo RBDL; e de executar e instanciar as federações de referência, as federações compostas por modelos modificados ou as mudanças específicas.

Os elementos gerados pela ferramenta são armazenados em uma biblioteca de cargas composta por: federações de referência, federações modificadas e cargas específicas de mudança. Os elementos da Biblioteca de Cargas são diretamente usados nos experimentos de benchmark.

Nas subseções seguintes é apresentado o uso da linguagem RBDL na construção da ferramenta de benchmark e delineado o projeto da mesma.

6.2.1. Mapeamento da RBDL para Geração da Ferramenta de Benchmark

A estrutura da RBDL, representada por Esquemas XML, permite que se utilize softwares disponíveis no mercado - comerciais e de código aberto - para facilitar e agilizar o processo de desenvolvimento do ferramental de benchmark.

Os Esquemas XML apresentam correspondências com aspectos de linguagens de programação Orientadas a Objetos: tipos simples e complexos, herança, encapsulamento, polimorfismo, etc. Assim, a tradução ou mapeamento de um Esquema XML para as linguagens de programação OO ou para diagramas de classes da UML, e vice-versa, não é apenas viável, mas também natural. Há disponíveis no mercado diversas ferramentas de transformação que permitem mapeamentos automáticos. Dependendo do objetivo principal da ferramenta, é possível partir de arquivos XSD para gerar diagramas UML e/ou *código fonte*, ou fazer o caminho inverso, partir de diagramas UML para gerar XSDs.

Como neste trabalho o objetivo é estruturar e representar os elementos do benchmark de resiliência e do domínio de infraestruturas de simulação utilizando a linguagem semiformal RBDL, foi mais natural a criação direta dos Esquemas XSDs e posterior mapeamento dos mesmos para os diagramas de classe UML e para a linguagem de programação na qual seriam desenvolvidas as ferramentas de benchmark. No entanto, poderíamos ter seguido o caminho inverso.

Vale salientar que a flexibilidade da estrutura da RBDL, que permite a adaptação e extensão a diferentes domínios, pode ser facilmente expandida para a implementação da ferramenta de benchmark uma vez que as *classes* em linguagem de programação são geradas automaticamente a partir da mesma.

6.2.2. Projeto da Ferramenta de Benchmark

A partir dos Esquemas RBDL, e seguindo a subdivisão lógica dos mesmos, foram gerados três diagramas de classes. O primeiro representando a definição do benchmark, o segundo representando a instanciação e execução do

benchmark e o terceiro representando a análise de resultados. Como a transformação foi direta, foram geradas tanto *classes* que fazem parte do *framework* do benchmark, quanto *classes* do domínio do problema, divididas em diferentes pacotes de software.

Com o objetivo de trazer para a ferramenta de benchmark de resiliência a flexibilidade da linguagem na instanciação de benchmarks concretos, o ferramental de benchmark foi concebido levando em consideração a incorporação de novos tipos de mudanças, novas características na carga de trabalho ou novos objetivos tanto na geração/execução do benchmark, quanto na análise dos resultados. Para tanto, o projeto dos módulos da ferramenta utiliza o padrão de projeto *Strategy* no qual a incorporação de novos objetivos, características da carga de trabalho ou mudanças é feito por meio da criação de *classes* que implementem os métodos de *geração* e *execução* dos mesmos.

A Figura 6.22 apresenta a arquitetura do módulo de Geração/Execução do Benchmark.

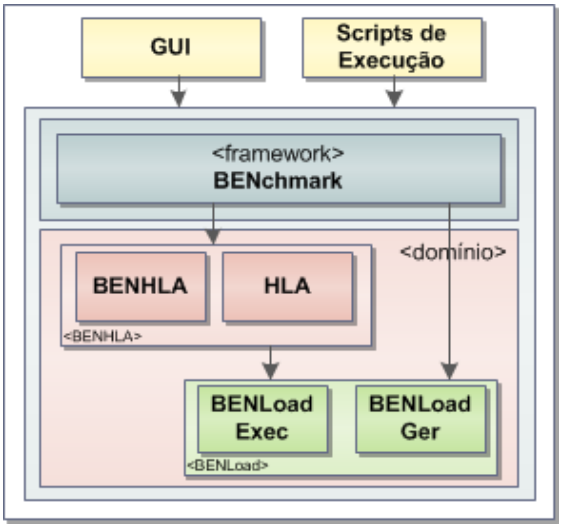


Figura 6.22 - Arquitetura da Ferramenta de Geração/Execução de Benchmark.

Os pacotes que compõe esse módulo da ferramenta de benchmark estão descritos a seguir:

- BENchmark: incorpora os elementos específicos do domínio de benchmark. Todos os elementos do benchmark herdam das classes raízes

BENElement, *BENDefinitionElement* e *BENInstantiationElement*, nas quais os elementos que compõem um benchmark devem sobrescrever métodos que lhes permitam carregar o valor dos seus atributos a partir de arquivos RBDL, e dependendo do tipo (definição ou instanciação), criar o elemento, gerar a carga específica ou se executar. A Figura 6.23 (item (a)) apresenta um exemplo relativamente a duas classes do modelo: *CBENChange* e *CBENChangeInstantiation*.

- BENLoad: este pacote é composto por classes que implementam tanto a geração (*BENLoadGer*) quanto a execução (*BENLoadExec*) de cargas de trabalho ou mudança. À medida que uma nova mudança é incorporada à carga de mudanças, são adicionadas *classes* que estendem as *classes BENChangeGenerationMethod* e *BENExecutionMethod*, sobrescrevendo os respectivos serviços, lembrando que as mudanças na RBDL referenciam os seus métodos de geração e execução (item (b) Figura 6.23).

- BENHLA: este pacote é composto pelas classes de benchmark específicas do domínio do problema. Estas classes definem requisitos da carga de trabalho, requisitos de configuração, requisitos de instanciação (*deployment*), etc. As classes do domínio HLA derivam das classes que configuram pontos de extensão no *framework*. A Figura 6.24 (item (a)) apresenta a fronteira entre o *framework* BENchmark e as classes do pacote BENHLA.

- HLA: este pacote é composto pelas classes de domínio HLA que efetivamente geram os federados e federações.

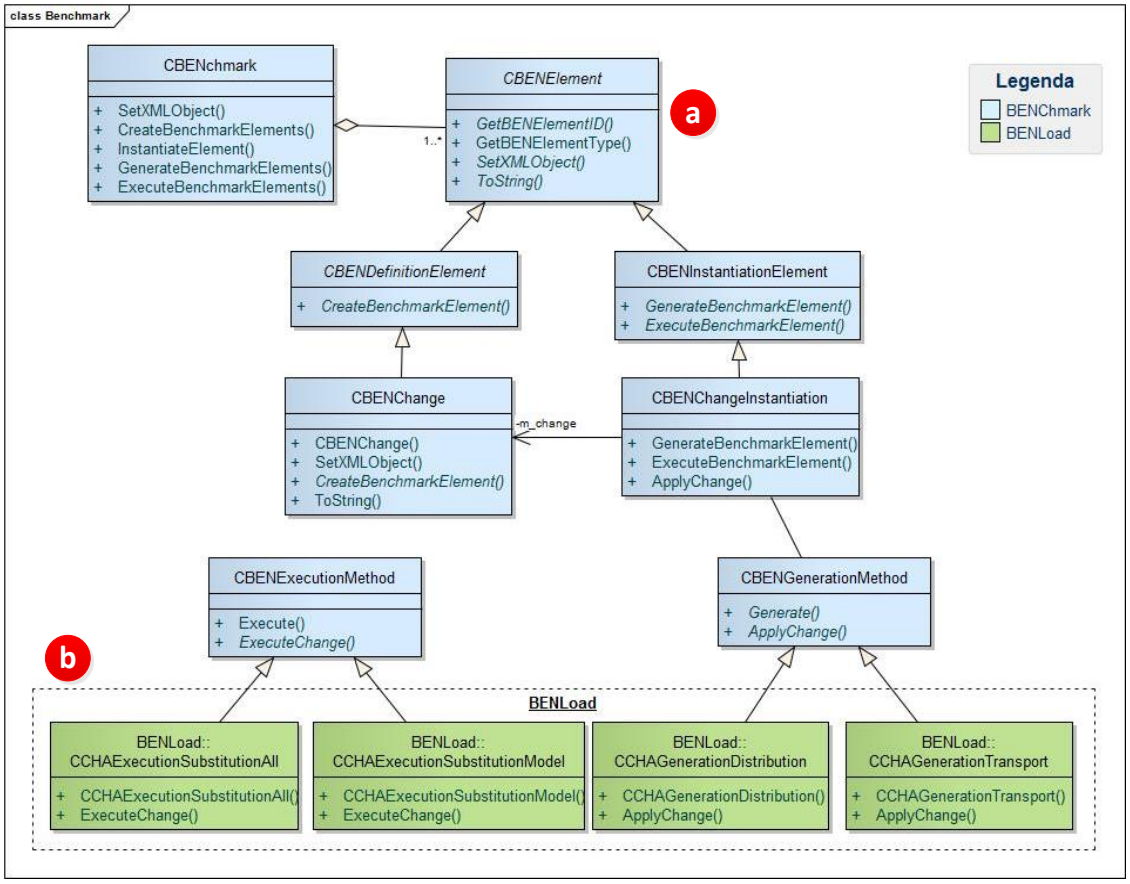


Figura 6.23 - Estrutura de Classes - Ferramenta Benchmark.

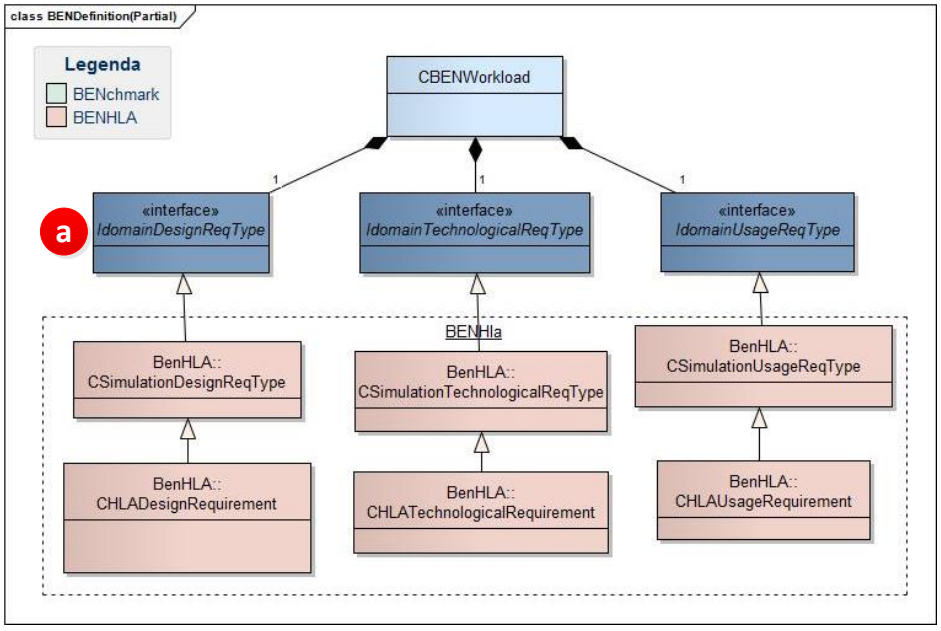


Figura 6.24 - Estrutura de Classes - Fronteira com as classes de Domínio

Relativamente à implementação da ferramenta, os pacotes se traduziram em bibliotecas que podem ser modificadas e atualizadas. A biblioteca de geração de cargas de mudança pode crescer quando uma nova mudança for implementada e as bibliotecas de domínio podem ser completamente substituídas. A Figura 6.25 apresenta a estrutura da ferramenta por meio do Diagrama de Componentes UML.

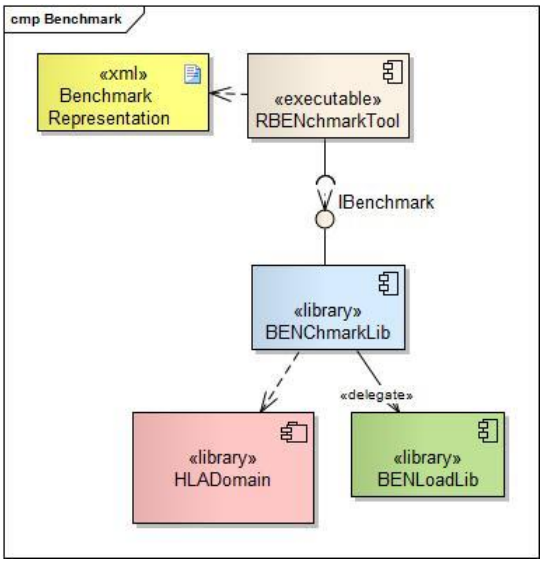


Figura 6.25 - Componentes da Ferramenta de Geração/Execução de Benchmark.

Importa salientar que a proposta apresentada nessa seção, seja no que tange a arquitetura, seja no que tange ao *framework*, foi também estendida para a Ferramenta de Análise de Resultados.

6.3. Considerações Finais

A descrição padronizada de um benchmark é essencial dos pontos de vista de formalização, comunicação e verificação. A RBDL é uma alternativa para especificação de benchmark de resiliência e pode servir como um guia para a definição de benchmarks, bem como ajudar na padronização, normalização e disseminação dos benchmarks especificados, podendo ser utilizada para representar diferentes tipos de benchmark e adaptada a diferentes domínios. Arquivos RBDL que representem benchmarks podem ser diretamente processados e utilizados pelo ferramental de benchmark para a geração de

cargas de trabalho, mudanças e na execução dos experimentos (AZEVEDO et al., 2014).

A utilização de Esquemas XML para a especificação da RBDL possibilita o uso de um conjunto de ferramentas disponíveis no mercado para a geração automática do arcabouço de especificação do benchmark, geração automática de documentos, tradução do *framework* especificado por meio da linguagem para diagramas UML e código em linguagens OO, etc. Essa abordagem mostrou-se vantajosa uma vez que facilita os processos de especificação e publicação do benchmark e acelera o desenvolvimento das ferramentas.

7 INSTANCIAÇÃO DE BENCHMARKS PARA INFRAESTRUTURAS HLA

Este capítulo demonstra a aplicação da metodologia proposta e dos elementos especificados nos capítulos anteriores na instanciação de dois tipos de benchmarks concretos - Resiliência e Robustez – que são apresentados por meio de três estudos de caso.

O primeiro estudo de caso apresenta um benchmark de resiliência que avalia atributos e métricas de dependabilidade associadas ao desempenho de uma implementação HLA e exercita mudanças da classe *escala* em cenários de mudança primários. O segundo estudo de caso é uma extensão do primeiro e tem como objetivo validar o uso da metodologia na definição e representação de cenários de mudança compostos. Já o terceiro estudo de caso demonstra o uso da metodologia e da RBDL na especificação de benchmarks que avaliam a *robustez* das implementações HLA quando submetidas a falhas de interface.

Os estudos de caso apresentados exploram diferentes conjuntos de mudanças, com diferentes objetivos de avaliação, conforme esquematizado na Figura 7.1.

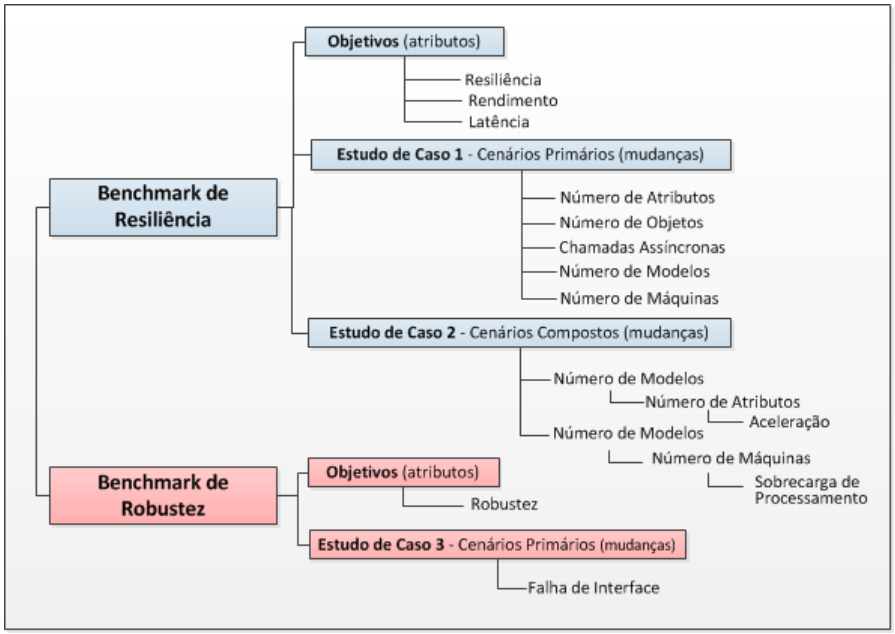


Figura 7.1 - Benchmarks - Estudos de Caso.

As próximas seções apresentam uma visão geral das implementações HLA avaliadas, a configuração do laboratório de experimentos e detalham os estudos de caso.

Os estudos de caso são apresentados usando a seguinte sequência: (i) definição do *Cenário de Instanciação*; (ii) definição da validação do experimento de benchmark; (iii) representação do benchmark; e (iv) resultados dos experimentos e análise dos resultados.

No final do capítulo são discutidas as lições aprendidas com o uso da metodologia, da RBDL, e com o desenvolvimento e uso do ferramental de benchmark.

7.1. Implementações HLA Avaliadas

Os estudos de caso apresentados neste capítulo avaliam três implementações RTI HLA, sendo duas implementações de *código aberto* e uma implementação comercial.

Em relação à escolha das implementações *de código aberto*, foram avaliadas aquelas que abrangiam o maior número de alternativas no que tange a versões da norma HLA, linguagens da API e sistemas operacionais. Foram, também, levados em consideração na escolha dessas RTIs fatores como a atualização de versões, a disponibilidade de documentação e o suporte dado à implementação. Já a RTI HLA comercial avaliada é amplamente utilizada no mercado e uma licença acadêmica encontra-se disponível no INPE.

As RTIs HLA estão caracterizadas segundo: (i) o *sistema operacional* nos quais as implementações RTIs podem ser executadas, incluindo a disponibilidade de versões da biblioteca RTI (*RTIlib*); (ii) a *arquitetura*, que pode ser centralizada ou descentralizada; (iii) o *modelo de callback* que representa a forma como a biblioteca RTI (*RTILib*) ligada aos federados trata o recebimento de chamadas originadas na RTI (*callback*) (ver Tabela 2.2); (iv) as *linguagens de programação* disponíveis para a API HLA; (v) as *versões do padrão* HLA

implementadas pela RTI; e (vi) o *canal de comunicação* que indica o protocolo de comunicação ou *middleware* utilizado pela implementação.

A Tabela 7.1 apresenta as características principais das implementações RTI avaliadas pelos benchmarks aplicados. Vale salientar que a licença da RTI comercial (RTI_COM) disponível no INPE só permite a execução de 10 federados, o que limitou o alcance dos experimentos realizados.

Tabela 7.1 - Características das RTIs Avaliadas.

RTIs HLA						
	S.O.	**Arquitetura**	**Modelo de Callback**	**API**	**RTI**	**Canal de Comunicação**
RTI_OP1	Linux, Mac OS, Windows	Centralizada	Multi Processo	C++ Java	1.3 1516	TCP - UDP
RTI_OP2	Linux, Mac OS, Windows	Descentralizada	*Multithread*	C++ Java	1.3 1516e	*Jgroups (*UDP Multicast)
RTI_COM	Linux, Mac OS Windows,	Centralizada	*Multithread*	C++ Java	1.3, 1516, 1516e	TCP, UDP, UDP Multicast

7.2. Configuração do Experimento

A Figura 7.2 mostra a arquitetura física do laboratório utilizado para a execução dos estudos de caso apresentados e a Tabela 7.2 apresenta a configuração dos equipamentos do laboratório.

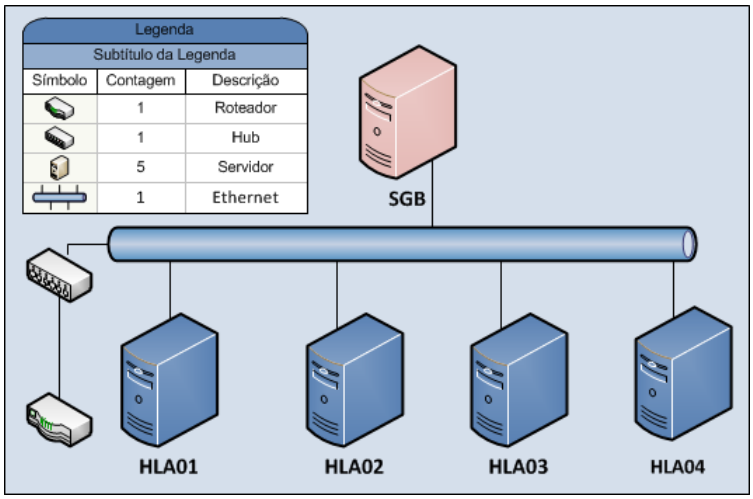


Figura 7.2 - Arquitetura do Laboratório.

Tabela 7.2 - Condições Operacionais - SUB.

Condições Operacionais - SUB					
Nome	**SGB**	**HLA01**	**HLA02**	**HLA03**	**HLA04**
Hardware					
Processador	Intel Core 1.87GHz	Intel Core2 Duo	Intel Core I5	Intel Core I5 3.2 GHz	Intel Core I5
Memória	1 Gb	3 GB	3 GB	4 GB	2 GB
Disco rígido	50 GB	250 GB	200 GB	320 GB	200 GB
Categoria	PC	PC	PC	PC	PC
Software					
Classe	Sistema Operacional	Sistema Operacional	Sistema Operacional	Sistema Operacional	Sistema Operacional
Nome	Windows XP	Windows XP	Windows XP	Windows 7	Windows XP
Atualização	Service Pack 3	Service Pack 3	Service Pack 3	Service Pack 1	Service Pack 3

Os estudos de caso apresentados utilizaram o equipamento HLA01 como referência, representando a configuração do SUB centralizado. A rede utilizada é uma Ethernet LAN de 100 Mbps.

7.3. Benchmark de Resiliência: Desempenho

O benchmark de resiliência é apresentado por meio de dois estudos de caso que têm como foco principal avaliar a resiliência do desempenho de RTIs HLA. O primeiro estudo de caso demonstra o uso da metodologia em cenários primários e avalia as três implementações HLA apresentadas, e o segundo estudo de caso demonstra a aplicabilidade da metodologia na avaliação de cenários compostos.

Vale salientar que o Estudo de Caso 2 só avaliou duas das implementações HLA (RTI_OP1 e RTI_COM) dada a instabilidade que a RTI_OP2 apresentou na avaliação do atributo Rendimento durante a execução de um dos seus cenários de instanciação.

7.3.1. Estudo de Caso 1: Cenários Primários

Este benchmark avalia os aspectos da resiliência do desempenho de implementações HLA na presença de **mudanças de escala** que representam o uso da mesma infraestrutura de simulação em diferentes missões, nas quais

alterações no número de modelos e no volume de dados intercambiados entre os modelos são usuais. Também foi avaliada a **capacidade de distribuição** da simulação e a inserção de **chamadas assíncronas** que caracterizam, por exemplo, alterações no tipo de simulador.

7.3.1.1. Cenário de Instanciação

A seguir é apresentada a execução dos passos previstos para instanciação de benchmarks concretos, conforme apresentado na Seção 4.3.1.

1) Objetivos do Benchmark

Os atributos a serem medidos pelo benchmark são: Latência e Rendimento.

Para o atributo Latência, as seguintes métricas são usadas: (i) média da latência direta de atualização de atributos (*LUpdAvg*, Tabela 5.2) e média da latência indireta (*round trip*) de atualização de atributos (*LUpdAvgRT*, Tabela B.3). Para o atributo Rendimento é usada a métrica frequência da taxa de atualização de atributos (*TFreq* - Tabela B.4). Já para medir a *Resiliência*, são considerados o índice de estabilidade relativa (*RRindex*, Tabela 5.3) e o índice de perda de serviço (*RLIndex* e *RLIndexMin*- Tabela B.2).

2) Cenário Base

A carga de trabalho usada representa o simulador de um satélite experimental (a definição da carga de um simulador experimental é apresentada no Apêndice B). Como a classe de mudança *Escala* exercita aspectos da variabilidade das cargas de trabalho, nós concebemos uma carga de trabalho que simula um satélite de pequeno porte, em um simulador configurado em software, do tipo FES (ver Tabela 2.1), que permitiu exercitar as três implementações HLA, incluindo a RTI comercial que tem limitações no número de federados. A Tabela 7.3 apresenta os elementos da carga de trabalho considerada nos experimentos e também as configurações da RTI HLA.

Durante a execução do experimento, os federados que compõem a carga de trabalho foram gerados automaticamente pela ferramenta de benchmark

usando os valores do arquivo de representação do benchmark (RBDL) e *templates* específicos (ver Apêndice A). Na avaliação do atributo Rendimento não foi definido o elemento de simulação "*frequência do passo de simulação*", considerando neste caso uma simulação tão rápida quanto possível (*as-fast-as-possible*), já que o objetivo foi avaliar a frequência máxima de intercâmbio de dados entre os federados.

Tabela 7.3 - Estudo de Caso 1: Carga de Trabalho e Configurações HLA.

Carga de Trabalho	
Elementos	**Valores**
Requisitos de Projeto	
Número de modelos	6
Dados intercambiados	32 atributos
Tipo dos dados intercambiados	double
Serviços assíncronos	nenhum
Parâmetros dos serviços assíncronos	NA
Tipo de dados dos parâmetros	NA
Ligação entre modelos	pares
Repetibilidade	NA
Frequência do passo de simulação	1 Hz
Comportamento do modelo (carga)	nenhum
Classes de Objetos – tipos	1
Classes de Interação – tipos	NA
Classes de Interação – número	NA
Classes de Objetos – instâncias	1
Modelo de *callback*	HLA_EVOKED
Serviços HLA	nenhum
Linguagem de programação	C++
Versão HLA	RTI13
Aspectos de Configuração HLA	
Chaves de configuração (advisoryKey)	NA
Taxa de atualização	NA
Tipo de transporte	HLAbestEffort
Configuração da rede – *bundling*	não
Configuração da rede – *multicast*	não

Em todos os experimentos os federados foram ligados aos pares, havendo sempre um modelo emissor de dados e outro receptor. No caso do atributo Rendimento, o desempenho de todos os federados foi avaliado, já para o atributo Latência levou-se em consideração apenas o tempo entre a

atualização dos dados no federado emissor e a reflexão (recepção) no federado receptor.

Relativamente às condições operacionais de referência, o experimento foi executado com as configurações apresentadas na Tabela 7.2.

3) Cenário de Mudanças

Este estudo de caso considera mudanças da classe *Requisito* e da subclasse *Escala*. As mudanças selecionadas levaram em consideração a variação de escala advinda do uso da infraestrutura de simulação em diferentes missões. Para tanto, exercitamos alterações na escala do volume de dados intercambiados entre os modelos (*FAttrib* e *FObject*) e do número de modelos (*FNModel*). Também foram consideradas mudanças para permitir chamadas assíncronas (p. ex. simuladores operacionais) (*FInter*) e a distribuição da simulação (*HNMach*), que pode ser necessária em simuladores de maior complexidade. A Tabela 7.4 apresenta as mudanças escolhidas (operadores de mudança).

Tabela 7.4 - Estudo de Caso 1: Operadores de Mudança.

Classe: Requisitos [Subclasse: Escala]		
#	**Operador**	**Descrição**
C11	FAttrib	Aumento no volume de dados intercambiados entre federados (publicação e subscrição de dados) - número de atributos.
C12	FObject	Aumento no volume de dados intercambiados entre federados (publicação e subscrição de dados) - tipos de classes de objeto.
C13	FInter	Aumento no número de chamadas a serviços de outros Federados (chamadas assíncronas).
C14	FNModel	Aumento no número de modelos que compõem um simulador.
C15	HNMach	Mudança no número de máquinas na simulação (aumento, diminuição).

Os parâmetros de cada mudança escolhida assumiram valores crescentes que estressam cada uma das fontes de mudanças. Na definição dos valores dos parâmetros, para a maioria dos casos, foi utilizada a potência de dois tendo como limites máximos valores que excedem aqueles que seriam esperados na simulação de um satélite de sensoriamento remoto (Tabela B.11) ou restrições operacionais (p. ex. número de máquinas disponíveis). A Tabela 7.5 apresenta os valores dos parâmetros de cada um dos operadores de mudança utilizados.

Tabela 7.5 - Estudo de Caso 1: Instanciação dos Parâmetros de Mudança.

Parâmetros das Mudanças					
Operador	**Parâmetro**	**Tipo**	**Função/Valor**	**Mínimo**	**Máximo**
FAttrib	nOfAttribute	geração/execução	natt*=2	64	1024
FObject	nOfObject	geração/execução	nobj*=2	2	32
FInter	nOfInteraction	geração/execução	nint*=2	2	64
FNModel	nOfModel	geração/execução	nmod+=4	8	32
HNMach	nOfMachine	geração/execução	2, 3, 4	NA	NA

Neste experimento os federados modificados foram instanciados no início de cada rodada, a mudança foi mantida até o final da mesma e não foram necessários procedimentos de recuperação. Desta forma, os atributos gatilho, duração, cardinalidade e tempo de recuperação não se aplicam.

A geração dos federados com mudanças foi realizada utilizando os métodos definidos para cada operador de mudança (ver Apêndice B). Neste estudo de caso só foram considerados cenários de mudança primários.

4) Execução do Experimento

No caso do projeto do experimento de benchmark, nós consideramos rodadas de simulação com duração de quatro minutos, já que esse período garante um conjunto suficiente de dados coletados, mantendo o tempo de execução do experimento factível.

Um mínimo de duas rodadas seria suficiente para exercitar o reinício da comunicação entre os federados, mas foram executadas três rodadas. O tempo de estabilização, no qual não são coletados dados, teve que ser suficiente para que todos os federados iniciassem o seu processamento e estabelecessem a comunicação, bem como o tempo entre as rodadas de simulação teve que ser suficiente para permitir o armazenamento dos resultados. A Tabela 7.6 apresenta os valores para os parâmetros de execução utilizados nos experimentos.

Tabela 7.6 - Estudo de Caso 1: Parâmetros de Execução.

Parâmetros de Execução	
Parâmetro	**Valores**
Número de rodadas	3
Duração da rodada	4 min
Duração Inter-rodadas	1 min
Estabilização	20 s

7.3.1.2. Definição de Aspectos de Validação do Benchmark

Nós definimos quatro repetições do experimento para atestar a propriedade de repetibilidade, considerando que um número maior de repetições seria melhor em termos estatísticos, mas que esse número de repetições é suficiente e não inviabiliza a aplicação do benchmark, afetando muito pouco a propriedade simplicidade. Relativamente ao índice requerido, embora o experimento tenha sido executado em ambiente de execução e de rede controlados, a variação de latência em microssegundos é bastante sensível e definimos o índice de repetibilidade aceitável como sendo de até 5%. A duração esperada para a execução dos experimentos (para cada implementação HLA avaliada) e os parâmetros de repetibilidade estão descritos na Tabela 7.7.

Tabela 7.7 - Estudo de Caso 1: Validação do Experimento.

Validação do Experimento			
Operador de Mudança	**Repetibilidade**		**Duração (horas)**
	Número	**Índice %**	
FAttrib	4	5.0	7,5
FObject	4	5.0	7,5
FInter	4	5.0	9
FNModel	4	5.0	10
HNMach	4	5.0	5

7.3.1.3. Representação do Benchmark

Dois cenários de instanciação foram criados, um para cada atributo avaliado - Latência e Rendimento. Os cenários de instanciação estão representados em dois arquivos RBDL distintos.

7.3.1.4. Resultados dos Experimentos

As próximas subseções apresentam os resultados gerais obtidos nos experimentos de benchmark para os atributos Rendimento e Latência levando em consideração as seguintes perspectivas: (i) *impacto das mudança*s, na qual é analisado como cada mudança aplicada impactou cada uma das RTIs segundo os valores assumidos por seus parâmetros; (ii) *eficiência das RTIs*, na qual é apresentado a média dos resultados de Rendimento e Latência obtidos pelas RTIs para cada mudança aplicada; (iii) *resiliência*, na qual é apresentada a análise dos índices de resiliência obtidos pelas RTIs para cada mudança aplicada; e (iv) *avaliação geral*, na qual os resultados são apresentados de forma conjunta e resumida com o intuito de fornecer subsídios para a comparação das RTIs.

Vale salientar que a Seção 7.3.3 apresenta uma análise geral das RTIs no que tange aos resultados obtidos neste benchmark para os dois estudos de caso executados.

1) Impacto das Mudanças

A Figura 7.3 apresenta o comportamento das RTIs relativamente ao atributo Rendimento para cada uma das mudanças aplicadas.

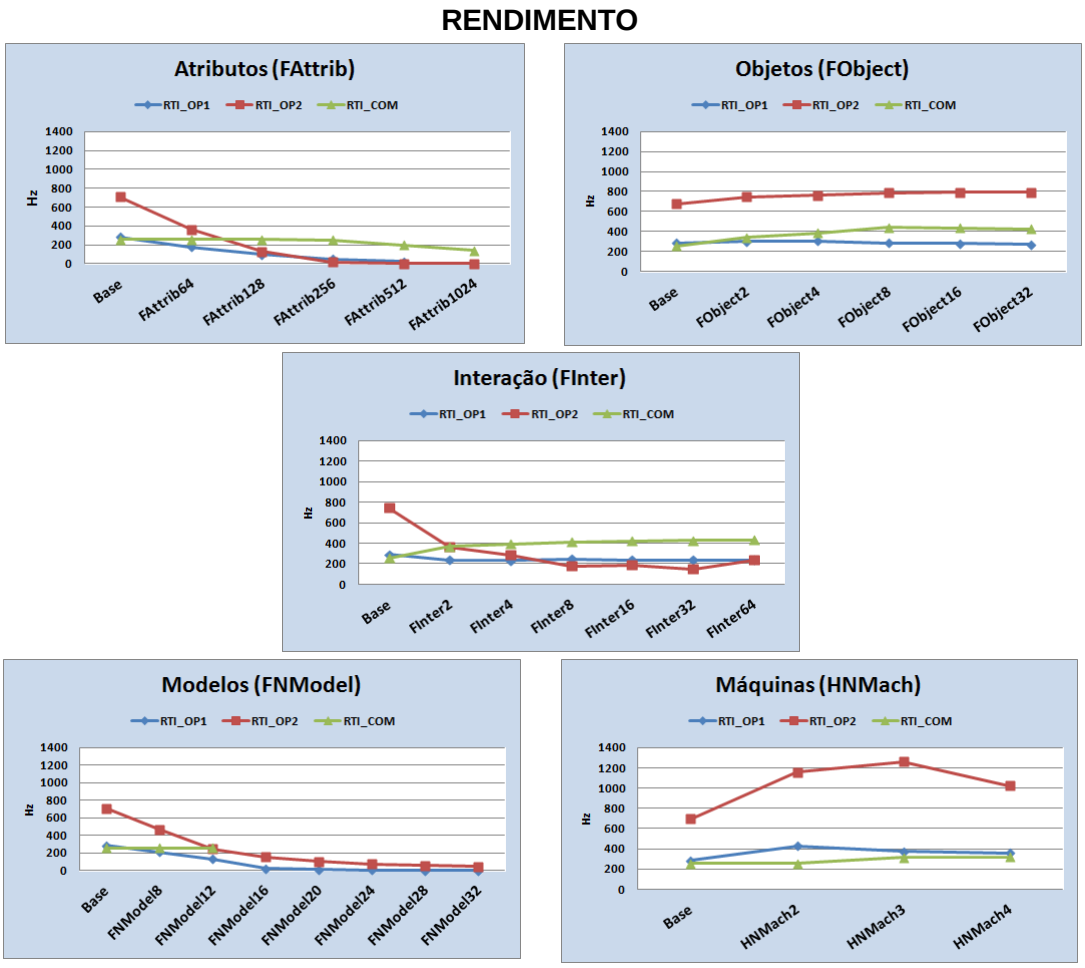


Figura 7.3 - Estudo de Caso 1: Impacto das mudanças - Rendimento.

Em relação à avaliação direta dos resultados em função das mudanças, de maneira geral, houve diminuição do rendimento nos seguintes casos: (i) mudança no número de atributos, o que mostra perda de desempenho quando a RTI faz atualização de um volume maior de dados; (ii) aumento do número de modelos na simulação, o que seria esperado tanto por conta da própria carga de processamento da máquina, quanto em função da perda de eficiência da RTI ao processar serviços oriundos de um número maior de federados; (iii) em função do recebimento de serviços assíncronos, o que também seria esperado já que a *RTILib* tem que executar a atualização dos dados enquanto recebe solicitação de serviços via chamadas de *callback*. Neste último caso, apenas a RTI_COM manteve o índice de rendimento, apresentando inclusive um melhor desempenho no tratamento dos *callbacks* recebidos.

Por outro lado, houve aumento de rendimento quando: (i) foram utilizados vários tipos de objetos para o intercâmbio de dados; esse efeito era esperado já que há um enfileiramento de vários objetos que são atualizados e enviados em um passo de simulação, o que aumenta o rendimento de atualizações de maneira geral, no entanto, se a medida fosse feita por volume de dados publicados o resultado seria distinto; (ii) quando a simulação foi distribuída, já que cada máquina executou um número menor de modelos, havendo uma pequena perda de desempenho quando utilizamos quatro máquinas. Vale salientar que, neste último caso, não dispúnhamos na execução desse experimento máquinas com homogeneidade de capacidade de processamento e memória, sendo a quarta máquina a de menor capacidade. Nesses dois casos o resultado geral esteve de acordo com o esperado.

A Figura 7.4 apresenta os resultados dos experimentos mostrando o comportamento das RTIs relativamente ao atributo Latência para cada uma das mudanças aplicadas.

Analisando os resultados em função das mudanças, pode-se observar um aumento na latência nos seguintes casos: (i) mudança no número de atributos, o que mostra aumento na latência tanto em função do tempo de atualização de um maior número de atributos, quanto em função do tamanho das mensagens trocadas; (ii) aumento do número de modelos na simulação, houve uma pequeno aumento na latência em função do número de modelos, provavelmente relacionado à sobrecarga da RTI; e (iii) em função do número de máquinas e portanto do uso da rede. A perda de eficiência em função da distribuição da simulação foi pequena, entretanto devemos levar em consideração o fato de utilizarmos um ambiente de rede controlado. Era esperado que o aumento do número de atributos e o uso da rede causassem um aumento na latência.

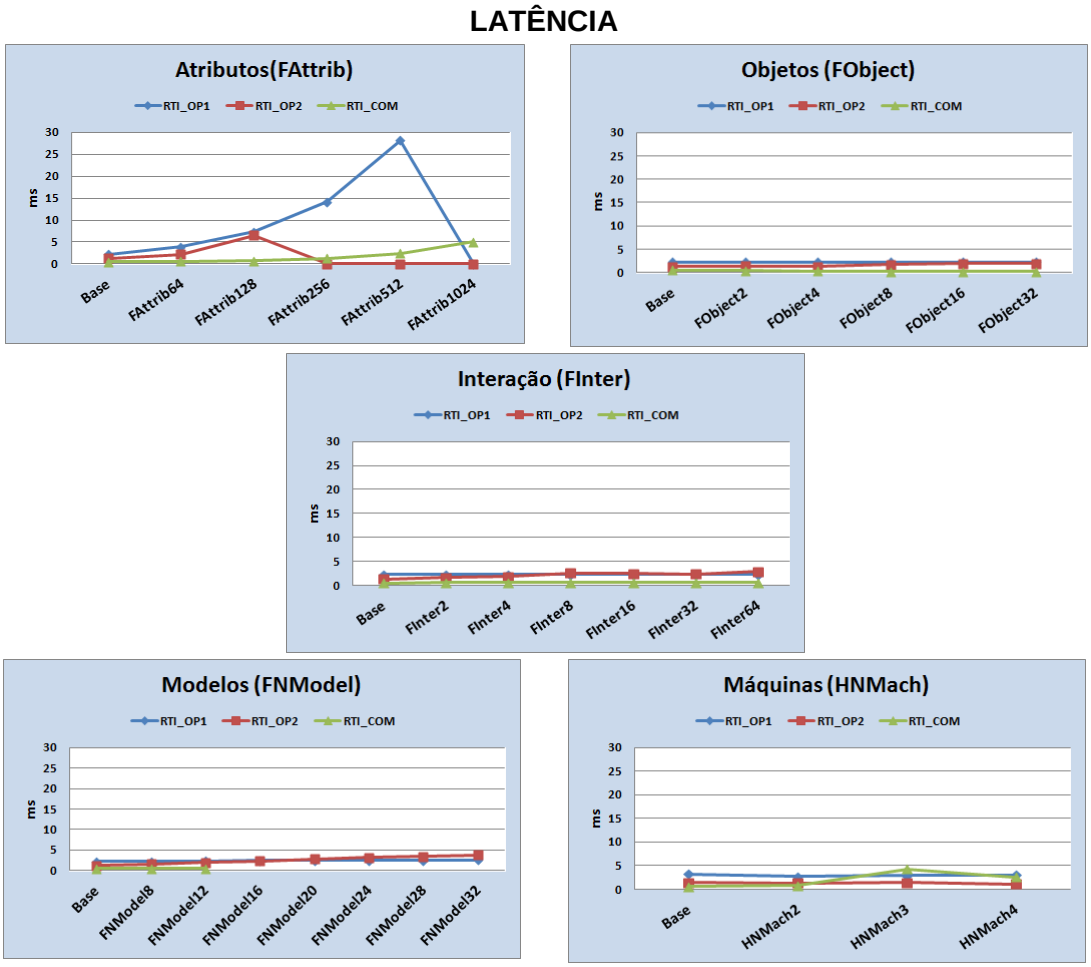


Figura 7.4 - Estudo de Caso 1: Impacto das mudanças - Latência.

2) Eficiência das RTIs HLA

A Figura 7.5 apresenta, para cada uma das implementações RTI HLA avaliadas, o resultado do Rendimento na execução de referência (base) e a média dos resultados de Rendimento nas execuções com mudanças.

RENDIMENTO

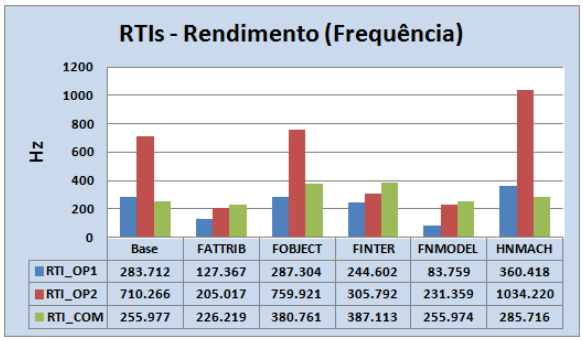


Figura 7.5 - Estudo de Caso 1: Rendimento das RTIs HLA.

Observando a Figura 7.5 pode-se perceber que a RTI_OP2 é mais eficiente, entretanto, em todos os testes essa implementação mostrou-se mais instável que as demais. Por exemplo, em relação ao aumento no número de atributos trocados entre os federados, a partir de 256 atributos o federado receptor deixou de receber as mensagens corretamente e quando havia mais de duas máquinas na simulação houve erro em vários testes.

A RTI_OP1 teve desempenho compatível com o desempenho da implementação comercial RTI_COM, mas sofreu impacto maior com a aplicação das mudanças e apresentou erro quando houve intercâmbio de 1024 atributos. A RTI_COM teve bom desempenho e não apresentou erros quando exposta a nenhum tipo de mudança.

A Figura 7.6 apresenta, para cada uma das implementações RTI HLA avaliadas, o resultado da Latência na execução de referência (base) e a média dos resultados da Latência nas execuções com mudanças.

LATÊNCIA

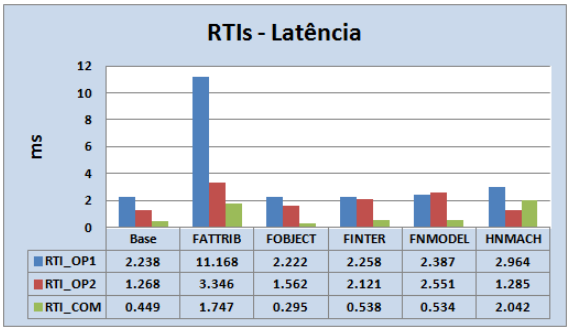


Figura 7.6 - Estudo de Caso 1: Latência das RTIs HLA.

As RTI_COM e RTI_OP1 utilizam o protocolo TCP-IP (ponto a ponto entre federados) mesmo para transmissão de mensagens do tipo *não-confiável*. Em termos de eficiência, a RTI_OP1 foi a que apresentou a maior latência na troca de dados, enquanto a RTI comercial apresentou o melhor desempenho nesse quesito.

3) Resiliência das RTIs HLA

A Figura 7.7 apresenta os resultados comparativos dos índices de resiliência obtidos pelas RTIs para o atributo Rendimento.

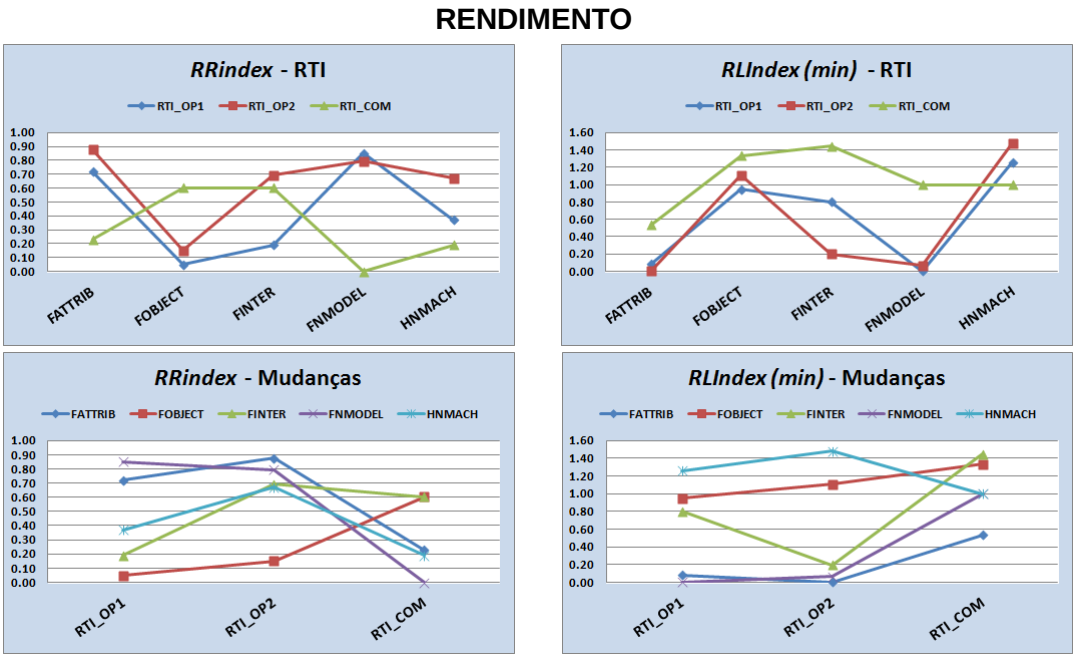


Figura 7.7 - Estudo de Caso 1: Índices de Resiliência - Rendimento.

Pode-se observar na Figura 7.7 que para as duas RTIs de *código aberto* o perfil de resiliência ficou bastante parecido, demonstrando que as mudanças afetaram essas RTIs de forma análoga, com pequenas variações. Entretanto, a RTI_OP1 mostrou-se mais resiliente, confirmando a instabilidade da RTI_OP2. A implementação comercial RTI_COM obteve melhores índices de resiliência, tendo altos índices apenas para a resiliência positiva, ou seja, quando houve um aumento no rendimento. Para o operador de mudanças *FInter* (chamada assíncrona), o resultado para a RTI_COM foi contrário ao resultado das demais, demonstrando uma maior capacidade de executar os serviços de

callback paralelamente à publicação dos dados do modelo. A RTI_COM e RTI_OP2 utilizam *threads* separadas para tratamento de *callbacks*.

O gráfico comparativo do índice de resiliência por mudanças aponta que as mudanças no número de atributos e no número de modelos foram as que causaram maior perda de resiliência, comprovando a análise realizada observando os gráficos separadamente, demonstrando que os índices de resiliência representaram corretamente o impacto das mudanças.

A Figura 7.8 apresenta os resultados do índice de resiliência para o atributo Latência.

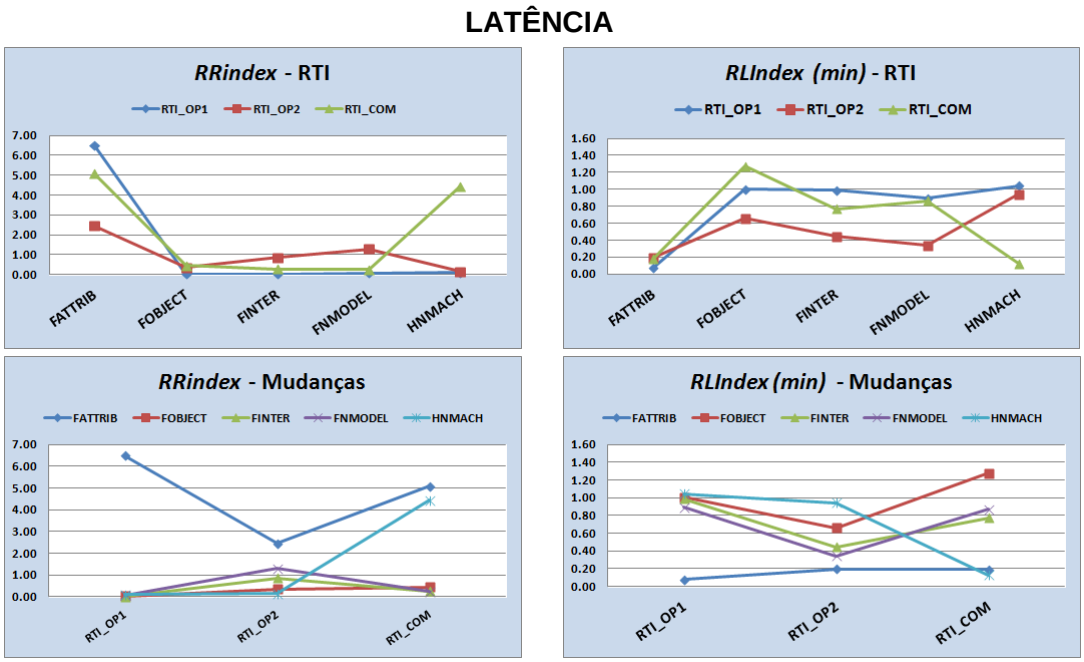


Figura 7.8 - Estudo de Caso 1: Índices de Resiliência - Latência.

Relativamente à resiliência da RTIs, podem-se fazer as seguintes observações: (i) a implementação RTI_OP1 tem o pior desempenho em termos de latência, mas a maior resiliência; (ii) o perfil de resiliência da mudança no número de atributos (*FAttrib*) foi o de maior impacto na resiliência das implementações; (iii) o perfil de resiliência das implementações de *código aberto* também mostrou semelhanças no caso da latência; e (iv) a RTI_COM foi muito pouco resiliente relativamente à distribuição da simulação.

Como já havia ocorrido relativamente ao atributo Rendimento, a RTI_OP2 teve o pior índice de resiliência geral e a implementação também foi a mais instável e sujeita a erros.

4) Avaliação Geral das RTIs HLA

Para a escolha de uma RTI utilizando os resultados do Estudo de Caso 1, nós avaliamos a eficiência absoluta das RTIs em conjunto com os índices de resiliência obtidos. Para tanto, utilizamos o valor do desempenho medido no experimento de referência (ver Figuras 7.5 e 7.6) e o pior caso (*RLIndexMin*), ou seja, o pior índice de perda de serviço, para avaliar cada uma das mudanças (ver Figuras 7.7 e 7.8). Assim, o resultado da avaliação global é o produto da eficiência no experimento de referência pelo índice de resiliência que indica perda de serviço no pior caso (*RLIndexMin*). A Tabela 7.8 apresenta a sumarização dos resultados para Rendimento.

Tabela 7.8 - Estudo de Caso 1: Avaliação Geral RTIs - Rendimento

RENDIMENTO						
		FATTRIB	FOBJECT	FINTER	HNMACH	GERAL
RTI_OP1	Base	283.712	283.712	283.712	283.712	
	RLIndexMin	0.082	0.948	0.800	1.260	0.773
	Resultado	**23.264**	**268.959**	**226.970**	**357.477**	**219.168**
RTI_OP2	Base	710.266	710.266	710.266	710.266	
	RLIndexMin	0.002	1.108	0.198	1.478	0.697
	Resultado	**1.421**	**786.975**	**140.633**	**1050.057**	**494.771**
RTI_COM	Base	255.977	255.977	255.977	255.977	
	RLIndexMin	0.537	1.333	1.445	1.000	1.079
	Resultado	**137.460**	**341.217**	**369.887**	**255.977**	**276.135**

> RESILIÊNCIA < RESILIÊNCIA MELHOR AVALIAÇÃO GLOBAL

Embora a RTI_OP2 tenha o melhor desempenho para o atributo Rendimento considerando o pior caso de resiliência para cada operador de mudança, ela é a mais instável das implementações, o que está indicado pelo índice de resiliência. Como já mencionado, a implementação apresentou um conjunto de erros e o índice de resiliência associado à análise de outros atributos, tais

como confiabilidade e estabilidade, poderiam dar esse indicativo e levar à escolha de outra implementação.

A Tabela 7.9 apresenta a sumarização dos resultados para Latência.

Tabela 7.9 - Estudo de Caso 1: Avaliação Geral RTIs - Latência.

LATÊNCIA						
		FATTRIB	FOBJECT	FINTER	HNMACH	GERAL
RTI_OP1	Referência	2.238	2.238	2.238	2.238	
	RLIndexMin	0.079	1.006	0.989	1.043	0.779
	Resultado	**28.335**	**2.225**	**2.263**	**2.146**	**8.742**
RTI_OP2	Referência	1.268	1.268	1.268	1.268	
	RLIndexMin	0.198	0.657	0.447	0.942	0.561
	Resultado	**6.405**	**1.930**	**2.837**	**1.346**	**3.129**
RTI_COM	Referência	0.449	0.449	0.449	0.449	
	RLIndexMin	0.189	1.278	0.771	0.129	0.592
	Resultado	**2.373**	**0.351**	**0.582**	**3.477**	**1.696**

> RESILIÊNCIA < RESILIÊNCIA MELHOR AVALIAÇÃO GLOBAL

A RTI_COM tem o melhor desempenho geral mesmo em face do pior caso de perda de serviço para cada mudança, no entanto, a distribuição da simulação afetou fortemente essa implementação. Relativamente à avaliação de resiliência, a RTI_OP1 mostrou-se a mais resiliente das três, embora tenha sido a implementação menos resiliente no caso da mudança "*aumento no número de atributos*".

Poderiam ter sido utilizadas outras abordagens para uma avaliação geral dos resultados, considerando as diferentes mudanças e os diferentes atributos de forma conjunta. Por exemplo, poderiam ter sido atribuídos pesos aos resultados das mudanças de acordo com o *nível de exposição* da RTI a cada mudança definido na carga de mudanças, ou ainda, as RTIs poderiam ter sido avaliadas considerando os dois atributos de forma combinada, também utilizando pesos que representassem o grau de importância de cada um. Importa salientar, entretanto, que qualquer abordagem que busque um índice único de resiliência ou de escolha penaliza a análise comparativa.

7.3.2. Estudo de Caso 2: Cenários Compostos

Este estudo de caso, uma extensão do Estudo de Caso 1, tem por objetivo demonstrar a viabilidade e flexibilidade do uso da metodologia, da RBDL e da ferramenta de benchmarking na representação de cenários de mudança compostos e no uso dos atributos de aplicação das mudanças.

Neste estudo de caso foram consideradas mudanças da subclasse *Escala* e aspectos potencialmente críticos para o desempenho de uma RTI, por exemplo, aceleração do passo de simulação no que se refere à Latência ou a sobrecarga de processamento nas máquinas usadas na simulação no que se refere à Latência e ao Rendimento.

7.3.2.1. Cenário de Instanciação

Para instanciação do benchmark foram adotados os passos propostos pela metodologia.

1) Objetivos do Benchmark

Neste passo foi considerada a resiliência dos atributos de desempenho – Latência e Rendimento – e foram utilizadas as mesmas métricas avaliadas no Estudo de Caso 1.

2) Cenário Base

Como carga de trabalho, nós consideramos um simulador com as mesmas características da carga de trabalho do Estudo de Caso 1.

Os experimentos deste estudo de caso foram realizados considerando configurações semelhantes às apresentadas na Figura 7.2. No entanto, este estudo de caso usou apenas três equipamentos (HLA01 a HLA03), com as seguintes mudanças nas configurações: HLA01 (Intel Core I7, 8 GB de memória e 1 TB de disco) e HLA02 (Intel Core I5. 4 GB de memória e 320 GB de disco). As demais configurações e equipamentos foram mantidos.

3) Operadores de Mudanças

Neste estudo de caso foram consideradas mudanças da classe *Requisito* e subclasse *Escala*, similares ao Estudo de Caso 1, e foram realizados dois experimentos distintos.

Visando demonstrar a composição dos cenários, foram escolhidos operadores de mudança da subclasse *Perfil de Uso* que poderiam ter impacto em aspectos de desempenho. Nesse sentido, consideramos mudanças na frequência do passo de simulação (*FAccel*) e o seu efeito sobre a Latência, bem como o impacto de se distribuir a simulação em máquinas que executam outros processos (*HProcUse*) em um cenário que promovesse o aproveitamento de plataformas já existentes utilizadas para outras finalidades. As duas mudanças introduzidas nos cenários foram classificadas com nível de exposição *Mandatório* e *Muito Alto* respectivamente. A Tabela 7.10 apresenta os operadores de mudança usados.

Tabela 7.10 - Estudo de Caso 2: Operadores de Mudança.

Classe: Ambientais [Subclasse: Perfil de Uso]		
#	**Operador**	**Descrição**
C21	FAttrib	Aumento no volume de dados intercambiados entre federados (publicação e subscrição de dados) - número de atributos.
C21	FAccel	Variação na frequência do passo de simulação.
C22	HNMach	Mudança no número de máquinas na simulação (aumento, diminuição).
C22	HProcUse	Variação no padrão de uso dos processadores.

Relativamente aos parâmetros dos operadores *FAccel* e *HProcUse*, foram utilizados os valores apresentados na tabela 7.11. Para os demais operadores, foi utilizado somente um subconjunto dos valores apresentados na Tabela 7.4, já que o encadeamento de mudanças em um cenário composto pode levar ao comprometimento da propriedade simplicidade do benchmark.

Tabela 7.11 - Estudo de Caso 2: Instanciação dos Parâmetros de Mudança.

Parâmetros das Mudanças					
Operador	**Parâmetro**	**Tipo**	**Função/Valor**	**Mínimo**	**Máximo**
FAccel	stepFrequency	geração/execução	step +=30	0	120
HProcUse	numberOfProcess	geração/execução	nProc += 3	0	12
HProcUse	processName	geração	HLACPULoad.exe[14]	NA	NA
HProcUse	*threads*	geração	1	NA	NA

Neste estudo de caso os atributos de aplicação da mudança não exercitados no Estudo de Caso 1, tais como gatilho, duração e cardinalidade, foram exercitados. Para aceleração, consideramos um cenário no qual a rodada de simulação começa e em seguida o condutor da simulação acelera o processamento até quase o final da mesma; já para carga de processamento, consideramos processos que são executam por cerca de um minuto. Os valores usados visam demonstrar o uso dos atributos de aplicação das mudanças e estão de acordo com o tempo de cada rodada de simulação adotado no experimento, conforme apresentado na Tabela 7.12.

Tabela 7.12 - Estudo de Caso 2: Atributos das Mudanças.

Atributos das Mudanças				
Operador	**Gatilho**	**Duração**	**Recuperação**	**Cardinalidade**
FAccel	20s	2min 20s	NA	1
HProcUse	30s	1min 10s	NA	2

Neste estudo de caso foram considerados cenários de mudança compostos do tipo cascata, no qual cada operador de mudança é combinado aos demais. A ordem de aplicação das mudanças está apresentada na Tabela 7.13.

Tabela 7.13 - Estudo de Caso 2: Cenários de Mudança.

Cenários de Instanciação	
#	**Operadores**
C21	FNModel -> FAttrib -> FAccel
C22	FNModel -> HNmach -> HProcUse

[14] O programa HLACPULoad.exe executa *threads* que simulam uma carga de processamento em uma máquina. Cada processo consome em média, para as máquinas usadas nos experimentos, 25% de CPU.

Importa salientar que, embora tenham sido implementadas mudanças compostas considerando a mudança no número de modelos (*FNModel*), os experimentos para os cenários C21 e C22 foram realizados usando oito e seis modelos respectivamente, dada as limitações da RTI_COM.

4) Execução do Experimento

Os parâmetros de execução deste estudo de caso foram os mesmos usados no Estudo de Caso 1.

7.3.2.2. Definição de Aspectos de Validação do Benchmark

Para os cenários compostos, visando manter a viabilidade dos experimentos, definimos três repetições. Relativamente ao índice de repetibilidade requerido, nós consideramos o mesmo limite aceito no estudo de caso anterior (5%).

A duração da execução dos experimentos para cada implementação HLA avaliada, bem como os parâmetros de repetibilidade estão descritos na Tabela 7.14.

Tabela 7.14 - Estudo de Caso 2: Validação do Experimento.

Validação do Experimento			
Operador de Mudança	**Repetibilidade**		**Duração (horas)**
	Número	**Índice %**	
C21	3	5.0	17
C22 (Latência)	3	5.0	14
C22 (Rendimento)	3	5.0	14

7.3.2.3. Representação do Benchmark

O Estudo de Caso foi definido em dois Cenários de Instanciação (C21 e C22), cada um deles representado em um arquivo RBDL.

7.3.2.4. Resultados dos Experimentos

Os resultados deste estudo de caso foram apresentados considerando as mesmas perspectivas apresentadas para o Estudo de Caso 1: impacto das

mudanças, desempenho das RTIs, índices de resiliência e avaliação geral das RTIs.

1) Impacto das Mudanças

A Figura 7.9 apresenta o perfil do impacto do Cenário de Instanciação C22 relativamente ao atributo Rendimento para as duas RTIs HLA avaliadas.

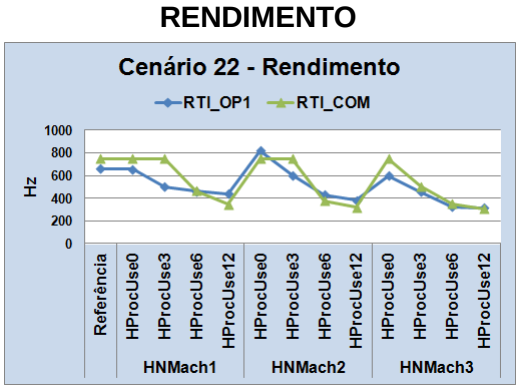


Figura 7.9 - Estudo de Caso 2: Impacto das mudanças - Rendimento.

Pode-se observar que, quando os federados são distribuídos em diferentes máquinas e não há sobrecarga de processamento da CPU, há um aumento no Rendimento em função do balanceamento de carga. Já à medida que são instanciados processos que sobrecarregam o processamento das CPU das máquinas utilizadas, há uma perda no atributo Rendimento. Assim, o conjunto de mudanças representado pelo Cenário de Instanciação C22 teve impacto condizente com o esperado.

A Figura 7.10 apresenta os resultados relativamente ao atributo Latência para os Cenário de Instanciação C21 e C22.

LATÊNCIA

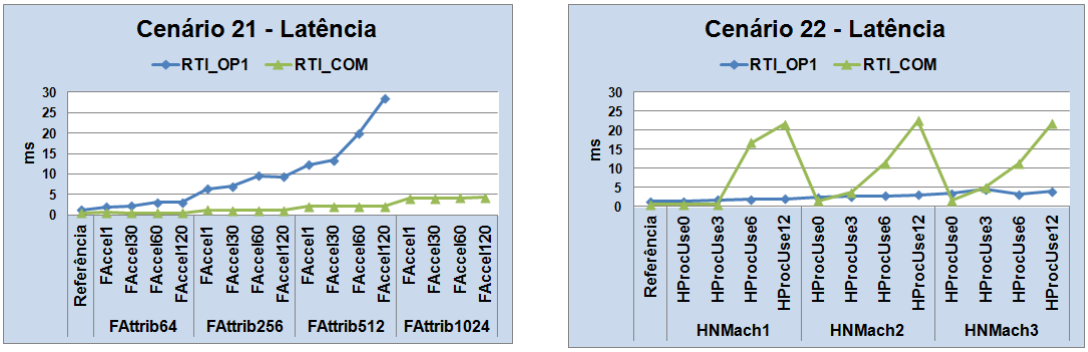


Figura 7.10 - Estudo de Caso 2: Impacto das mudanças - Latência.

Quanto ao atributo Latência, pode-se observar que os cenários de instanciação impactaram as implementações de formas distintas.

Para o Cenário de Instanciação C21, enquanto a RTI_COM sofreu apenas um pequeno impacto com a mudança no número de atributos e praticamente nenhum impacto em função do aumento da frequência do passo da simulação, a RTI_OP1 sofreu algum impacto advindo da mudança aceleração do passo da simulação e grande impacto em função do aumento do número de atributos. Esse mesmo impacto do número de atributos na Latência da RTI_OP1 já havia sido observado e analisado no Estudo de Caso 1.

Já para o Cenário de Instanciação C22, ocorreu o oposto, a implementação RTI_COM sofreu um grande impacto em função da mudança no perfil de distribuição da simulação, o que pode ser explicado por otimizações quando a comunicação é local, e sofreu, comparativamente, impacto maior com a carga de processamento nas máquinas da simulação. A RTI_OP1 teve a sua Latência menos alterada diante da sobrecarga de processamento, o que pode ser explicado pela descentralização no envio das mensagens, já que há um processo a parte para atender aos *callbacks.*

2) Eficiência das RTIs HLA

A seguir são apresentados os resultados de cada um dos experimentos relativamente à eficiência das RTIs. Para os cenários C21 (*FAttrib, FAccel*) e C22 (*HNmach, HProcUse*), e para cada um dos atributos avaliados, é

apresentada a medida de referência e a média das medidas com mudanças obtidas por cada uma das implementações HLA.

A Figura 7.11 apresenta os resultados da avaliação do atributo Rendimento para Cenário de Instanciação C22 (ver Tabela 7.12).

RENDIMENTO

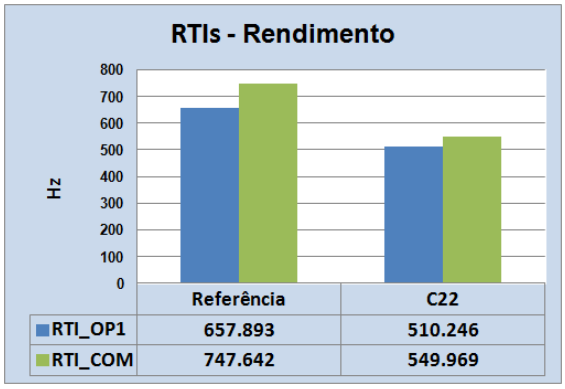


Figura 7.11 - Estudo de Caso 2: Rendimento das RTIs HLA.

A Figura 7.11 mostra que o conjunto de mudanças representado pelo Cenário de Instanciação C22 teve impacto semelhante no rendimento das duas implementações. O Rendimento de ambas mostrou-se bastante semelhante, o que confirmou os resultados do Estudo de Caso 1.

A Figura 7.12 apresenta os resultados relativamente à avaliação do atributo Latência para os Cenários de Instanciação C21 e C22.

LATÊNCIA

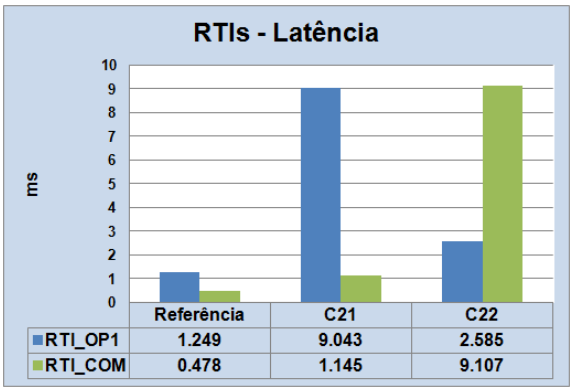


Figura 7.12 - Estudo de Caso 2: Latência das RTIs HLA.

A RTI_COM tem um desempenho melhor em relação ao atributo Latência quando a simulação acontece em uma máquina centralizada e a RTI_OP1 sofre grande impacto relativamente ao aumento no número de atributos. A aplicação das mudanças combinadas reforçou os resultados já apresentados no Estudo de Caso 1.

3) Resiliência das RTIs HLA

O índice de resiliência é apresentado de forma conjunta para os dois cenários de instanciação e para os atributos Latência e Rendimento. Assim, a Figura 7.13 apresenta os resultados para o Cenário de Instanciação C21 e para os Cenários de Instanciação C22T (Rendimento) e C22L (Latência).

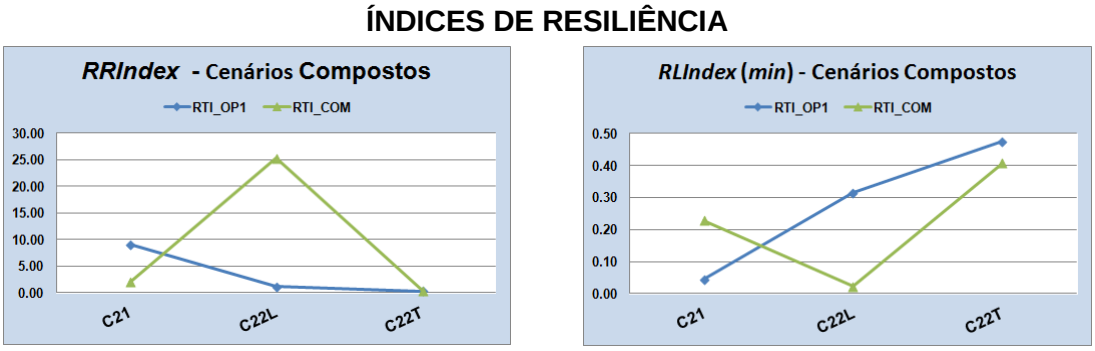


Figura 7.13 - Estudo de Caso 2: Índices de Resiliência.

Tal qual ocorrido no Estudo de Caso 1, os perfis dos índices de resiliência exprimiram o comportamento das RTIs e são suficientes para que se possa inferir o impacto geral das mudanças. Por exemplo, analisando apenas os índices de resiliência é possível observar que as duas RTIs foram impactadas de forma oposta relativamente à avaliação da Latência para os cenários de mudança aplicados.

4) Avaliação Geral das RTIs HLA

A avaliação geral das RTIs foi realizada de forma análoga à do Estudo de Caso 1. A Tabela 7.15 sumariza os resultados de Rendimento e Latência considerando os dois cenários de instanciação utilizados.

Tabela 7.15 - Estudo de Caso 2: Avaliação Geral RTIs.

		LATÊNCIA			RENDIMENTO		
		C21	C22	GERAL	C22	GERAL
RTI_OP1	Referência	1.249	1.249		657.893	
	RLIndexMin	0.044	0.314	0.179	0.475	0.475
	Resultado	**28.457**	**3.971**	**16.214**	**312.406**	**312.406**
RTI_COM	Referência	0.478	0.478		747.642	
	RLIndexMin	0.229	0.022	0.126	0.408	0.408
	Resultado	**2.086**	**21.547**	**11.817**	**304.998**	**304.998**

> RESILIÊNCIA < RESILIÊNCIA MELHOR AVALIAÇÃO GLOBAL

Usando o mesmo critério de escolha para os cenários compostos analisados, a RTI_COM teve vantagem comparativa relativamente à Latência, especialmente porque tem uma latência de referência (máquina centralizada) bastante competitiva. No entanto, essa implementação sofreu um grande impacto relativamente à distribuição da simulação e sobrecarga de processamento nas máquinas utilizadas. Quando os cenários são avaliados em conjunto (C21 e C22), considerando os perfis de uso avaliados, e para o conjunto de mudanças aplicado, a RTI_OP1 é a mais resiliente.

Para o atributo rendimento as duas implementações obtiveram resultados muito semelhantes durante os testes, com uma pequena vantagem para a RTI_OP1, que também se mostrou mais resiliente.

7.3.3. Análise Geral dos Resultados

Algumas observações sobre os resultados dos experimentos realizados podem ser feitas à luz de alguns aspectos de projeto das RTIs HLA, nomeadamente a arquitetura e o uso dos canais de comunicação. No entanto, esta análise carece de aprofundamento devido à indisponibilidade de documentação que apresente detalhes acerca de aspectos de implementação de todas as RTIs.

- A arquitetura descentralizada da RTI_OP2 mostrou-se melhor em relação ao atributo Rendimento, mas menos resiliente que as demais, tendo sido a implementação que apresentou pior índice de resiliência no geral, especialmente relativamente ao tratamento de mensagens assíncronas. Em arquiteturas descentralizadas as funcionalidades da RTI são executadas

pela biblioteca RTI ligadas aos federados (*RTILib*). Além disso, a RTI_OP2 utiliza uma camada intermediária para comunicação (*JGroups*), enquanto as demais utilizam o protocolo TCP-IP e UDP diretamente.

- A RTI_OP1 utiliza arquitetura centralizada (processo *RTIExec*) mas implementa as funcionalidades do *RTI Ambassador* (recebimento de *callbacks*, processamento de filas de mensagens, etc.) por meio de um processo separado (RTIA) ao invés de uma *thread* separada. Assim, quando o federado chama um serviço da *RTILib* o mesmo é repassado via protocolo TCP-IP para o processo *Ambassador* (RTIA). Essa arquitetura apresentou pior desempenho de referência para o atributo Latência em uma máquina local e a implementação sofreu um grande impacto relativamente a mudanças no número de atributos, o que pode ser explicado pela sobrecarga causada pelo processo intermediário. No entanto, essa arquitetura se mostrou mais resiliente relativamente à Latência tanto na distribuição da simulação, quanto em presença de cargas de processamento nas máquinas de simulação.

- O *callback* executado em *thread* separada na implementação comercial, quando comparado a um processo separado como o utilizado pela implementação RTI_OP1, garantiu maior eficiência no que se refere ao recebimento de mensagens assíncronas, mesmo quando é responsabilidade do federado a solicitação do recebimento da mensagem (modelo de callback HLA_EVOKED). O mesmo não ocorreu na implementação RTI_OP2.

- Em simulações centralizadas, relativamente ao atributo de latência, o uso do protocolo não se mostrou determinante já que a implementação que teve o melhor desempenho (RTI_COM) e o pior desempenho (RTI_OP1) utilizaram o protocolo TCP-IP para comunicação entre os federados. No entanto, quando a simulação é distribuída, a RTI_COM não mantém o seu desempenho, sendo muito menos resiliente a esse fator. Há um indicativo

de que a RTI_COM otimize a comunicação local usando memória compartilhada.

De maneira geral, quando são levados em consideração os dois Estudos de Caso, pode-se observar que embora a RTI_COM tenha apresentado melhores resultados de referência em grande parte dos experimentos, a RTI_OP1 se mostrou bastante resiliente e seria a escolhida em alguns deles. Assim, a RTI_OP1 que é uma RTI de *código aberto* poderia ser considerada para uso em simuladores de satélite.

Vale salientar que a RTI_COM e a RTI_OP2 podem ser configuradas de forma a ter o seu desempenho melhorado. No entanto, isso não foi feito, de um lado para manter a igualdade de configuração entre as três RTIS, de outro porque esse já um aspecto da carga de mudanças que permite avaliar e comparar o efeito de ajustes (*tunning*) nas implementações e poderia ser alvo de outro benchmark concreto.

As observações acima foram feitas tendo como base as RTIs analisadas e, embora demonstrem comportamento lógico e esperado relativamente às características de projeto das RTIs, elas não podem ser diretamente extrapoladas para outras implementações que empreguem a mesma arquitetura ou tecnologia, nem extrapoladas para o impacto de outras mudanças.

7.3.4. Validação do Benchmark

Os benchmarks de resiliência instanciados a partir da metodologia devem ser validados de acordo com as propriedades e critérios especificados (ver Capítulo 4). Esta seção apresenta a validação do benchmark instanciado, visando atestar o atendimento ao seguinte conjunto de propriedades: escalabilidade/portabilidade, representatividade, repetibilidade, simplicidade de uso e não-intrusão.

É importante salientar que consideramos o Estudo de Caso 2 como uma extensão do Estudo de Caso 1 e, assim, a validação aqui apresentada

representa a instanciação do benchmark de resiliência nos dois estudos de caso. Sempre que pertinente, mencionaremos as especificidades de cada um deles.

7.3.4.1. Escalabilidade/portabilidade

Relativamente à portabilidade do benchmark, nós consideramos que benchmarks de resiliência já pressupõem cenários compostos nos quais algumas mudanças da classe de mudança *Tecnológica* e subclasse *Implementação* podem ser exercitadas relativamente às demais mudanças de interesse, compondo cenários guiados pela tecnologia que permitem não apenas comprovar a portabilidade do experimento, mas também atestar a resiliência do sistema sob essa perspectiva. Raciocínio análogo pode ser feito relativamente à propriedade escalabilidade. Os estudos de caso apresentados tinham como foco a mudança de *Escala*, mas não exercitaram outras tecnologias, entretanto bastaria considerar cenários compostos guiados, por exemplo, pela mudança *OSVerExec* ("*mudanças nos sistemas operacionais*") para atestar a portabilidade do experimento para outros sistemas operacionais.

7.3.4.2. Representatividade

O uso de uma mesma infraestrutura de simulação em diferentes missões pressupõe mudanças de escala do objeto simulado. Nos estudos de caso apresentados foi utilizada uma carga de trabalho que representava o simulador de um satélite experimental (ver Apêndice B). Já os Cenários de Mudança levaram em consideração: satélites mais complexos que aceitavam a chamadas assíncronas (utilizadas em simuladores operacionais que aceitam o recebimento de telecomandos), a distribuição da simulação em um conjunto de máquinas com diferentes cargas de processamento e aspectos de aceleração da simulação.

Relativamente à representatividade dos parâmetros das mudanças, pode-se salientar: (i) tanto o número máximo de modelos (*FNModel*), quanto o número máximo de atributos em cada modelo (*FAttrib*) assumiram valores maiores que

os esperados para um simulador de satélite de sensoriamento remoto conforme apresentado na Tabela B.11; (ii) o número de objetos (*FObject*) utilizado para intercâmbio de dados seguiu o número de atributos; (iii) chamadas assíncronas foram introduzidas para representar eventos em simuladores, por exemplo, envio de telecomandos em simuladores operacionais, e os valores utilizados para os parâmetros excederam os número de eventos esperados; (iv) o valor máximo para o parâmetro de aceleração (*FAccel*) buscou exercitar o limite mínimo de duração do passo de simulação; (v) o número de máquinas na distribuição da simulação (*HNMach*) foi limitado pela disponibilidade de equipamentos no laboratório; e (vi) a carga das máquinas (*HProcUse*) ficou em até 100% de utilização e cada processo utilizava em média 25% da capacidade da CPU das máquinas de execução utilizadas.

7.3.4.3. Repetibilidade

Os experimentos do Estudo de Caso 1 foram reaplicados três vezes apresentando índices de repetibilidade (IR) para Rendimento e Latência conforme apresentado na Tabela 7.16 e Tabela 7.17 respectivamente. Três repetições não é o número de repetições ideal para validar estatisticamente a repetibilidade do experimento, mas dá uma ideia suficientemente boa da variabilidade dos resultados.

Tabela 7.16 - Estudo de Caso 1: Índice de Repetibilidade - Rendimento

Repetibilidade: Rendimento (*TFreq*)			
	RTI_OP1	**RTI_OP2**	**RTI_COM**
FAttrib	0.775	2.015	3.663
FObject	0.648	2.282	0.913
FInter	0.416	5.829	0.020
FNModel	2.757	0.854	0.001
HNMach	3.167	4.482	1.110

Tabela 7.17 - Estudo de Caso 1: Índice de Repetibilidade - Latência

Repetibilidade: Latência (*LUdpdSqrt*)			
Operador	**RTI_OP1**	**RTI_OP2**	**RTI_COM**
FAttrib	0.320	4.524	0.659
FObject	0.139	0.407	0.993
FInter	0.228	2.141	0.564
FNModel	0.178	2.067	1.132
HNMach	3.295	1.477	0.215

O índice de repetibilidade obtido ficou abaixo do índice definido como aceitável (5%) para a totalidade dos operadores de mudança. Por meio do índice de repetibilidade pode-se observar que para a maioria das implementações, tanto relativamente atributos Latência quanto Rendimento, o operador HNMach foi o que apresentou maior variabilidade quando os experimentos foram repetidos, o que é esperado dada à maior instabilidade causada pelo uso da rede de computadores. No entanto, mesmo nesse caso os índices de repetibilidade ficaram abaixo dos 5%.

Os experimentos do Estudo de Caso 2 também foram repetidos três vezes e apresentaram os resultados conforme indicados nas Tabelas 7.18.

Tabela 7.18 - Estudo de Caso 2: Índice de Repetibilidade.

Repetibilidade		
	RTI_OP1	**RTI_COM**
C21 (*LUdpdSqrt)*	0.379	0.7557
C22 (*LUdpdSqrtRT*)	5.269	19.013
C22 (*TFreq*)	2.925	2.095

Para os Cenários de Instanciação C21 e C22 Rendimento, o índice de repetibilidade ficou abaixo do nível definido. No entanto, a RTI_COM para o cenário C22 (Latência) apresentou variações nos resultados dos experimentos quando houve sobrecarga de processamento nas máquinas associada ao uso da rede. Essa instabilidade já havia sido observada analisando o índice de resiliência.

7.3.4.4. Simplicidade de Uso

Os experimentos foram representados por arquivos RBDL, em 5 cenários de instanciação. A simplicidade de aplicação do experimento foi garantida pela ferramenta de benchmark que teve a capacidade de, a partir dos arquivos RBDL, gerar as cargas de trabalho e mudanças automaticamente, bem como de executar os experimentos de forma automática para cada uma das RTIs HLA. A duração dos experimentos ficou próxima à prevista (dez dias para o Estudo de Caso 1 e quatorze dias para o Estudo de Caso 2).

O *framework* descrito no Capítulo 6 foi usado na implementação da ferramenta de benchmark e apenas as mudanças específicas tiveram que ser implementadas.

7.3.4.5. Não-intrusão

O benchmark apresentado nestes estudos de caso utiliza a RTI HLA de forma padronizada, testando apenas aspectos padronizados de escalabilidade que não afetam ou mudam a implementação em si, por exemplo, mudanças genéricas em federados da federação, no número de federados, no padrão de distribuição, aceleração do passo de simulação e carga dos processadores. Não foi feita nenhuma alteração nas implementações testadas e, com o objetivo de preservar a igualdade entre as implementações, a configuração das RTIs foi padronizada e equânime.

7.4. Benchmark de Robustez

O Benchmark de Robustez aqui apresentado aplica técnica análoga à utilizada na ferramenta Ballista (KOOPMAN; DEVALE, 1999), na qual cada serviço da API HLA é exercitado considerando valores inválidos para seus parâmetros.

São apresentados os resultados da aplicação desse benchmark na avaliação de duas implementações RTI HLA (RTI_OP1 e RTI_OP2), já que no momento da execução dos experimentos eram as únicas implementações disponíveis.

7.4.1. Estudo de Caso 3

Neste estudo de caso os modelos de simulação (federados) foram considerados como fonte de mudanças, em falhas do operador *FIntFault* que representa as injeções de falhas de interface ou erros nas chamadas à API HLA. O objetivo foi avaliar quão robusto se mostrava a RTI HLA em presença de chamadas de serviços da API com erros que emulam falhas que podem estar latentes em novos modelos introduzidos na simulação.

7.4.1.1. Cenário de Instanciação

Os seguintes passos foram realizados para criar uma instância do benchmark de robustez concreto.

1) Objetivos do Benchmark

As métricas escolhidas para o atributo Robustez foram: (i) *RFTCas*, taxa das falhas catastróficas que são falhas que terminam a simulação, parando o processo *RTIExec*, a federação ou o sistema operacional; (ii) *RFTRes,* taxa de falhas que exigem reinicialização da RTI ou de toda a federação, ou seja, falhas que exigem a reinicialização da infraestrutura de simulação depois de um travamento; (iii) *RFTAbort*, taxa de falhas que causam a parada abrupta dos federados sem que haja o acionamento de nenhum mecanismo de tratamento de exceção; (iv) *RFTSilent*, taxa de falhas que não são detectadas; e (v) *RFTHinder*, taxa de falhas que são tratadas pela exceção genérica da RTI (RTI - *InternalError*) e não pela exceção correta para a falha em questão. O Apêndice B apresenta detalhes das métricas utilizadas.

2) Cenário Base

Na instanciação do *Cenário Base* nós utilizamos a mesma carga de trabalho e as mesmas condições operacionais consideradas para o Estudo de Caso 1 (Tabela 7.2).

3) Cenários de Mudanças

Os tipos de dados considerados na metodologia Ballista (KOOPMAN; DEVALE, 1999) foram estendidos para incorporar os tipos de dados específicos dos parâmetros da API HLA.

As falhas injetadas nos parâmetros dos serviços foram especificadas usando valores excepcionais ou valores limites, tais como: (i) valores nulos e vazios; (ii) caracteres especiais (por exemplo, *escapes* em uma *string*); (iii) valores que podem causar um estouro nos tipos (por exemplo, exceder o tamanho de uma string); e (iv) valores máximos e mínimos (p. ex. somar um ao valor máximo de um tipo).

Em relação aos tipos específicos da API HLA, foram utilizados objetos não inicializados e, quando possível, erro nos construtores desses objetos. Para os tipos complexos (sets, vetores etc.) foram utilizados estouro no tamanho dos vetores e índices inválidos.

Alguns serviços não puderam ser testados apropriadamente porque colocavam a federação em um estado lógico inconsistente (*requestFederationSave* e *requestFederationRestore*), e também não foram testados os serviços *Data Distribution Management* por não estarem implementados em uma das RTIs avaliadas. Foram testados 73 serviços da API e a injeção de erros nos parâmetros geraram 2323 chamadas com falha. A Tabela 7.19 apresenta os operadores de mudança utilizados.

Tabela 7.19 - Estudo de Caso 3: Operadores de Mudança.

Classe: Ambientais [Subclasse: Falhas de Software]		
#	**Operador**	**Descrição**
C31	FRTIVer	Mudança na versão da *RTILib*.
C31	FIntFault	Falha de Interface injetada por meio de erros nos tipos de dados dos parâmetros dos serviços da API.

Como parâmetros da mudança foram utilizados: a versão da RTI, o arquivo de configuração de erros em tipos de dados da API HLA e o número de falhas aplicadas. A Tabela 7.20 detalha os parâmetros utilizados.

Tabela 7.20 - Estudo de Caso 3: Instanciação dos Parâmetros de Mudança.

Parâmetros das Mudanças					
Operador	**Parâmetro**	**Tipo**	**Função/Valor**	**Mínimo**	**Máximo**
FRTIVer	rtiVersion	geração/execução	RTI1.3	NA	NA
FIntFault	failFile	geração/execução	fail_rti	NA	NA
	nRounds	execução	nRounds +=1	1	(total de falhas)

O experimento foi especificado de forma a permitir que se observasse o efeito cumulativo de chamadas com falha de interface e considerou a inserção de mais de uma falha por rodada. Neste estudo de caso o gatilho da mudança é determinado pelo tempo e a cada 40 segundos um modelo com falha foi injetado na federação. O valor do gatilho e a cardinalidade da falha foram dirigidos pelo tempo de cada rodada, que neste caso não poderia ser muito extenso dado o número de falhas a serem injetadas. Por esse mesmo motivo, nós optamos por repetir a injeção da mesma falha uma única vez, com um tempo de recuperação entre a injeção de uma falha e outra. Não seria possível um número maior na cardinalidade das falhas já que isso exigiria rodadas longas que inviabilizariam a execução do experimento. Os atributos da mudança foram definidos conforme especificados pela Tabela 7.21.

Tabela 7.21 - Estudo de Caso 3: Atributos das Mudanças.

Atributos das Mudanças	
Atributos	**Valores**
Gatilho	40 s
Duração	NA
Cardinalidade	2
Tempo de recuperação	20 s

A geração dos federados com mudanças foi realizada utilizando os serviços definidos na Tabela B.17 para o operador *FIntFault*. Foi gerado um federado que segue o template apresentado no Apêndice A para cada serviço da API. Os federados com falha foram instanciados por um federado Instanciador (ver arquitetura da carga de trabalho, Figura 5.5) de acordo com os valores especificados para cardinalidade e gatilho.

Foram considerados cenários compostos do tipo *cascata*, no qual a mudança base é a versão da RTI (*rtiVersion*). Só executamos o experimento para a

versão 1.3 da biblioteca RTI HLA (RTI 1.3) já que esta era a única versão comum às duas implementações HLA avaliadas.

4) Execução do Experimento

Foi executada uma rodada de referência na qual o federado injetor instanciava federados corretos e, em seguida, foram executadas as rodadas com mudanças nas quais a cada rodada era instanciado um federado que introduzia uma falha em um parâmetro do serviço testado obedecendo aos valores definidos para os atributos gatilho e cardinalidade. Ao final de cada rodada, foi executado um *script* de recuperação de mudança no qual todos os processos remanescentes de falhas injetadas eram terminados.

As rodadas de referência e as rodadas com falhas seguiram a especificação do procedimento do benchmark descrito na definição do benchmark (Figuras 5.9 e 5.10). A interface da API HLA é bastante extensa e isso nos forçou a considerar rodadas curtas, de aproximadamente dois minutos cada, para não ter impacto no tempo total de aplicação do benchmark. Foi considerado um tempo de estabilização para sincronização dos modelos da federação e para recuperação da falha. A Tabela 7.22 apresenta os valores utilizados na execução do experimento.

Tabela 7.22 - Estudo de Caso 2: Parâmetros de Execução.

Parâmetros de Execução	
Parâmetro	**Valores**
Número de rodadas	Total de falhas (rodada de referência 1%)
Duração da rodada	2 min 10 s
Duração Inter-rodadas	20 s
Estabilização	15 s

7.4.1.2. Definição de Aspectos de Validação do Benchmark

Como o experimento considerava um número grande de falhas, foram executadas apenas duas repetições. O índice de variação deveria estar próximo de zero, pois é esperado que o sistema se comporte da mesma forma

diante de cada chamada inválida. A duração esperada do experimento é de dez dias.

7.4.1.3. Representação do Benchmark

Foi definido um *Cenário de Instanciação* para o atributo Robustez representado em um arquivo RBDL único.

7.4.1.4. Resultados dos Experimentos

Os resultados deste estudo de caso estão apresentados de forma tabular de acordo com as especificações de cada medida de robustez e em um gráfico comparativo para as duas implementações. A Tabela 7.23 e a Figura 7.14 sumarizam estes resultados.

Tabela 7.23 - Estudo de Caso 3: Resultados - Robustez.

Resultados				
Métrica	**RTI_OP1**		**RTI_OP2**	
	Falhas	**Taxa**	**Falhas**	**Taxa**
RFTCas	21	0.90	0	0
RFTRes	0	0	0	0
RFTAbort	493	21.22	409	17.61
RFTSilent	419	18.04	223	9.6
RFTHinder	476	20.49	331	14.25

ROBUSTEZ

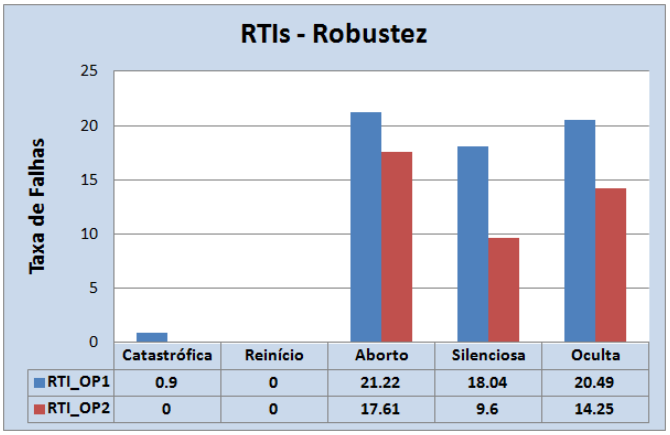


Figura 7.14 - Estudo de Caso3: Resultados - Benchmark de Robustez.

A partir dos resultados pode-se observar que a implementação RTI_OP1 mostrou-se menos robusta considerando qualquer uma das métricas. A implementação, na versão testada, apresentou 21 falhas catastróficas que pararam a RTI. Essas falhas catastróficas ocorreram em serviços do Gerenciamento de Propriedade quando um dos parâmetros apresentava valor espúrio, e não foram detectadas por nenhuma exceção prevista, tendo sido repassadas ao processo da *RTIExec*, parando a simulação. O mesmo ocorreu para o serviço *destroyFederationExecution*.

Outra observação feita na execução do experimento é que, apesar da implementação RTI_OP2 ter taxa zero para as falhas catastróficas e ter apresentado resultados melhores em praticamente todas as métricas, ela se mostrou instável tendo parado a simulação durante a injeção de duas falhas diferentes, fato que não se reproduziu durante as repetições do experimento. Entretanto, de maneira geral, a implementação RTI_OP2 mostrou-se mais robusta apresentando um mecanismo mais eficiente de tratamento de exceção, já que em 1360 falhas (58,54%), contra 914 (39,35%) da implementação RTI_OP1, ela apresentou o comportamento esperado, ou seja, tratou as exceções de forma correta e com exceções específicas como mostra Azevedo et al. (2013).

7.4.2. Validação do Benchmark

A seguir é apresentada a validação do benchmark de robustez em relação às propriedades de escalabilidade/portabilidade, representatividade, repetibilidade, simplicidade de uso e não-intrusão.

7.4.2.1. Escalabilidade/Portabilidade

Relativamente à propriedade escalabilidade, valores de parâmetros que representem diferentes escalas da carga de trabalho podem ser utilizados e serão suficientes para demonstrar a escalabilidade do experimento. No entanto, em um benchmark de robustez, esse fator não deve alterar os resultados do experimento. Relativamente à portabilidade, é importante

ressaltar que o benchmark pode considerar a robustez sob o ponto de vista da resiliência da API quando em face de mudanças tecnológicas: versão, linguagem e sistema operacional, podendo incorporar as questões de portabilidade ao próprio experimento. A propriedade de portabilidade pode ser atendida por meio de Cenários de Mudança compostos que incorporem mudanças tecnológicas, mas isso não foi implementado no estudo de caso apresentado.

7.4.2.2. Representatividade

Conforme apresentado, um simulador de satélite, para ter seu uso adaptável às várias fases de uma missão espacial e a diferentes missões, deve ter uma infraestrutura padronizada que possa ser utilizada por diferentes modelos, criando novos simuladores (EICKHOFF, 2009; ECSS, 2010). Essa filosofia de reúso motivou a avaliação da infraestrutura HLA em presença de falhas que podem estar latentes em novos modelos inseridos em uma simulação.

Como apresentado em Durães e Madeira (2003, 2006), estudos de campo mostraram que falhas de interface representam aproximadamente 6% das falhas latentes em softwares entregues. Entretanto, como o uso do padrão HLA pelos modelos está centrado principalmente no uso da API, nós consideramos que erros de interface são bastante importantes e avaliamos os mecanismos de tratamento de exceção exercitando o conjunto de serviços oferecidos.

Foi considerado o conjunto quase completo de serviços da API (as exceções já foram apontadas) e as falhas injetadas nos parâmetros da API usaram uma extensão dos tipos de dados preconizados pela técnica usada na ferramenta Ballista (KOOPMAN; DEVALE, 1999), amplamente utilizada.

7.4.2.3. Repetibilidade

O experimento foi reaplicado duas vezes apresentando o mesmo resultado. Exceção para a implementação RTI_OP2 que parou a execução em duas ocasiões, tendo sido necessário recomeçar a partir de alguns serviços

anteriores ao da parada, entretanto, o comportamento não se repetiu. Afora isso, os resultados, tal qual esperado, foram idênticos.

7.4.2.4. Simplicidade de Uso

Nós utilizamos a ferramenta de benchmark para a geração da federação e dos federados com falha. A aplicação do experimento também foi automática. O experimento foi executado em tempo pouco maior do que o estabelecido em função das falhas catastróficas que exigiram a intervenção do operador e do comportamento intermitente da RTI_OP2 em duas situações. O tempo do experimento foi considerado bastante longo, quatorze dias.

7.4.2.5. Não-Intrusão

Nós consideramos que o benchmark é não-intrusivo pois o experimento foi realizado por meio da inserção de federados que reproduziam erros típicos nas chamadas dos serviços, mas que usavam a API HLA de forma padronizada. Nenhuma modificação foi feita nas implementações e, usando a mesma abordagem aplicada nos Estudos de Caso 1 e 2, também nesse estudo de caso a configuração das duas RTIs avaliadas foi a mais padronizada possível.

7.5. Lições Aprendidas

Por meio dos Estudos de Caso apresentados nós avaliamos a aplicação da metodologia definida, dos elementos especificados, da RBDL, bem como aspectos da execução dos experimentos e da implementação das ferramentas de benchmark.

a) Métricas de resiliência utilizadas

Os índices de resiliência representaram satisfatoriamente a variabilidade das medidas dos atributos de dependabilidade Latência e Rendimento quando submetemos as RTIs HLA ao conjunto de mudanças escolhidas para os estudos de caso apresentados. Ao considerarmos o conjunto de mudanças, seria suficiente analisarmos os índices de resiliência e a medida de referência para o atributo, sem que fosse necessário analisar os resultados detalhados

para cada mudança. Assim, apenas analisando o índice de resiliência é possível identificar qual mudança tem maior impacto na dependabilidade ou quais mudanças têm impacto aceitável. Sendo possível, por exemplo, observar que a distribuição da simulação em uma rede em ambiente controlado teria um impacto aceitável, ou ainda, observar que a RTI_COM seria a única que atenderia a um simulador centralizado com modelos que requisitassem baixa latência no intercâmbio de dados.

Ao compararmos os dois índices utilizados *RRIndex* e *RLIndex* observamos, como era esperado, que o índice de estabilidade tem uma aplicação mais genérica, podendo ser aplicado a qualquer conjunto de métricas independentemente da sua interpretação. Entretanto, ele não consegue representar a resiliência positiva, ou seja, o impacto positivo de uma execução com mudanças em relação à uma execução de referência, já que as variações nas medidas tanto positivas quanto negativas são consideradas desvios relativamente ao valor esperado. Além disso, esse índice não se aplica bem à avaliação da resiliência em relação a cada valor do parâmetro da mudança dificultando o estabelecimento de limiares aceitáveis de resiliência (*thresholds*) para valores de parâmetros. Por exemplo, poderíamos definir que dada a Latência de referência de uma RTI aceitaríamos um limite de 50% para perda de serviço em um simulador de *tempo real restrito*, assim poderia ser identificado, para aquela infraestrutura e contexto, qual o valor máximo para o parâmetro de uma dada mudança. O *RLIndex* representa bem as mudanças com impactos positivos, permite a análise dos parâmetros de mudança valor a valor e pior caso, e permite o estabelecimento de limites. Entretanto, como várias métricas têm interpretações diferentes, para algumas deseja-se obter o menor valor (p. ex. Latência), para outras o maior valor (p. ex. Rendimento), a fórmula utilizada deve estar relacionada ao objetivo da métrica do atributo e de sua interpretação. Para avaliação global das RTIs nós utilizamos o índice de perda mínima (*RLIndexMin*) para representar o pior caso.

Nos Estudos de Caso apresentados não foram aplicados Cenários de Mudança aleatórios ou sequenciais e que consideram falhas, e os índices de resiliência

utilizados foram calculados de forma independente do tempo. Métricas que levassem em consideração não apenas a estabilidade e a perda de serviço, mas também os mecanismos de recuperação e o tempo de recuperação poderiam ser utilizados nesse contexto. O trabalho de Henry e Ramirez-Marquez (2012) apresenta uma métrica genérica que leva em consideração também as medidas de um dado atributo no estado pós-mudança ou perturbação. Essa métrica poderia ser considerada ou adaptada.

b) Uso da RBDL na Geração e Condução do Benchmark

A linguagem teve capacidade de representar os diferentes aspectos de um benchmark de resiliência, incluindo atributos e métricas, cenários base, cenários de instanciação e as características das diferentes mudanças. Na aplicação dos nossos experimentos, nós utilizamos um conjunto de arquivos RBDL que tinham em comum a definição do benchmark, variando os cenários de instanciação. Esses arquivos foram usados tanto na geração da carga de trabalho e mudanças, quanto na execução do benchmark. Mostrou-se fácil a alteração de condições dos experimentos, por exemplo, aumentar a escala de um parâmetro significa alterar o valor limite no arquivo RBDL e gerar novamente as cargas. A linguagem também conseguiu representar de forma satisfatória o Benchmark de Robustez para infraestruturas HLA.

c) Implementação da Ferramenta de Benchmark

Nós utilizamos ferramentas comerciais para transformar o Esquema RBDL em Classes C++, tanto para as classes do *framework* do benchmark, quanto para as Classes do domínio HLA. Embora tenhamos usado o arcabouço gerado para a codificação das classes e herdado classes de leitura de XML (arquivos RBDL) diretamente, ainda houve esforço de desenvolvimento, especialmente para a geração automática das cargas HLA (federação HLA típica gerada a partir de *templates*). No entanto, uma vez implementadas as classes do benchmark e de geração de cargas, a inserção de serviços relativos a novas mudanças (classes de geração e execução de mudanças) foi relativamente simples, embora, o esforço necessário para implementação desses serviços

possa variar bastante em função do objetivo benchmark concreto a ser aplicado ou em função da mudança em questão.

O esforço para a geração e execução dos dois benchmarks apresentados mostrou-se diferente. Por exemplo, implementar um serviço para gerar modelos com mudanças da classe Escala, na qual parâmetros da carga de trabalho são alterados, exigiu menos esforço que gerar arquivos de configuração para os erros relativos a tipos de dados dos serviços da API HLA, já que esses arquivos são bastantes extensos.

Nem sempre será trivial medir o esforço necessário para a instanciação de um novo benchmark, mas parte do trabalho está coberta pelos elementos da metodologia e pelo *framework* da ferramenta proposta, conforme se pode observar nas estatísticas de número de linhas de código do protótipo das ferramentas desenvolvidas (geração/condução do experimento e análise de resultados), apresentadas na Tabela 7.24.

Tabela 7.24 - Estatísticas - Protótipo da Ferramenta.

Estatísticas						
Pacotes	**Ferramenta de Geração**		**Ferramenta de Análise**		**GERAL**	
	número de linhas	**%**	**número de linhas**	**%**	**número de linhas**	**%**
BENFramework	2925	21,10%	715	45,11%	3640	23,57%
BENHla	1119	8,07%	870	54,89%	1989	12,88%
HLA	3692	26,64 %			3692	23,90%
LOADLib	1512	10,91%			1512	9,79%
ModelLib	1997	14,41%			1997	12,93%
Templates	2615	18,87%			2615	16,93%
TOTAL	13860	100%	1585	100%	15445	100%

É importante frisar que embora os *templates* de federados sejam extensos no que concerne ao número de linhas, cada novo *template* apenas altera poucas características do *template* base.

7.6. Considerações Finais

Os estudos de caso apresentados tiveram o objetivo de demonstrar a aplicabilidade da metodologia, bem como o uso dos elementos previamente especificados e da RBDL na instanciação de benchmarks concretos, e, também, de apresentar os resultados obtidos na avaliação de implementações HLA.

Outro aspecto considerado foi a extensibilidade das ferramentas de benchmark quanto à inserção de novas mudanças, objetivos e alterações de contexto. Durante a geração e execução dos estudos de caso, instanciar um novo benchmark significou, no domínio das infraestruturas HLA, implementar os serviços específicos de geração, execução e análise dos resultados para cada novo cenário de instanciação.

Os dois benchmarks tiveram objetivos, projetos e mesmo regras de aplicação diferentes, mas foi possível representá-los por meio da RBDL, utilizar a metodologia para instanciá-los e estender a ferramenta de benchmarking para gerá-los e executá-los.

8 CONCLUSÃO E TRABALHO FUTURO

Simuladores são amplamente usados na Engenharia Espacial como suporte ao projeto de sistemas espaciais com o objetivo de reduzir prazos, custos e riscos. Entretanto, simuladores são sistemas complexos e, por isso, adotar filosofias de reúso na sua concepção, utilizar infraestruturas padronizadas no seu desenvolvimento e empregá-los em várias fases de uma missão espacial ou entre várias missões são formas de viabilizar e promover o uso desses sistemas.

Esta tese considerou o uso de infraestruturas de simulação baseadas no padrão HLA para o desenvolvimento de simuladores de satélite e especificou uma abordagem de benchmarking para avaliar e comparar essas infraestruturas no que tange a atributos de dependabilidade e resiliência.

Neste Capítulo são apresentadas as principais contribuições deste trabalho, as suas limitações e as perspectivas futuras.

8.1. Principais Contribuições

Esta tese contribuiu na área de Engenharia Espacial com a especificação de uma abordagem de benchmarking de resiliência que fornece subsídios para a escolha de produtos HLA procedentes de diferentes fabricantes e para a avaliação de produtos previamente utilizados quando em face de novos contextos.

As principais contribuições deste trabalho foram:

- A definição de uma **metodologia de especificação de benchmarks de resiliência** que sistematiza o processo de benchmarking, indicando as etapas e passos a serem seguidos na **definição** de elementos do benchmark (atributos avaliados, métricas, cargas de trabalho, etc.), na **instanciação** de benchmarks concretos e na **representação** tanto dos elementos definidos, quanto dos benchmarks instanciados. A metodologia foi proposta para especificar benchmarks de resiliência no domínio de

infraestruturas HLA, mas pode ser adaptada a outros tipos de benchmarks e a domínios similares.

- A especificação de uma **abordagem de benchmarking de resiliência**, baseada na metodologia proposta, que definiu e validou um conjunto de elementos que podem ser utilizados em diferentes benchmarks concretos. A partir da definição desses elementos, nós mostramos que é possível derivar benchmarks capazes de avaliar a resiliência de infraestruturas HLA relativamente a atributos de dependabilidade específicos quando em presença de um subconjunto de uma carga de mudanças sistematizada e pré-definida.

- A definição de uma **Linguagem de Descrição de Benchmark de Resiliência** (RBDL) que é uma alternativa para a representação e disseminação de benchmarks de resiliência no domínio de infraestruturas HLA. A RBDL promove a padronização, facilita a comunicação e possibilita o uso direto da especificação do benchmark tanto na sua implementação e execução, quanto na análise dos resultados dos seus experimentos. A linguagem pode ser usada para representar vários tipos de benchmarks e fornece pontos de extensão que permitem a sua adaptação a diferentes domínios.

- A proposta de um ***framework* e de uma arquitetura baseados na RBDL para a implementação do ferramental de benchmark de resiliência**. O *framework* proposto está dividido entre uma parte geral que representa benchmarks de resiliência e uma parte específica que implementa o domínio de infraestruturas de simuladores baseadas HLA. Com o objetivo de flexibilizar a implementação de novos benchmarks concretos e considerando o aspecto imprevisível das mudanças, o *framework* foi concebido visando facilitar a incorporação de novos objetivos ou tipos de mudanças, tanto na geração/execução do benchmark, quanto na análise dos resultados. Além disso, a partir do *framework* para benchmarks de

resiliência, a arquitetura e projeto do ferramental podem ser facilmente estendidos a outros domínios.

- A instanciação de benchmarks de resiliência concretos e de um benchmark de robustez que demonstraram o uso dos elementos previamente especificados e a viabilidade da aplicação da metodologia nos experimentos de benchmarking. Os resultados do benchmark de resiliência indicaram que a implementação comercial é mais resiliente e eficiente que as demais para grande parte das mudanças aplicadas, mas que, no entanto, uma das implementações de código aberto é suficientemente resiliente e eficiente e poderia ser considerada como alternativa no contexto dos projetos de simuladores de satélite do INPE. Os resultados do benchmark de robustez mostraram que uma das implementações é mais robusta sob o ponto de vista da ausência de falhas catastróficas e da capacidade dos mecanismos de tratamento de exceção, mas, apesar disso, essa implementação mostrou-se mais instável e foi apontada como a menos resiliente nos demais experimentos. De maneira geral, os resultados dos experimentos de benchmark foram capazes de expressar quantitativamente, de forma simples e eficaz, a resiliência e robustez das implementações HLA e de fornecer diretrizes para a escolha de implementações em diferentes contextos de avaliação e mudanças.

Neste trabalho nós consideramos que as questões relativas às mudanças serão sempre recorrentes e permanentes, por esse motivo, estabelecemos um paradigma que parte da possibilidade de instanciação de novos benchmarks, priorizando a flexibilidade na especificação, representação e implementação dos mesmos, visando facilitar tanto a inserção de novas mudanças, quanto o atendimento a novos objetivos.

8.2. Contribuições no Contexto do INPE

O INPE desenvolve e utiliza simuladores com diferentes propósitos. O Laboratório de Sistemas Inerciais para Aplicação Aeroespacial (LABSIA), centrado em simulação, realiza testes de qualificação de sistemas de controle e

atitude de satélites. Para atender as necessidades operacionais da família de satélites CBERS, encontra-se em desenvolvimento um simulador operacional. E, recentemente, para o satélite Amazônia-1, foi adquirido um simulador de verificação e validação do computador de bordo. Além disso, no âmbito da Coordenação de Engenharia e Tecnologia Espacial, a área de pesquisa em simuladores tem por objetivo prospectar novas tecnologias, soluções e usos de simuladores que possam contribuir com as missões do INPE.

Relativamente aos simuladores em desenvolvimento no INPE, este trabalho pode ser diretamente utilizado na prospecção da distribuição desses simuladores por meio de infraestruturas HLA, bem como para auxiliar na escolha das implementações mais adequadas para este fim. Futuramente, o trabalho poderá ser estendido para análise da viabilidade do uso remoto de recursos dos laboratórios de simulação no desenvolvimento de novos simuladores distribuídos.

8.3. Limitações da Proposta

A metodologia de especificação e disseminação de benchmarking proposta neste trabalho pode ser aplicada a outros domínios e os elementos elencados podem ser facilmente adaptados a domínios correlatos. Entretanto, benchmarks de resiliência são dependentes do domínio do problema e, desta forma, uma das principais restrições deste trabalho é ter a sua aplicação direta limitada a infraestruturas de simulação baseadas no padrão HLA.

Embora a metodologia tenha especificado uma carga de mudanças abrangente e a ferramenta de benchmark seja extensível permitindo a inclusão de novas mudanças, para cada nova mudança identificada, sempre haverá o esforço de especificação e implementação dos métodos de geração e execução, incluindo a adaptação ou uso de ferramentas de injeção de falhas. Neste trabalho nós não esgotamos nem o conjunto de atributos de dependabilidade considerados relevantes no contexto de simuladores, nem o conjunto completo de mudanças às quais o sistema estaria potencialmente exposto.

Além disso, os benchmarks concretos apresentados tiveram como objetivo principal mostrar a viabilidade do uso da metodologia de especificação de benchmarks de resiliência e dos elementos definidos na nossa abordagem de benchmarking para infraestruturas HLA. Embora eles tenham fornecido resultados que podem ser usados para avaliar e escolher RTIs HLA, o benchmark teria que ter a sua aceitação confirmada por meio do seu uso pela comunidade ou em projetos práticos.

Foi também limitação deste trabalho o número de implementações HLA avaliadas, duas em cada estudo de caso. Seria interessante que os benchmarks concretos fossem aplicados usando outras implementações. Além disso, a RTI comercial avaliada também tinha limitações de licença, permitindo a execução de dez federados apenas.

8.4. Trabalho Futuro

A seguir é apresentado o conjunto de perspectivas que vislumbramos como sequência a este trabalho:

- A carga de mudanças e as métricas definidas no contexto deste trabalho permitem que se estenda o benchmark de robustez apresentado no Estudo de Caso 3 para um benchmark de resiliência com foco na robustez. Para tanto, é necessário que se associe, usando cenários compostos, as falhas de interface tal como foram implementadas a mudanças específicas, tais como versões da biblioteca RTI, mudanças de sistema operacional, etc. Isso permitirá avaliar a variabilidade da robustez das RTIs por meio de métricas de resiliência específicas.

- A avaliação de outros atributos relevantes, incluindo os atributos *acurácia, estabilidade e agendabilidade,* e a injeção de falhas em modelos reais usando técnicas de mutantes seria bastante interessante já que uma das questões no uso de infraestruturas de simulação em um contexto evolutivo é a inserção de novos modelos desenvolvidos por diferentes equipes, com níveis distintos de qualidade.

- Mudanças relacionadas a falhas requererem a avaliação das infraestruturas HLA relativamente aos mecanismos de recuperação e ao tempo de recuperação de falhas. Assim, a proposta, adaptação ou uso experimental de outras métricas de resiliência pode ser importante para a instanciação de novos benchmarks de resiliência concretos.

- A metodologia especificada, bem como partes do benchmarking especificado podem ser adaptados para aplicação a outros padrões de infraestrutura de simulação como o SMP (ECSS, 2011), auxiliando na avaliação de produtos internos e externos. Esse padrão, definido pela ECSS, é atualmente utilizado em apenas algumas plataformas de simulação, mas a partir do seu amadurecimento e por meio do incentivo do seu uso em produtos que apoiam o processo de engenharia da ESA, tende a ser implementado por outros fabricantes ou agências. Além disso, como um benchmark de resiliência também pode ser usado para medir a adequação das infraestruturas quando em face de novos modelos ou tecnologias, o benchmarking proposto, com adaptações, pode ser usado para avaliar infraestruturas proprietárias.

- O trabalho proposto pode ser estendido a diversas outras áreas de conhecimento, tais como aeronáutica, de manufatura, etc., uma vez que uso de simuladores e ambientes de simulação não se confina à área espacial.

- Enquanto benchmarks de desempenho são bastante comuns na área computacional, a definição de uma abordagem de benchmarking que leva em consideração as mudanças às quais um sistema estará sujeito e que avalia atributos de dependabilidade e resiliência compõe uma área de pesquisa que pode ser estendida a outros domínios, particularmente para a avaliação de sistemas críticos. Por exemplo, na área espacial, o trabalho poderia ser adaptado para avaliar a resiliência de sistemas operacionais de tempo real para sistemas embarcados usados em computadores de bordo.

Finalmente, a metodologia de especificação, a linguagem de representação de benchmarking e o projeto do ferramental proposto podem ser generalizados

para abranger diferentes tipos de benchmarks para domínios correlatos e podem ser um primeiro passo rumo a uma metodologia genérica de desenvolvimento, representação e disseminação de benchmarks de resiliência.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, R.; VIEIRA, M. Benchmarking the resilience of self-adaptive software systems: perspectives and challenges. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTATIVE AND SELF-MANAGING SYSTEMS (SEAMS '11), 6., 2011, Honolulu. **Proceedings…** New York: ACM, 2011, p. 190-195.

ALMEIDA, R.; VIEIRA, M. Changeloads for resilience benchmarking of self-adaptive systems: a risk-based approach. In: DEPENDABLE COMPUTING CONFERENCE (EDCC), 9., 2012, Sibiu, Romênia. **Proceedings…** Washington: IEEE Computer Society, 2012a. p. 173-184.

ALMEIDA, R.; VIEIRA, M. Changeloads: a fundamental piece on the SASO systems benchmarking puzzle. In: SELF-ADAPTIVE AND SELF-ORGANIZING SYSTEMS WORKSHOPS (SASOW), 6., 2012, Lyon, França. **Proceedings…** IEEE Sixth International Conference, 2012b. p. 93-96.

ALMEIDA, R.; ARAUJO NETO, A.; VIEIRA, M. SCoRe: An across-the-board metric for computer systems resilience benchmarking. In: DEPENDABLE SYSTEMS AND NETWORKS WORKSHOP (DSN-W), 43., 2013, Budapeste, Hungria. **Proceedings…** IEEE/IFIP, 2013. p. 1-8.

AMBROSIO A. M.; CARDOSO, P. E.; ORLANDO, V., BIANCHI. N. Brazilian satellite simulators: previous solutions trade-off and new perspectives for the CBERS program. In: CONFERENCE ON SPACE OPERATIONS (SAPCEOPS), 8., 2006, Roma. **Proceedings…** SAPCEOPS, 2006.

ANDRIOTTI, J. L. S. **Técnicas estatísticas aplicáveis a tratamento de informações oriundas de procedimentos laboratoriais**. Porto Alegre, CPRM. Superintendência Regional de Porto Alegre, 2005. p. 28-29.

AVIZIENIS, A.; LAPRIE, J. C.; RANDEL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v.1, n.1, p. 11-33, 2004.

AZEVEDO, D. N. R.; HOFFMANN, L. T.; AMBROSIO, A. M.; PERONDI, L. F. Analysis of the Simulation Model Platform adoption in the context of INPE simulators. In: SIMULATION AND EGSE FACILITIES FOR SPACE PROGRAMMES WORKSHOP (SESP), 2012, Noordwijk, Holanda. **Proceedings…** SESP, 2012.

AZEVEDO, D. N. R.; AMBROSIO, A. M.; VIEIRA, M. HLA middleware robustness and scalability evaluation in the context of satellite simulators. In: PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING (PRDC), 19., 2013, Vancouver, Canada. **Proceedings…** Washington: IEEE Computer Society, 2013. p. 312-317.

AZEVEDO, D. N. R.; AMBROSIO, A. M.; VIEIRA, M. Towards a Resilience Benchmarking Description Language for the context of satellite simulators. In: EUROPEAN DEPENDABLE COMPUTING CONFERENCE (EDCC), 10., 2014, Newcastle upon Tyne, Reino Unido. **Proceedings…** Washington: IEEE Computer Society, 2014. p. 194-197.

BARBOSA, R.; KARLSSON, J.; MADEIRA, H.; VIEIRA, M. Fault injection. In: WOLTER, K.; AVRITZER, A.; VIEIRA, M.; VAN MOORSEL, A. (eds.). **Resilience Assessment and Evaluation of Computer Systems**. Springer-Verlag Berlin Heidelberg, 2012. cap. 13, p. 263-281.

BARKER, K.; RAMIREZ-MARQUEZ, J. E.; ROCCO C. M. Resilience-based network component importance measures. **Reliability Engineering and System Safet**y, v. 117, n.1, p. 89-98, 2013.

BASILI, V.; WEISS, D. M. A methodology for collecting valid software engineering data. **IEEE Transactions on Software Engineering,** n. 6, p. 728-738, 1984.

BASILI, V.; DONZELLI, P. ; ASGARI, S. A unified model of dependability: capturing dependability in context. **Software Engineering IEEE Transactions**. v. 21, n. 6, p. 19-25, 2004.

BOEHM, B.; HUANG, L.; JAIN, A.; MADACHY, R. **The nature of information system dependability: a stakeholder/value approach**. 2004. USC-CSSE Technical Report.

BÖHME, R.; REUSSNER, R. Validation of predictions with measurements. In: EUSGELD, I.; FREILING, F.; REUSSNER, R. (eds.). **Dependability Metrics**. Springer Berlin Heidelberg, 2008. cap. 3, p. 14-18.

BONDAVALLI, A. et al. Research Roadmap Deliverable D3.2, **AMBER – Assessing, Measuring and Benchmarking Resilience**. Funded by European Union, 2009. (IST-216295).

BOVEE, M.; KOGAN, A.; NELSON, K.; SRIVASTAVA, R. P. ; VASARHELYI, M. A financial reporting and auditing agent with net knowledge (FRAANK) and extensible business reporting language (XBRL). **Journal of Information Systems,** v. 19, n. 1, p. 19-41, 2005.

CALZAROSSA, M.; SERAZZI, G. Workload characterization: a survey. **Proceedings of the IEEE**, v. 81, n. 8, p. 1136-1150, 1993.

CAMARA, J.; DE LEMOS, R.; VIEIRA, M.; ALMEIDA, R.; VENTURA, R. Architecture-based resilience evaluation for self-adaptive systems. **Computing**. v. 95, n. 8, p. 689-722, 2013a.

CAMARA, J.; DE LEMOS, R.; LARANJEIRO, N.; VENTURA, R.; VIEIRA, M. Robustness evaluation of controllers in self-adaptive software systems. In: DEPENDABLE COMPUTING (LADC), 16., 2013, Rio de Janeiro, Brasil. **Proceedings…** Washington: IEEE Computer Society, 2013b. p. 1-10.

CAMARA, J.; CORREIA, P. ; DE LEMOS, R.; VIEIRA, M. Empirical resilience evaluation of an architecture-based self-adaptive software system. In: ACM SIGSOFT CONFERENCE ON QUALITY OF SOFTWARE ARCHITECTURES (QoSA, part of CompArch 2014), 10., 2014, Lille, França. **Proceedings…** ACM, 2014, p. 63-72.

CHAUDRON, J.; NOULARD, E.; SIRON, P. Design and modeling techniques for real-time RTI time management. In: THE SPRING SIMULATION INTEROPERABILITY WORKSHOP, 2011, Boston, Massachusetts, EUA. **Proceedings…**SISO, 2011.

CHILLAREGE, R. et al. Orthogonal defect classification: a concept for in-process measurements. **IEEE Transactions on Software Engineering**, v. 18, n. 11, p. 943-956, 1992.

CHILLAREGE, R.; BASSIN, K. A. Software triggers as a function of time-ODC on field faults. **Dependable Computing and Fault Tolerant Systems**, v. 10, p. 327-342, 1998.

COORDENADORIA DE ENGENHARIA E TECNOLOGIA ESPACIAL (ETE). **Curso de Tecnologia de Satélites**, INPE, 1999.

COSTA, D.; BARBOSA, R.; MAIA, R.; MOREIRA, F. DeBERT: Dependability benchmarking of embedded real-time off-the-shelf components for space applications. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability benchmarking for computer systems**. New Jersey: John Wiley & Sons, Inc., 2008. cap. 13, p. 255-283.

COSTA, P. ; SILVA, J. G.; MADEIRA, H. Dependability benchmarking using software faults: how to create practical and representative faultloads. In: IEEE PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING, 15., 2009, Shanghai, China. **Proceedings…**, Washington: IEEE Computer Society*,* 2009, p. 289-294.

DAWSON S.; JAHANIAN, F.; MITTON, T.; TUNG, T. L. Testing of fault-tolerant and real-time distributed systems via protocol fault injection. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING, 26., 1996, Washington, EUA. **Proceedings…**, Washington: IEEE Computer Society*,* 1996, p. 404–414.

DENG, L.; HAN, C.; CAO, J.; SONG, Y. Information management system for space science mission concurrent design. In: INFORMATION MANAGEMENT, INNOVATION MANAGEMENT AND INDUSTRIAL ENGINEERING (ICIII), 6., 2013. **Proceedings…**, ICIII, 2013, p. 573-576.

DEPENDABILITY BENCHMARK (DBENCH). Project Full Final Report. 2004. Disponível em: <http://webhost.laas.fr/TSF/DBench>. Acesso em: 15 jun. 2011.

DRAKE, R.; AGARWAL, P. ; ESPINOSA, M.; OWENS, C.; LIEDEL, R.; STEELE, J. Independent benchmarking of RTI real-time performance. In: FALL SIMULATION INTEROPERABILITY WORKSHOP, 2003. Orlando, EUA. **Proceedings…** SISO, 2003.

DURÃES, J. A.; MADEIRA, H. S. Emulation of software faults by educated mutations at machine-code level. In: SOFTWARE RELIABILITY ENGINEERING (ISSRE 2003), 13., 2002, Denver, Colorado, EUA. **Proceedings…** Washington: IEEE Computer Society*,* 2002, p. 329-340.

DURÃES, J. A.; MADEIRA, H. S. Definition of software fault emulation operators: a field data study. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2003, São Francisco, Califórnia, EUA. **Proceedings…** Washington: IEEE Computer Society*,* 2003, p. 105-114.

DURÃES, J. A.; MADEIRA, H. S. Generic faultloads based on software faults for dependability benchmarking. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2004, Florença, Itália. **Proceedings…** Washington: IEEE Computer Society*,* 2004, p. 285-294.

DURÃES, J. A.; MADEIRA, H. S. Emulation of software faults: a field data study and a practical approach. **IEEE Transactions on Software Engineering**, v. 32, n. 11, p. 849-867, 2006.

DURÃES, J. A.; VIEIRA, M.; MADEIRA, H. S. Dependability benchmarking of web servers. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability Benchmarking for Computer Systems**. New Jersey: John Wiley & Sons, Inc., 2008. cap. 6, p. 91-110.

EICKHOFF, J.; FALKE, A.; RÖSER, H.P. Model-based design and verification--state of the art from Galileo constellation down to small university satellites. **Acta Astronautica**, v. 61, p. 383–390, 2007.

EICKHOFF, J. **Simulating Spacecraft Systems**. Heidelberg: Springer-Verlag Berlin, 2009. 358p.

ELECTION Markup Language. In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2013. Disponível em: <http://en.wikipedia.org/w/index.php?title=Election_Markup_Language&oldid=626329444>. Acesso em: 10 mar. 2013.

ELLISON, R. J.; FISHER, D. A.; LINGER, R. C.; LIPSON, H. F.; LONGSTAFF, T. **Survivable network systems:** an emerging discipline. Carnegie-Mellon University, Software Engineering Institute, Pittsburgh, Pensilvânia, EUA, 1997. (CMU/SEI-97-TR-013).

ELNAFFAR, S.; MARTIN, P. **Characterizing Computer Systems' Workloads**. School of Computing, Queen's University, Canada, 2002. (Technical Report 2002-461).

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **ECSS-M-ST-10C:** Space project management: project planning and implementation. Noordwijk: ECSS, 2009. 50p.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **ECSS-E-TM-10-21A:** Space engineering: system modelling and simulation. Noordwijk: ECSS, 2010. 79p.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **ECSS-E-TM-40-07**: Space Engineering: Simulation modelling platform. V1. Noordwijk: ECSS, 2011. 49p.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **ECSS-S-ST-00-01C**: Glossary of terms. Noordwijk: ECSS, 2012. 63p.

FERNSLER, K.; KOOPMAN, P. Robustness testing of a distributed simulation backplane. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 10., 1999, Boca Raton, Flórida, EUA. **Proceedings…**IEEE, 1999. p. 189-198.

FIRESMITH, D. G. **Common concepts underlying safety security and survivability engineering**. Pittsburgh: Carnegie-Mellon University, Software Engineering Institute, 2003. (CMU/SEI-2003-TN-033).

FITZGIBBONS, B.; MCLEAN, T.; FUJIMOTO, R. RTI Benchmark studies. In: SPRING SIMULATION INTEROPERABILITY WORKSHOP, 2002, Orlando, EUA. **Proceedings…** SISO, 2002.

FUJIMOTO, R. **Parallel and Distributed Simulation Systems**. New York: John Wiley & Sons, 2000. 300p.

GAMMA, E.; HELM R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: elements of reusable object oriented software**. Portland: Addison-Wesley, 1994. 395p.

GRAY, J. **Benchmark handbook:** for database and transaction processing systems. Morgan Kaufmann Publishers Inc., 1992.

HENRY, D.; RAMIREZ-MARQUEZ, J. E. Generic metrics and quantitative approaches for system resilience as a function of time. **Reliability Engineering & System Safety,** v. 99, n. 0, p. 114–122, 2012.

HEWITT E. **Java SOA Cookbook**. O'Reilly, 2009.

HOFFMANN, L. T.; PERONDI, L. F. Estudo de simuladores computacionais aplicados ao ciclo de desenvolvimento de plataformas orbitais. In: WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, 1., 2010, São José dos Campos. **Anais...** IWETE, 2010.

HOLLING, C. S. Resilience and stability of ecological systems. **Annual review of ecology and systematics**, v. 4, p. 1-23, 1973.

HUPPLER, K. The art of building a good benchmark. In: NAMBIAR, R.; POESS. M. (eds.). **Performance Evaluation and Benchmarking**. Springer Berlin Heidelberg, 2009. p. 18-30.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **Std. 610.3-1989:** IEEE Standard glossary of modeling and simulation terminology, 1989. 19p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **Std. 610.121990:** IEEE Standard glossary of software engineering terminology, 1990. 84p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **IEEE 1516-2000:** IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA) - framework and rules. Nova Iorque, EUA: IEEE Press, 2001, 27p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **IEEE 1516-2010:** IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA) - framework and rules. New York, EUA: IEEE Press, 2010a. 21p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **IEEE 1516.1-2010:** IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA) - federate interface specification. Nova Iorque, EUA: IEEE Press, 2010b, 363p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **IEEE 1516.2-2010:** IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA) - object model template (OMT) Specification. Nova Iorque, EUA: IEEE Press, 2010c, 100p.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). IEC **14977:1996**: Information technology - syntactic metalanguage - extended BNF. Genebra, Suíça, 1996.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). IEC **9126-1**: Software engineering-product quality-part 1: quality model. Genebra, Suíça, 2001.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). IEC **9126-2:** Software engineering - product quality - part 2 - external metrics. Genebra, Suíça, 2002.

KANOUN, K.; CROUZET, Y.; KALAKECH, A.; REGINA, A. Windows and Linux robustness benchmark with respect to application erroneous behavior. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability Benchmarking for Computer Systems.** New Jersey: John Wiley & Sons, Inc., 2008. cap. 12, p. 227-254**.**

KAO, W. I.; IYER, R. K.; TANG, D. FINE: a fault injection and monitoring environment for tracing the Unix system behavior under faults. **IEEE Transactions on Software Engineering**, v. 19, n. 11, p. 1105-1118, 1993.

KAO, W. I.; IYER, R. K. DEFINE: a distributed fault injection and monitoring environment. In: IEEE WORKSHOP ON FAULT-TOLERANT PARALLEL AND DISTRIBUTED SYSTEMS, 1994. **Proceedings…** Washington: IEEE Computer Society*,* 1994. p. 252-259.

KNIGHT, P. ; CORDER, A.; LIEDEL, R.; JENKINS, C.; DRAKE, R.; AGARWAL, P. ; NUNEZ, E. Independent throughput and latency benchmarking for the evaluation of RTI implementations. In: FALL SIMULATION INTEROPERABILITY WORKSHOP, 2001, Orlando, Flórida, EUA. **Proceedings…** SISO, 2001.

KNIGHT, P. ; CORDER, A.; LIEDEL, R.; GIDDENS, J.; DRAKE, R.; JENKINS, C.; AGARWAL, P. Evaluation of run time infrastructure (RTI) implementations. In: HUNTSVILLE SIMULATION CONFERENCE, 2002, Huntsville, Alabama, EUA. **Proceedings…** SCS, 2002a.

KNIGHT, P. et al. Analysis of independent throughput and latency benchmarks for multiple RTI implementations. In: FALL SIMULATION INTEROPERABILITY WORKSHOP, 2002, Orlando, Flórida, EUA. **Proceedings…** SISO, 2002b.

KNIGHT, P.; LIEDEL, R.; KLINNER, M. WBT RTI Independent benchmark tests: design, implementation and updated results. In: SPRING SIMULATION INTEROPERABILITY WORKSHOP, 2002, Orlando, Flórida, EUA. **Proceedings…** SISO, 2002c.

KOOPMAN, P; DEVALE, J. Comparing the robustness of POSIX operating systems. In: INTERNATIONAL SYMPOSIUM ON FAULT-TOLERANT COMPUTING**,** 29., 1999, Madison, Wisconsin, EUA. **Proceedings…** Washington: IEEE Computer Society, 1999. p. 30-37.

KOOPMAN, P.; DEVALE, K.; DEVALE, J. Interface robustness testing: experience and lessons learned from the Ballista project. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability benchmarking for computer systems.** New Jersey: John Wiley & Sons, Inc., 2008. cap. 11, p. 201-226**.**

KOSKI, D.; LEE, C. P.; MAGANTY, V.; MURTHY, R.; NATARAJAN, A.; STEIDL, J. **Fuzz revisited:** a re-examination of the reliability of UNIX utilities and services. Wisconsin: University of Wisconsin-Madison, Computer Sciences Department, 1995.

LAPRIE, J. C. Resilience for the scalability of dependability. In: IEEE INTERNATIONAL SYMPOSIUM ON NETWORK COMPUTING AND APPLICATIONS, 4., 2005, Cambridge, Reino Unido. **Proceedings...** Washington: IEEE Computer Society, 2005. p. 5-6.

LAPRIE, J. C. From dependability to resilience. In: IEEE INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORK (DSN), 38., 2008, Anchorage, EUA. **Proceedings...** Washington: IEEE Computer Society, 2008.

LARANJEIRO, N.; VIEIRA, M.; MADEIRA, H. S. Experimental robustness evaluation of JMS middleware. In: INTERNATIONAL CONFERENCE ON SERVICES COMPUTING (SCC), 2008, Honolulu, Havaí, EUA. **Proceedings...** Washington: IEEE Computer Society, 2008. p. 119-126.

LIU, C.; WANG, Y.; JIN, Z. Elicit the requirements on software dependability: a knowledge-based approach software. In: ENGINEERING CONFERENCE (APSEC), 16., 2009, Penang, Malásia. **Proceedings...** APSEC, 2009, p. 233 - 240.

MADNI, A. M.; JACKSON, S. Towards a conceptual framework for resilience engineering. **Systems Journal, IEEE**, v. 3, n. 2, p. 181-191, 2009.

MALINGA, L., LE ROUX, W. H. HLA RTI performance evaluation. In: SISO EUROPEAN SIMULATION INTEROPERABILITY WORKSHOP, 2009, Istambul, Turquia. **Proceedings …** 2009. p. 1-6.

MARSDEN, E.; FABRE, J. C.; ARLAT, J. Dependability of CORBA systems: service characterization by fault injection. In: SYMPOSIUM IN RELIABLE DISTRIBUTED SYSTEMS, 21., 2002, Osaka, Japão. **Proceedings...** Washington: IEEE Computer Society, 2002. p. 276-285.

MARTIN, A. C. M.; CARVALHO, M. M**. Avaliação do uso da simulação virtual no processo de desenvolvimento de produtos**. Dissertação (Mestrado Profissional em Engenharia Automotiva) - Escola Politécnica da USP, São Paulo, 2005.

MARTIN, E.; BASU, S.; XIE, T. Websob: a tool for robustness testing of web services. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 29., 2007, Minneapolis, Minnesota, EUA. **Proceedings...** Washington: IEEE Computer Society, 2007. p. 65-66.

MARTINS, E.; RUBIRA, C.M.F.; LEME, N.G.M. Jaca: a reflective fault injection tool based on patterns. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2002, Bethesda, Maryland, EUA. **Proceedings...,** Washington: IEEE Computer Society, 2002. p. 483-487.

MARTINS, E.; MATTIELLO-FRANCISCO, M. F. A tool for fault injection and conformance testing of distributed systems. **Dependable Computing**. Springer Berlin Heidelberg, 2003. p. 282-302.

MASTEN, A. S.; BEST, K. M.; GARMEZY, N. Resilience and development: contributions from the study of children who overcome adversity. **Development and psychopathology**, v. 2, n. 4, p. 425-444, 1990.

MAURO, J.; ZHU, J.; PRAMANICK, I. The system recovery benchmark In: IEEE PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING (PRDC), 10., 2004, Tokyo, Japão. **Proceedings…** Washington: IEEE Computer Society, 2004, p. 271-280.

MCLEAN, T.; FUJIMOTO, R. Repeatability in real-time distributed simulation executions. In: WORKSHOP ON PARALLEL AND DISTRIBUTED SIMULATION, 14., 2000, Bologna, Itália. **Proceedings…** Washington: IEEE Computer Society, 2000. p. 23-32.

MELHART, B.; WHITE, S. Issues in defining, analyzing, refining, and specifying system dependability requirements. In: ENGINEERING OF COMPUTER BASED SYSTEMS (ECBS), 17., 2000. **Proceedings...** Washington: IEEE Computer Society, 2000. p. 334-340.

MEYER, J. F. Defining and evaluating resilience: a performability perspective. In: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON PERFORMABILITY MODELING OF COMPUTER AND COMMUNICATION SYSTEMS (PMCCS), 2009, Eger, Hungria. **Proceedings...** PMCCS, 2009.

MIAO, Y.; CHEN, C.; SUN, Z. A satellite system distributed simulation design and synchronous control. In: INTERNATIONAL CONFERENCE ON MECHATRONICS AND AUTOMATION (ICMA), 2009, Changchun, China. **Proceedings...** IEEE, 2009. p. 3889-3893.

MICSKEI, Z.; MAJZIK, I.; TAM, F. Comparing robustness of AIS-based middleware implementations. In: **Service Availability**. Springer Berlin Heidelberg, 2007. p. 20-30.

MÖLLER, B.; OLSSON, L. Practical experiences from HLA 1.3 to HLA IEEE 1516 interoperability. In: SPRING SIMULATION INTEROPERABILITY WORKSHOP, 2004. **Proceedings...** SISO, 2004.

MÖLLER, B.; KARLSSON, M.; LÖFSTRAND, B. Developing fault tolerant federations using HLA evolved, In: SPRING SIMULATION INTEROPERABILITY WORKSHOP, 2005, San Diego, Califórnia, EUA. **Proceedings…** SISO, 2005**.**

MÖLLER, B.; MORSE, K.; LIGHTNER, M.; LITTLE, R.; LUTZ, R. L. HLA evolved - a summary of major technical improvements. In: SPRING SIMULATION INTEROPERABILITY WORKSHOP, 2008, Providence, Rhode Island, EUA. **Proceedings…** SISO, 2008**.**

MORAES, R.; BARBOSA, R.; DURÃES, J. A.; MENDES, N.; MARTINS, E.; MADEIRA, H. S. Injection of faults at component interfaces and inside the component code: are they equivalent? In: DEPENDABLE COMPUTING CONFERENCE (EDCC), 6., 2006, Coimbra, Portugal. **Proceedings …** IEEE, 2006. p. 53-64.

MOSTERT, D. N. J.; VON SOLMS, S. H. A technique to include computer security, safety, and resilience requirements as part of the requirements specification. **Journal of Systems and Software**, v. 31, n. 1, p. 45-53, 1995.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). **NPR7120.7**: NASA Information technology and institutional infrastructure program and project management requirements. Washington, 2008a. Disponível em: <http://www.hq.nasa.gov/office/codeq/doctree/71207.htm>. Acesso em: 15 jun. 2011.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). **NASA-STD-7009**: Standard for model and simulation. Washington, 2008b. 58p.

NEMETH, D. Benchmarking the RTI for use in a simulated radio environment. In: SPRING SIMULATION INTEROPERABILITY WORKSHOP, 1999, Orlando, Flórida, EUA. **Proceedings…** SISO, 1999.

NOULARD, E.; ROUSSELOT, J.; SIRON, P. CERTI, an open source RTI, why and how. In: EUROPEAN SIMULATION INTEROPERABILITY WORKSHOP, 2009, Istambul, Turquia. **Proceedings…** SISO, 2009.

OBASANJO, D. **Designing extensible, versionable XML formats**, 2004, disponível em: <http://msdn.microsoft.com/en-us/library/ms950793.aspx>. Acesso em: 20 jul. 2013.

ORLANDO, V.; ROZENFELD, P.; MIGUEZ, R. R. B.; FONSECA, I. M. Brazilian data collecting satellite simulator. In: INTERNATIONAL SYMPOSIUM ON SPACE TECHNOLOGY AND SCIENCE (IST), 1992, Kagoshima, Japão. **Proceedings…** 1992.

PAN, J.; KOOPMAN, P. ; HUANG, Y.; GRUBER, R.; JIANG, M. L. Robustness testing and hardening of CORBA ORB implementations. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2001, Göteborg, Suécia. **Proceedings…** Washington: IEEE Computer Society, 2001. p. 141-150.

PIMM, S. L. The complexity and stability of ecosystems. **Nature**, v. 307, n. 5949, p. 321-326, 1984.

REIS, A. M. **Simulação distribuída em tempo-real de um sistema de controle de atitude e órbita para a plataforma multi-missão utilizando a arquitetura HLA.** 2009. 196 p. (INPE-15782-TDI/1525). Dissertação (Mestrado em Mecânica Espacial e Controle) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2009. Disponível em: <http://urlib.net/8JMKD3MGP8W/34RR5T5>. Acesso em: 11 nov. 2014.

RESILIÊNCIA. In: **Dicionário Priberam da Língua Portuguesa**, 2008-2013. Disponível em: <http://www.priberam.pt/dlpo/resiliência>. Acesso em: 02 mar. 2013.

RESIST NoE. **Deliverable D13**: From resilience-building to resilience-scaling technologies: directions. 2007.

ROMANI, M. A. D. S.; LAHOZ, C. H. N.; YANO, E. T. Identifying dependability requirements for space software systems. **Journal of Aerospace Technology and Management,** v.2, p. 287-300, 2010.

ROSS, P. Comparison of High Level Architecture run-time infrastructure wire protocols–part one. In: ASIA PACIFIC SIMULATION TECHNOLOGY AND TRAINING CONFERENCE AND EXHIBITION (SIMTECT), 2012, Adelaide, Austrália. **Proceedings…** SimTecT, 2012.

RUIZ J.; GIL, P.; YUESTE, P.; DAVID, D. DeBERT: Dependability benchmark of automotive engine. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability Benchmarking for Computer Systems**. New Jersey: John Wiley & Sons, Inc., 2008. cap. 13, p. 255-283.

RUS, I.; KOMI-SIRVIO, S.; COSTA, P. **Software dependability properties:** a survey of definitions, measures and techniques. Maryland: Fraunhofer Center for Experimental Software Engineering, 2003. (Technical Report 03-110).

SCHAEFFER-FILHO, A.; SMITH, P.; MAUTHE, A. Policy-driven network simulation: a resilience case study. In: ACM SYMPOSIUM ON APPLIED COMPUTING (SAC), 26., 2011, Taichung, Taiwan. **Proceedings…** ACM, 2011. p. 492-497.

SEBASTIAO, N.; REGGESTAD, V.; SPADA, M.; WILLIAMS, A.; PECCHIOLI, M.; LINDMAN, N.; FRITZEN, P. A reference architecture for spacecraft simulators. In: INTERNATIONAL CONFERENCE AND SPACE OPERATION(SPACEOPS), 2008, Heidelberg, Alemanha, **Proceedings…** SPACEOPS, 2008.

SHELTON, C. P. ; KOOPMAN, P. ; DEVALE, K. Robustness testing of the Microsoft Win32 API. In: CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2000, Nova Iorque, EUA. **Proceedings…** Washington: IEEE Computer Society, 2000. p. 261-270.

STERBENZ, J. P. ; HUTCHISON, D.; ÇETINKAYA, E. K.; JABBAR, A.; ROHRER, J. P. ; SCHÖLLER, M.; SMITH, P. Resilience and survivability in communication networks: strategies, principles, and survey of disciplines. **Computer Networks**, v. 54, n. 8, p. 1245-1265, 2010.

TAMVAKIS, P. ; XENIDIS, Y. Comparative evaluation of resilience quantification methods for infrastructure systems. **Procedia - Social and Behavioral Sciences**, v. 74, n. 0, p. 339-348, 2013.

TAN, K. M. C.; MAXION, R. A. Toward evaluating the dependability of anomaly detectors. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability Benchmarking for Computer Systems**. New Jersey: John Wiley & Sons, Inc., 2008. cap. 8, p. 141-161.

TOMINAGA, J.; SILVA, J. D.; FERREIRA, M. A proposal for implementing automation in satellite control planning. In INTERNATIONAL CONFERENCE AND SPACE OPERATION (SPACEOPS), 2008, Heidelberg, Alemanha, **Proceedings…** SPACEOPS, 2008.

TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC). **TPC-C**, TPC BENCHMARK™ C - Standard specification. 2010a. Disponível em: <http://www.tpc.org/tpch/>. Acesso em: 16 jul. 2013.

TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC). **TPC-E**, TPC BENCHMARK™ E - Standard specification. 2010b. Disponível em: <http://www.tpc.org/tpch/>. Acesso em: 16 jul. 2013.

TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC). **TPC-DS**, TPC BENCHMARK™ DS - Standard specification. 2012. Disponível em: <http://www.tpc.org/tpch/>. Acesso em: 16 jul. 2013.

TRANSACTION PROCESSING PERFORMANCE COUNCIL (TPC). **TPC-H**, TPC BENCHMARK™ H - Standard specification. 2013. Disponível em: <http://www.tpc.org/tpch/>. Acesso em: 16 jul. 2013.

TRIVEDI, K S.; KIM, D. S.; GHOSH, R. Resilience in computer systems and networks. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 2009, San Jose, Califórnia, EUA. **Proceedings…** ACM, 2009. p. 74-77.

TRIVELATO, G. C. **Agendamento e simulação paralela/distribuída de processos em tempo real/crítico com arquiteturas modernas tipo HLA**. 2004. 194 p. Tese (Doutorado em Mecânica Espacial e Controle) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012. Disponível em: <http://urlib.net/sid.inpe.br/jeferson/2004/06.22.15.36>. Acesso em: 11 nov. 2013.

TSAI, W. T.; WEI, X.; CHEN, Y.; PAUL, R. A robust testing framework for verifying web services by completeness and consistency analysis. In: INTERNATIONAL SYMPOSIUM ON SERVICE-ORIENTED SYSTEM ENGINEERING (SOSE), 2005, Beijing, China. **Proceedings…** IEEE, 2005. p. 151-158.

UNITED STATES DEPARTAMENT OF DEFENSE (DoD). **5000.59-M**: Modeling and simulation (M&S) glossary, Alexandria, 2011. 161p.

VIEIRA, M; MADEIRA, H. A dependability benchmark for OLTP application environments. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB), 29., 2003, Berlim, Alemanha. **Proceedings…** VLDB Endowment, 2003. v.29, p. 742-753.

VIEIRA, M. P. A. **Testes padronizados de confiabilidade para sistemas transaccionais**. 332 p. Tese (Doutorado) - Universidade de Coimbra, Coimbra, Portugal, 2005.

VIEIRA, M; LARANJEIRO, N.; MADEIRA, H. Benchmarking the robustness of web services. In: PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING (PRDC), 13., 2007, Melbourne, Austrália. **Proceedings…** Washington: IEEE Computer Society, 2007. p. 322-329.

VIEIRA, M.; DURÃES, J. A.; MADEIRA, H. S. Dependability benchmark for OLTP systems. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability benchmarking for computer systems**. New Jersey: John Wiley & Sons, Inc., 2008. cap. 5, p. 63-90.

VIEIRA, M.; MADEIRA, H. S.; SACHS, K. Resilience benchmarking. In: WOLTER, K.; AVRITZER, A.; VIEIRA, M.; VAN MOORSEL, A. (eds.). **Resilience Assessment and Evaluation of Computer Systems**. Heidelberg: Springer-Verlag Berlin Heidelberg, 2012. cap. 14, p. 283-301.

WAINER, G.; GLINSKY, E.; GUTIERREZ-ALCARAZ, M. Studying performance of DEVS modeling and simulation environments using the DEVStone benchmark. **SIMULATION**, v. 87, p. 555-580, 2011.

WATROUS, B.; GRANOWETTER, L.; WOOD, D. HLA federation performance: what really matters. In: FALL SIMULATION INTEROPERABILITY WORKSHOP, 2006, Orlando, Flórida, EUA. **Proceedings…** SISO, 2006.

WEINSTOCK, C. B.; GOODENOUGH, J. B.; HUDAK, J. J. **Dependability cases**. Pittsburgh : Carnegie-Mellon University, Software Engineering Institute, 2004. 31p. (CMU/SEI-2004-TN-016).

WERKMAN, A.; PICARD, C.; MOSENKIS, R. ATV and ISS flight controller training. In INTERNATIONAL CONFERENCE AND SPACE OPERATION (SPACEOPS), 2012, Estocolmo, Suécia, **Proceedings…** SPACEOPS, 2012.

WIERMAN, S. J.; NARASIMHAN, P. Vajra: evaluating byzantine-fault-tolerant distributed systems. In: KANOUN, K.; SPAINHOWER, L. (eds.). **Dependability Benchmarking for Computer Systems**. New Jersey: John Wiley & Sons, Inc., 2008. cap. 9, p. 163-184, 2008.

WOLTER, K.; AVRITZER, A.; VIEIRA, M.; VAN MOORSEL, A. **Resilience Assessment and Evaluation of Computer Systems**. Springer-Verlag Berlin Heidelberg, 2012.

WORLD WIDE WEB CONSORTIUM (WC3)**. XML Schema Part 1:** structures. second edition, 2004. Disponível em: <http://www.w3.org/TR/xmlschema-1**/**>**.** Acesso em: 10 jul. 2013.

YIQUN, Y.; ZONG, P. ; JUN, S. Distributed interactive simulation for iridium communication system based on HLA and OPNET. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON COMPUTER AND ELECTRICAL ENGINEERING, 3., 2010, Chengdu, China. **Proceedings…** ICCEE, 2010.

ZHONGSHI, P. ; YUZHU, Z.; LI, D. Space exploration mission concurrent design and simulation research. In: INTERNATIONAL CONFERENCE ON NETWORKED COMPUTING AND ADVANCED INFORMATION MANAGEMENT (NCM), 7., 2011, Gyeongju, Coreia. **Proceedings…** NCM, 2011. p. 247-251.

GLOSSÁRIO

aceleração (tempo) - o tempo de simulação avança em relação ao tempo real usando um fator de aceleração constante (rápido ou lento) (ECSS, 2011).

acurácia - diferença entre um parâmetro ou variável (ou um conjunto de parâmetros ou variáveis) dentro de um modelo, simulação ou experimento e o valor verdadeiro ou o valor verdadeiro assumido (NASA, 2008b).

agendabilidade - nível no qual comportamentos e serviços podem ser agendados ocorrendo nos horários previstos (FIRESMITH, 2003).

arcabouço (*framework*) – um conjunto de classes colaborativas que compõem um projeto reutilizável para uma classe específica de problemas (GAMMA, 1994)

atributo - uma propriedade física ou abstrata mensurável de uma entidade (ISO/IEC, 2001).

avaria (*failure*) - a incapacidade de um produto para executar uma função requerida ou a sua incapacidade de realizá-la dentro de limites previamente especificados (ISO/IEC, 2001).

componente – unidade de funcionalidade que pode ser implementada e que tem uma interface bem definida com o ambiente. Em simuladores, componentes podem ser modelos e serviços. (ECSS, 2011).

comunicação confiável (*reliable*) - propriedade que garante a entrega da mensagem ao destinatário (DOD, 2011).

comunicação não-confiável (*best effort*) - comunicação em que os dados transmitidos não utilizam protocolo com reconhecimento. Esses dados normalmente chegam completos e sem erros ao destino, no entanto, se ocorrer um erro, nada é feito para corrigi-lo, por exemplo, não há retransmissão (DOD, 2011).

confiabilidade - operação de um sistema sem falha durante um dado período de tempo (FIRESMITH, 2003).

confidencialidade - garantia de que uma informação não estará disponível ou não será divulgada para indivíduos, entidades ou processos não autorizados (AVIZIENIS et al., 2004).

correção - capacidade que um software tem de produzir os resultados ou efeitos corretos ou acordados com precisão (FIRESMITH, 2003).

currency - indica o quanto os dados se mantêm correntes (atualizados) (FIRESMITH, 2003).

desempenho - extensão em que um sistema maximiza o processamento das informações com os recursos (i.e., os processadores, dispositivos de armazenamento, largura de banda de comunicação, etc.) a serem utilizados para processar a carga de trabalho do sistema (o volume e a distribuição de serviços solicitados ao longo do tempo) (BOEHM et al., 2004).

disponibilidade - grau de disponibilidade do sistema, ou seja, a prontidão para oferta do serviço correto (AVIZIENIS et al., 2004).

emular - representar um sistema por meio de um modelo que aceita as mesmas entradas e produz os mesmos resultados que o sistema representado (IEEE, 1989).

estabilidade – em simulação, é a estabilidade das saídas de um modelo ao longo do tempo (ECSS, 2010).

erro – estado do sistema ativado por uma falha (*fault*) que pode levar à subsequente avaria (*failure*) do serviço (AVIZIENIS et al., 2004).

falha (*fault*) - uma instrução, processo ou definição de dados errada em um programa de computador (ISO/IEC, 2001).

fidelidade - (1) o grau em que a representação dentro de uma simulação é semelhante a um objeto do mundo real; (2) acurácia da representação de um elemento quando comparada ao mundo real (DOD, 2011).

hardware na malha (*hardware-in-the-loop*) – simulação que tem interface com um hardware externo – equipamento real ou de teste (ECSS, 2010).

infraestrutura – (1) uma base ou fundamento; (2) as instalações básicas, equipamentos e instalações necessárias para o funcionamento de um sistema (DOD, 2011).

infraestrutura de simulação - infraestrutura de software que é usada para executar modelos simulados. Geralmente tem um agendador, fornece o tempo de simulação, e possibilita, via *scripts* ou interface GUI, o controle dos modelos, a visualização de suas variáveis públicas e estados. Pode prover outros serviços, tais como salvamento e recuperação de estados, registro de eventos de modelo, etc. (ECSS, 2010).

integridade - ausência de alterações inadequadas no sistema ou nos dados (AVIZIENIS et al., 2004).

interoperabilidade - a capacidade que um sistema tem de se comunicar de forma transparente com outro sistema (semelhante ou não) (ISO/IEC, 2001).

jitter - caracterizado pelas variações de um sinal digital no tempo (por exemplo, um pulso de sincronização do relógio) (ECSS, 2010)

latência - tempo de atraso entre o momento em que algo é iniciado e o momento em que um dos seus efeitos começa, por exemplo, o tempo entre o início do comando que define uma interrupção e a resposta do computador de bordo (ECSS, 2010).

linha de base (*baseline*) - conjunto de informações que descreve exaustivamente uma situação em um dado instante de tempo ou ao longo de um determinado intervalo de tempo. Uma linha de base é geralmente usada como referência para comparação e análise das subsequentes evoluções da informação (ECSS, 2012).

medição - a utilização de uma métrica para atribuir um valor (que pode ser um número ou categoria), a partir de uma escala, a um atributo de uma entidade (ISO/IEC, 2001).

medida - o valor ou categoria atribuída a um atributo de uma entidade por meio de uma medição (ISO/IEC, 2001).

métrica – definição do método de medição e escala de medição (ISO/IEC, 2001).

middleware - software que conecta ou integra módulos de outros softwares ou componentes, normalmente fornecendo serviços de comunicação ou funções de interação que podem ser invocados pelos módulos interligados (DOD, 2011).

modelos simulados - podem ser modelos de dados, modelos matemáticos, modelos geométricos de um sistema ou modelos de comportamento. Por exemplo, os algoritmos que representam o comportamento de um componente do ambiente expressos em uma linguagem de programação de alto nível. Dependendo do contexto, os modelos podem ser classificados de acordo com sua fidelidade, seu domínio ou sua técnica de modelagem (ECSS, 2011).

metodologia - princípios, práticas e procedimentos aplicados a um ramo específico do conhecimento (DOD, 2011).

rendimento (*throughput)* - o número de vezes que um serviço pode ser fornecido dentro de uma unidade de tempo especificada (FIRESMITH, 2003).

reúso - (1) prática de usar de novo, no todo ou em parte, ferramentas, dados ou serviços existentes; (2) em simulação, a reutilização engloba as políticas, práticas e tecnologias de suporte para promover a efetiva reutilização dos recursos da modelagem e simulação, por exemplo, modelos conceituais, arquitetura de software, projetos, componentes de software, modelos simulados, infraestrutura de simulação, etc. (DOD, 2011).

robustez - grau em que um sistema ou um componente pode funcionar corretamente na presença de entradas inválidas, falhas ou estresse das condições ambientais (IEEE, 1990).

segmento - conjunto de elementos ou uma combinação de sistemas que cumpre um subconjunto principal ou autocontidos dos objetivos da missão espacial. Exemplos disso são o segmento espacial, segmento de solo, segmento de lançamento e apoio (ECSS, 2012).

segmento espacial - parte de um sistema espacial situada no espaço e que visa cumprir os objetivos da missão (ECSS, 2012).

segmento solo - parte de um sistema espacial situada em solo que monitora e controla os elementos do segmento espacial (ECSS, 2012).

segmento de lançamento - parte do sistema espacial que é usada para transportar os elementos do segmento espacial para o espaço (ECSS, 2012).

segurança (*safety*) - habilidade do sistema em entregar um serviço sob dadas condições sem efeitos catastróficos (MELHART; WHITE, 2000).

segurança (*security*) - capacidade do sistema em prevenir, reduzir e reagir a um ataque malicioso (FIRESMITH, 2003).

simulação - a imitação das características de um sistema, entidade, fenômeno ou processo usando um modelo computacional (NASA, 2008b).

simulação de tempo real – simulação em que o tempo simulado progride na mesma velocidade que o tempo do relógio de parede (mas não necessariamente coincide com o tempo do relógio) (ECSS, 2010).

simulação orientada a eventos: simulação em que a atenção está focada na ocorrência de eventos e os tempos em que esses eventos ocorrem (IEEE, 1989).

simulação *as-fast-as-possible* - o tempo de simulação avança tão rápido quanto possível, e não está relacionado ao tempo real (ECSS, 2011).

simulação de tempo acelerado – a duração das atividades dentro de uma simulação na qual o tempo simulado avança mais rápido do que o tempo real (IEEE, 1989).

simulação de tempo retardado – a duração das atividades dentro de uma simulação na qual o tempo simulado avança mais lentamente do que o tempo real (IEEE, 1989).

simuladores fim-a-fim (*end-to-end*) – usados para simular o produto final de uma missão (ECSS, 2010).

sistema espacial – contém ou o segmento espaço, ou o segmento solo, ou o segmento de lançamento. Geralmente um sistema espacial é composto pelos três segmentos (ECSS, 2010).

software na malha (*software-in-the-loop*) - simulação e simuladores que empregam um ou mais elementos do software operacional no sistema de simulação (DOD, 2011).

sobrevivência *(survivability)* – habilidade de um dado sistema em continuar operando corretamente na presença de falhas acidentais e ataques maliciosos (ELLISON et al., 1997).

taxa de atualização - frequência com que o estado do modelo será atualizado.

tempo de resposta - tempo transcorrido entre a solicitação de um serviço por um usuário e a resposta do mesmo ou o tempo de acesso a um recurso (FIRESMITH, 2003).

tempo real restrito (*hard real-time*) - em simulação, garantias específicas são dadas sobre a duração dos períodos de atualização de um modelo. Normalmente, um modelo será atribuído um "*slot*" de tempo em que ele deve ser executado e, caso isso não ocorra, encerra-se a simulação. Usado principalmente quando há hardware na malha de simulação (ECSS, 2010).

tempo real flexível (*soft real-time*) - em simulação, tempo real no qual o tempo simulado pode sofrer desvios sem afetar os resultados da simulação, com a expectativa de que o tempo desviado seja recuperado mais tarde (ECSS, 2010).

tempo simulado (tempo virtual) - tempo conforme representado dentro da simulação, sem relação com o tempo de parede (IEEE, 1989)

tolerância a falhas - a capacidade de um sistema ou componente para continuar a operação normal apesar da presença de falhas de hardware ou software (IEEE, 1990).

validação - processo que demonstra que o produto é capaz de realizar a sua função no ambiente de operação pretendido (ECSS, 2012).

verificação - processo que demonstra, por meio do fornecimento de evidências objetivas, que o produto foi projetado e produzido de acordo com as suas especificações e desvios acordados e é livre de defeitos (ECSS, 2012).

APÊNDICE A - HIGH LEVEL ARCHITECTURE

Este anexo apresenta as regras HLA expressas pela norma 1516-2010 (IEEE, 2010a) e o fluxo de trabalho (*workflow*) de um federado HLA típico no que tange às chamadas de serviço da API HLA, bem como a representação de um passo de simulação no contexto deste trabalho.

A.1 Regras HLA

As Regras HLA estabelecem como se dá a interação entre federados e federações HLA-conforme e definem quais são as responsabilidades de cada grupo. São ao todo dez regras, cinco delas dizem respeito às federações e as outras cinco aos federados (IEEE, 2010a).

Cinco regras cobrem aspectos gerais das federações e estão descritas a seguir: (i) as federações devem ter um documento HLA FOM em conformidade com o padrão HLA OMT; (ii) em uma federação, a representação de uma instância de objeto associado à simulação deve estar no federado e não na RTI; (iii) durante a execução de uma federação, a troca de dados entre federados deverá ocorrer por meio da RTI; (iv) durante a execução federação, federados devem interagir com a RTI em conformidade com a especificação de interface HLA; e (v) durante a execução da federação, cada atributo publicado deve ser de propriedade de pelo menos um federado em um dado momento.

As demais regras dizem respeito às responsabilidades de cada federado e são: (i) cada federado deve ter um documento HLA SOM em conformidade com o padrão HLA OMT; (ii) cada federado deve ser capaz de atualizar e/ou receber dados de atributos e de enviar e/ou receber interações conforme especificado no seu SOM; (iii) cada federado deve ser capaz de transferir e/ou aceitar a propriedade de atributos dinamicamente durante a execução de uma federação, de acordo com a especificação do seu SOM; (iv) cada federado deve ser capaz de alterar as condições sob as quais atualiza os seus dados (por exemplo, limites), conforme especificado em seu SOM; e (v) cada

federado deve ser capaz de gerir o seu tempo local de forma que lhe permita coordenar os dados intercambiados com outros membros da federação.

A.2 Federado Típico HLA

O padrão HLA, por meio da especificação de Interface, define um conjunto completo de serviços que devem ser implementados por um RTI HLA e que podem ser utilizados pelos federados HLA-conforme. Esses serviços cobrem alguns aspectos genéricos de uma infraestrutura de simulação e podem ser utilizados de acordo com as necessidades de uma federação.

Federados HLA podem simular um satélite inteiro, um subsistema ou um equipamento do mesmo, dependendo da arquitetura proposta para a federação, mas devem seguir uma lógica comum mínima. Nesta seção é apresentado o fluxo de trabalho (*workflow*) que representa a lógica básica de um federado HLA e o uso dos serviços da API HLA. De maneira geral, um federado segue os seguintes passos:

1) Cria a federação (o primeiro federado que se liga à federação deve criá-la).
2) Associa-se à federação.
3) Define o esquema de sincronização de tempo (opcional).
4) Faz a publicação de *classes de objetos*, *atributos* e *classes de interação* às quais tem a capacidade de originar/atualizar e subscreve aos *atributos* e *classes de interação* de seu interesse.
5) Cria instâncias dos *objetos* publicados cujos dados o federado atualizará durante a execução da simulação.
6) Executa os passos de simulação até o final da mesma (o passo de simulação será detalhado posteriormente).
7) Destrói as instâncias de objetos criados no passo 5.
8) Desliga-se da federação.
9) Destrói a federação (o último federado destrói a federação).

A Figura A.1 resume o fluxo de trabalho genérico de um Federado HLA, apresentando os serviços da API de acordo com os passos enumerados acima.

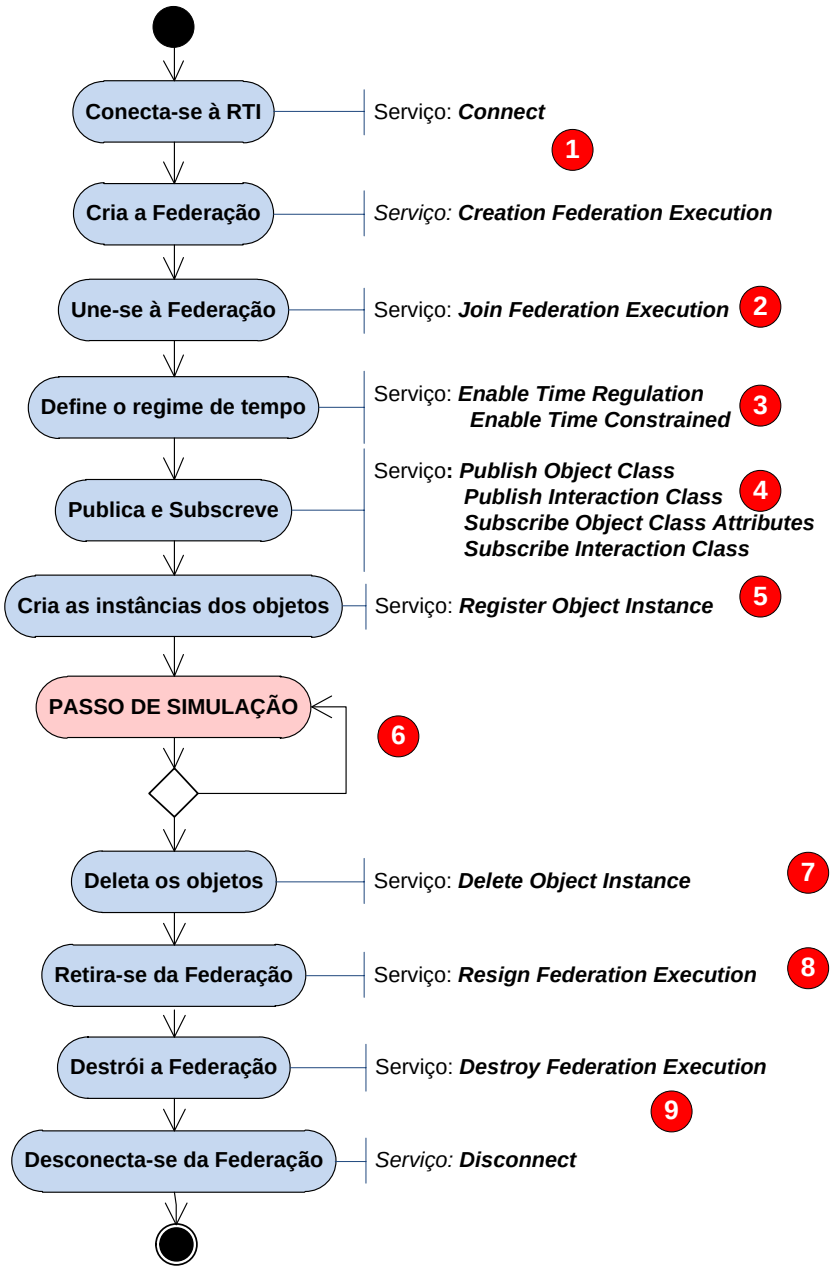


Figura A.1 - Fluxo de trabalho de um Federado Típico HLA (*template*).

Neste trabalho foi utilizada a abordagem de um passo de simulação na qual um federado HLA repete o mesmo padrão de execução periodicamente em um espaço de tempo Δt, conforme definido por McLean e Fujimoto (2000). A Figura A.2 descreve o passo de simulação típico adotado.

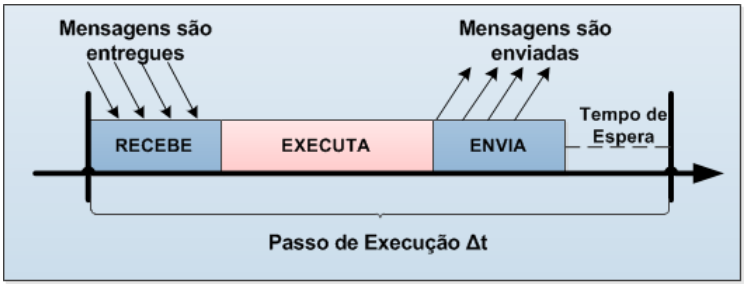


Figura A.2 - Passo de Simulação

Fonte: adaptada de McLean e Fujimoto (2000).

A cada passo de simulação estão associadas quatro fases: (i) fase de recepção de dados (*callbacks Receive Interaction* e/ou *Reflect Attribute Values*), na qual os dados de entrada necessários para o cálculo da simulação são recebidos dos demais modelos; (ii) fase de execução, na qual os cálculos pertinentes são executados; (iii) fase de transmissão de dados (*Update Attribute Values* e/ou *Send Interaction*), na qual são enviados os resultados a serem utilizados pelos demais modelos; e (iv) uma fase de espera até que se complete o tempo do passo de simulação.

A.2 Federado com Falhas de Interface (API HLA)

Os federados injetados na simulação para execução do benchmark de robustez (federados com falha) não seguem o mesmo fluxo de trabalho de federados típicos. Neste caso, o objetivo é: (1) conectar-se à federação; (2) associar-se à federação; (3) chamar um serviço da API usando um valor inválido; (4) deixar a federação; e (5) desconectar-se da federação.

A Figura A.3 apresenta o fluxo de trabalho usado para esses federados.

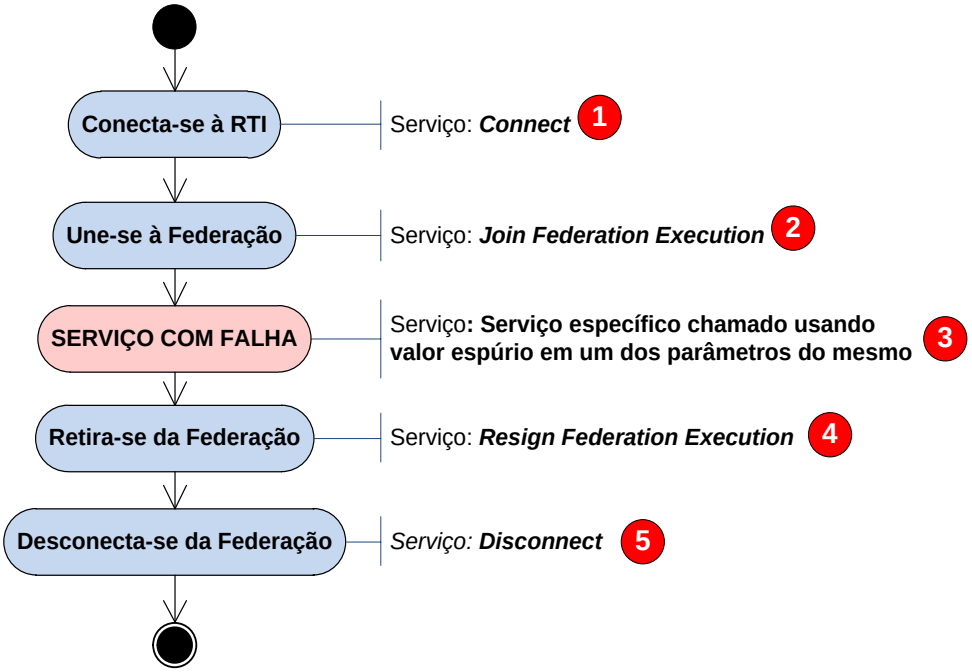


Figura A.3 - Fluxo de trabalho de um Federado com Falha de Interface (*template*).

APÊNDICE B - ELEMENTOS DO BENCHMARKING DE RESILIÊNCIA

Este apêndice apresenta os elementos que foram definidos pela abordagem de benchmarking para infraestruturas de simuladores de satélite baseadas em HLA e complementa o Capítulo 5.

B.1 Atributos Dependabilidade e Resiliência

Esta seção apresenta detalhes acerca das duas abordagens utilizadas na definição dos atributos de dependabilidade relevantes para o domínio de simuladores de satélite. Em uma primeira abordagem, foi realizada uma pesquisa com especialistas no domínio de simuladores de satélite e, em uma segunda abordagem, foram analisados os requisitos funcionais de uma infraestrutura de simulação relativamente à lista abrangente de atributos. O conjunto de atributos de dependabilidade descrito na Tabela 5.1 foi utilizado pelas duas abordagens e as seções a seguir apresentam detalhes das mesmas.

B.1.1 Pesquisa com Especialistas no Domínio de Simuladores

A pesquisa com especialistas do domínio de simuladores foi realizada apresentando a cada especialista um formulário que descrevia: (i) os atributos a serem avaliados (Tabela 5.1); (ii) as classes de especialistas (*stakeholders*); (iii) as classes de simuladores e os tipos de simuladores utilizados; e (iv) os níveis de exigência de cada atributo (nota). A Tabela B.1 apresenta os elementos da pesquisa realizada

Nessa pesquisa, cada especialista identificou a classe de *stakeholder* à qual pertencia e, para cada célula de uma matriz formada pelo tipo de simulador e atributo de dependabilidade, atribuiu uma nota da importância do atendimento daquele atributo para aquele tipo de simulador.

A pesquisa contou com a participação de onze especialistas sendo: dois adquirentes ou especificadores, dois especialistas em subsistemas, dois operadores do Centro de Controle e cinco desenvolvedores.

Tabela B.1 - Elementos da Pesquisa com *Stakeholders*.

ELEMENTOS	DESCRIÇÃO	ITENS
Especialistas	Classes de *stakeholders* relativamente a simuladores de satélites	Adquirente Engenheiros e especialistas em subsistemas de satélite Desenvolvedores de simuladores Operadores do centro de controle de satélite Usuários dos simuladores
Classes de Simuladores	As classes gerais de Simuladores de Satélites	Análise e apoio a projeto Integração e Teste Operacional
Tipos de Simuladores	Dentro de cada classe geral, os tipos de simuladores definidos pelo memorando técnico ECSS (2010)	*System Concept Simulator* (SCS) *Mission Performance Simulator* (MPS) *Functional Engineering Simulator* (FES) *Functional Validation Test Bench Simulator* (FVT) *Software Validation Facility* (SVF) *Spacecraft AIV Facility* (SVV) *Ground Segment Test Simulator* (GS) *Training Simulator* (GS) *Operation and Maintenance Simulator* (GS)
Avaliação	Nível de exigência de cada atributo de dependabilidade relativamente a cada tipo de simulador ou classe geral.	1- Muito Baixo 2 - Baixo 3 - Alto 4 - Muito Alto

A Figura B.1 apresenta o resultado geral de atributos relevantes por classes de simuladores.

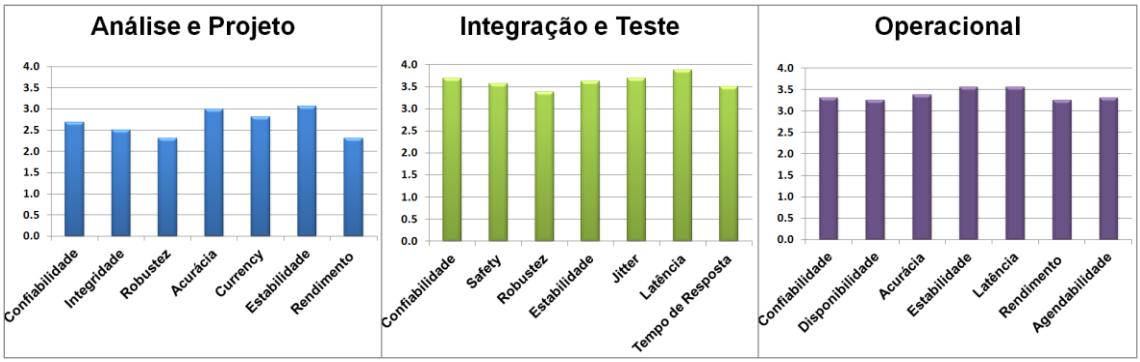


Figura B.1 - Atributos Relevantes - Classe de Simuladores.

B.1.2 Análise dos Atributos de Dependabilidade para a Infraestrutura de Simuladores

Para a análise dos atributos de dependabilidade que poderiam ter impactos sobre as infraestruturas de simuladores, foram elaboradas duas tabelas de requisitos. A primeira tabela é baseada nos requisitos de infraestrutura de

simuladores especificados pelo memorando técnico ECSS (2010). A segunda tabela foi elaborada por meio da análise do nível de cobertura do padrão HLA relativamente a cada requisito funcional de uma infraestrutura de simulação genérica e considerou os seguintes níveis de conformidade para cada requisito: (i) *conforme*, quando o padrão HLA atendia completamente o requisito sem que fosse necessário esforço de implementação na infraestrutura de simulação ou nos modelos; (ii) *parcialmente conforme*, quando o padrão dava suporte ao atendimento do requisito, mas era necessário que a infraestrutura de simulação ou os modelos provessem parte da implementação do requisito; *não-conforme*, quando o padrão HLA não atendia e não dava apoio ao atendimento do requisito ou quando o mesmo estava fora do escopo do padrão.

Em seguida, foram avaliados os requisitos da infraestrutura de simulação, bem como os requisitos atendidos pelo padrão HLA, relativamente ao conjunto de atributos de dependabilidade apresentado na Tabela 5.1. Utilizou-se para essa avaliação a mesma escala utilizada para avaliar as classes de simuladores: 1 (muito baixo), 2 (baixo), 3 (alto) e 4 (muito alto). Dois especialistas participaram dessa avaliação.

A Figura B.2 apresenta o resultado da avaliação dos atributos relativamente à infraestrutura.

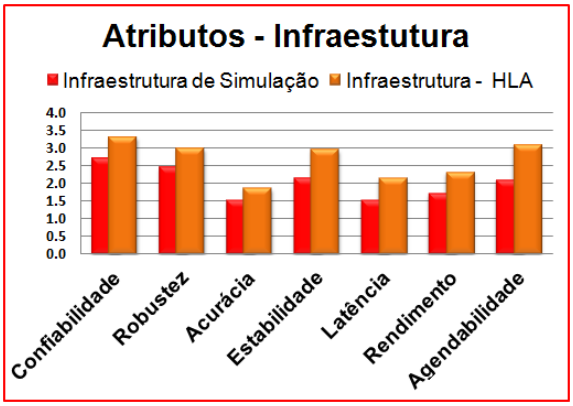


Figura B.2 - Atributos Relevantes - Infraestrutura de Simulação.

B.2 Métricas

Esta seção do apêndice apresenta o conjunto de métricas definidas para os atributos de dependabilidade e resiliência considerados mais relevantes no contexto de simuladores de satélite.

As Tabelas B.2 a B.9 foram elaboradas utilizando uma versão simplificada da metodologia de GQM com o objetivo de sistematizar o processo de análise dos atributos relativamente às suas métricas. Como estava fora do escopo deste trabalho validar novas métricas, nós descrevemos principalmente métricas encontradas na literatura ou amplamente utilizadas. O conjunto apresentado não exaure as possibilidades de métricas para os atributos considerados mais relevantes para o domínio de infraestruturas HLA, mas define um conjunto que pode ser utilizado em instanciações de benchmarks concretos.

Tabela B.2 - Métricas - Resiliência

Atributo: Reiliência	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Qual a perda de serviço observada nas medidas dos atributos de dependabilidade quando em face de mudanças?		
M1.1	Índice de resiliência - perda de serviço (*RLIndex*) (ALMEIDA et al., 2013)	
	Objetivo	Comparar a resiliência das infraestruturas de simulação relativamente a uma dada métrica de dependabilidade.
	Descrição	Índice de resiliência que quantifica a oferta de serviços da infraestrutura HLA em face de diferentes perturbações. O índice se aplica ao conjunto de métricas de dependabilidade e é calculado usando a média da relação entre a medida obtida na execução de referência e a medida obtida quando em presença de mudanças.
	Fórmula	$$RLIndex = \frac{1}{T}\sum_{i=1}^{T}\left(\frac{EC_i}{ER}\right)$$ T: Número de instâncias das mudanças aplicadas (definido pelo número de valores dos parâmetros das mudanças) *ER*: Medida de execução de referência EC_i: Medida da execução da i-ésima mudança Obs.: para casos nos quais o objetivo da métrica dependabilidade é a diminuição em seus valores, a seguinte fórmula deve ser usada: $$RLIndex = \frac{1}{T}\sum_{i=1}^{T}\left(\frac{EC_i}{ER}\right)^{-1}$$
	Interpretação	$\mathrm{RLIndex} \leq 1.0 \leq \mathrm{RLIndex}$ (quanto mais próximo de 1.0 melhor)

(Continua)

Tabela B.2 – Continuação.

M1.2	Índice de resiliência - perda de serviço (pior caso) (*RLIndexMin*) (adaptado de ALMEIDA et al., 2013)	
	Objetivo	Comparar a resiliência das infraestruturas de simulação relativamente a uma dada métrica de dependabilidade.
	Descrição	Índice de resiliência que quantifica a oferta de serviços da infraestrutura HLA em face de diferentes perturbações. O índice se aplica ao conjunto de métricas de dependabilidade e é calculado usando o pior caso para a relação entre a medida obtida na execução de referência e a medida obtida quando em presença de mudanças.
	Fórmula	$$RLIndexMin = MIN(\frac{EC}{ER}, T)$$ T: Número de instâncias das mudanças aplicadas (definido pelo número de valores dos parâmetros das mudanças) *ER*: Medida de execução de referência *EC:* Medida de execução da mudança MIN: Valor mínimo obtido nas medições (ER/EC) das execuções com mudanças. Obs.: para casos nos quais o objetivo da métrica dependabilidade é a diminuição em seus valores, a seguinte fórmula deve ser usada: $$RRLIndexMin = MIN(\left(\frac{EC}{ER}\right)^{-1}, T)$$
	Interpretação	$\text{RLIndexMin} \leq 1.0 \leq \text{RLIndexMin}$ (quanto mais próximo a 1.0 melhor)
Q2 Qual o impacto das mudanças nas diferentes métricas dos atributos de dependabilidade (Tabelas B.3 a B.9)?		
M2.1	Índice de resiliência - estabilidade (*RSindex*)	
	Objetivo	Comparar a resiliência das infraestruturas de simulação relativamente a uma dada métrica de dependabilidade.
	Descrição	Índice de resiliência que avalia a estabilidade da infraestrutura em face de diferentes *mudanças*. O índice se aplica ao conjunto de métricas de dependabilidade e é calculado com base no coeficiente de variabilidade para todo o experimento. Desta forma, considera-se a medida de referência como valor esperado e calcula-se o desvio padrão usando as medidas das execuções com *mudança*.
	Fórmula	$$RSindex = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(EC_i - ER)^2}$$ T: Número de instâncias da mudança aplicada (definido pelo número de valores dos parâmetros das mudanças) ER: Medida de execução de referência EC_i: Medida da execução da i-ésima mudança
	Interpretação	$0 <$ RSIndex (quanto menor, melhor)

(Continua)

Tabela B.2 – Conclusão.

M2.2	Índice de resiliência - estabilidade relativa (*RRindex*)	
	Objetivo	Comparar a resiliência das infraestruturas de simulação relativamente a uma dada métrica de dependabilidade.
	Descrição	Índice de resiliência que avalia a estabilidade relativa da infraestrutura em face de diferentes *mudanças*. O índice se aplica ao conjunto de métricas de dependabilidade e é calculado com base no coeficiente de variabilidade para todo o experimento. Desta forma, considera-se a medida de referência como valor esperado e calcula-se o desvio padrão relativo usando as medidas das execuções com *mudança*.
	Fórmula	$RRindex = \frac{\sqrt{\frac{1}{T}\sum_{i=1}^{T}(EC_i - ER)^2}}{\overline{ER}}$ T: Número de instâncias da mudança aplicada (definido pelo número de valores dos parâmetros das mudanças) ER: Medida de execução de referência EC_i: Medida da execução da i-ésima mudança
	Interpretação	0< $RRIndex$ (quanto menor, melhor)

Tabela B.3 - Métricas - Latência

Atributo: Latência	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Qual o tempo entre a atualização de atributos por um federado (emissor) e o reflexo dessa operação em outro federado (receptor)?		
M1.1.	Média de latência direta de atualização de atributos (*LUpdAvg*) (DRAKE et al., 2003)	
	Objetivo	Comparar as infraestruturas de simulação relativamente à latência no intercâmbio de dados entre modelos (federados).
	Descrição	Média do tempo de latência direta, ou seja, diferença entre o tempo de atualização de atributos no federado emissor e o tempo de reflexo da operação no federado receptor.
	Fórmula	$LUpdAvg = \frac{1}{P}\sum_{i=1}^{P}(tR - tU)$ tR: tempo do recebimento dos dados pelo receptor tU: tempo na atualização do dado pelo federado (emissor) P: número total de atualizações de atributos
	Interpretação	0 < LUpdAvg (quanto menor, melhor)
	Unidade	ms (milissegundos)
M1.2.	Média de latência direta de atualização de atributos por atributo (*LUpdAvgA*) - adaptada de Drake et al.(2003)	
	Objetivo	Comparar as infraestruturas de simulação relativamente à latência na troca de dados entre modelos independentemente do número de atributos.
	Descrição	Média do tempo de latência direta, ou seja, diferença entre o tempo de atualização de atributos no emissor e o tempo de reflexo da operação no federado por atributo atualizado.
	Fórmula	$LUpdAvgA = \frac{1}{P}\sum_{i=1}^{P}\frac{(tR-tU)}{N}$ tR: tempo do recebimento dos dados pelo receptor tU: tempo na atualização do dado pelo federado (emissor) N: número de atributos por atualização P: número total de atualizações de atributos
	Interpretação	0< LUpdAvgA (quanto menor, melhor)
	Unidade	ms (milissegundos)

(Continua)

Tabela B.3 – Continuação.

M1.3	Média de latência indireta (*round trip*) de atualização de atributos (*LUpdAvgRT*) (KNIGHT, 2002c)	
	Objetivo	Comparar as infraestruturas de simulação relativamente a latência na troca de dados entre modelos (federados).
	Descrição	Média do tempo de latência indireta (*round trip*), ou seja, diferença entre o tempo de atualização de atributos no emissor e o tempo de recebimento do retorno da operação no próprio emissor.
	Fórmula	$LUpdAvgRT = \frac{1}{P} \sum_{i=1}^{P} \frac{(tER - tEU) - (tRU - tRR)}{2}$ tEU: tempo na atualização dos atributos pelo emissor tRR: tempo no recebimento dos atributos pelo receptor tRU: tempo na atualização dos atributos pelo receptor tER: tempo no recebimento dos atributos pelo emissor P: número total de atualizações de atributos
	Interpretação	0< $LUpdAvgRT$ (quanto menor, melhor)
	Unidade	ms (milissegundos)
M1.4	Média de latência indireta (*round trip*) de atualização de atributos por atributo (*LUpdAvgRTA*) - adaptada de Knight (2002c)	
	Objetivo	Comparar as infraestruturas de simulação relativamente a latência na troca de dados entre modelos (federados).
	Descrição	Média do tempo de latência indireta (*round trip*), ou seja, diferença entre o tempo de atualização de atributos no emissor e o tempo de recebimento do retorno da operação no próprio emissor.
	Fórmula	$LUpdAvgRTA = \frac{1}{P} \sum_{i=1}^{P} \frac{(tER - tEU) - (tRU - tRR)}{2 * \mathrm{N}}$ tEU: tempo na atualização dos atributos pelo emissor tRR: tempo no recebimento dos atributos pelo receptor tRU: tempo na atualização dos atributos pelo receptor tER: tempo no recebimento dos atributos pelo emissor P: número total de atualizações de atributos N: número de atributos por atualização
	Interpretação	0< LUpdAvgRTA (quanto menor, melhor)
	Unidade	ms (milissegundos)
M1.5	Variação da latência de atualização de atributos (*LUpdDev*) (DRAKE et al, 2003)	
	Objetivo	Comparar as infraestruturas de simulação em relação à estabilidade na latência de atualização de atributos.
	Descrição	Desvio padrão do tempo de latência para atualização de atributos
	Fórmula	$LUpdDev = \sqrt{\frac{1}{P-1} \sum_{i=1}^{P} (x_i - \bar{X})^2}$ P: Número total de atualizações de atributos $\bar{X}$: Média da latência para atualização dos atributos x_i: Latência da i-ésima publicação
	Interpretação	0< LUpdDev (quanto menor, melhor)

(Continua)

Tabela B.3 – Conclusão.

Q2 Qual o impacto da latência na execução de um serviço em outro federado?		
M2.1	Média da latência direta de serviço (LIntAvg) (DRAKE et al, 2003)	
	Objetivo	Comparar as infraestruturas de simulação relativamente à latência de chamadas de serviços.
	Descrição	Média de tempo de latência para execução de serviços em outro federado.
	Fórmula	$LIntAvg = \frac{1}{C} \sum_{i=1}^{C} (tR - tC)$ tR: tempo no recebimento da chamada ao serviço (receptor) tC: tempo na chamada ao serviço (emissor) C: Número total de chamadas a Serviços
	Interpretação	0< LIntAvg (quanto menor, melhor)
	Unidade	ms (milissegundos)
M2.2	Média da latência indireta (*round trip*) de serviço (LIntAvgRT) (KNIGHT, 2002c)	
	Objetivo	Comparar as infraestruturas de simulação relativamente à latência de chamadas de serviços.
	Descrição	Média de tempo de latência indireta (round trip) para execução de serviços em outro federado.
	Fórmula	$LIntAvgRT = \frac{1}{C} \sum_{i=1}^{C} \frac{(tER - tEC) - (tRC - tRR)}{2}$ tEC: tempo da chamada pelo emissor tRR: tempo do recebimento da chamada no receptor tRC: tempo da chamada no receptor tER: tempo do recebimento da chamada no emissor C: Número total de chamadas a Serviços
	Interpretação	0< LIntAvgRT (quanto menor, melhor)
	Unidade	ms (milissegundos)
M2.3	Variação da latência (LIntDev) (DRAKE et al, 2003)	
	Objetivo	Comparar as infraestruturas de simulação em relação à estabilidade na latência de chamadas a serviços.
	Descrição	Desvio padrão do tempo de latência para chamada de serviços.
	Fórmula	$LIntDev = \sqrt{\frac{1}{C-1} \sum_{i=1}^{C} (x_i - \bar{X})^2}$ C: Número total de chamadas a Serviços $\bar{X}$: Média da latência para execução dos serviços x_i: Latência da i-ésima chamada
	Interpretação	0< LIntDev (quanto menor, melhor)

Tabela B.4 - Métricas - Rendimento

Atributo: Rendimento	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Quantas tarefas podem ser realizadas em um dado período de tempo? (ISO/IEC, 2002)		
M1.1	Frequência da taxa de atualização de atributos (TFreq) (ISO/IEC, 2002)	
	Objetivo	Comparar o rendimento das infraestruturas relativamente a atualização de atributos.
	Descrição	Mede a frequência de atualizações (*updates*) ou recebimentos (*reflect*) de atributos.
	Fórmula	$TFreq = \mathrm{AR} / \mathrm{T}$ AR: Número de atualizações/recebimentos T: Tempo transcorrido
	Interpretação	0 < TFreq (quanto maior, melhor)
	Unidade	Hz
M1.2	Taxa de transferência (TTrans) (ISO/IEC, 2002)	
	Objetivo	Comparar o rendimento das infraestruturas relativamente a atualização de atributos.
	Descrição	Mede o volume de dados transferido entre federados.
	Fórmula	$TTrans = \frac{V * AR}{1024}$ V: Volume de dados (bytes) de uma atualização AR: Número de atualizações/recebimentos
	Interpretação	0< TTrans (quanto maior, melhor)
	Unidade	Mb

Tabela B.5 - Métricas - Acurácia

Atributo: Acurácia	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Os modelos recebem os valores de atributos publicados corretamente?		
M1.1	Taxa de atributos divergentes (*ADBase*) (DURÃES et al., 2008)	
	Objetivo	Comparar as infraestruturas em função da correteza na publicação de atributos.
	Descrição	Número de atributos com divergências do valor esperado.
	Fórmula	$AD_{base} = 100 - ((\mathrm{A} / \mathrm{P}) * 100)$ A: Número de erros encontrados (parâmetros com valores divergentes) P: Número de publicações multiplicado pelo número de parâmetros por publicação
	Interpretação	$\mathrm{AD_{per}}$ <= 100 (quanto mais próximo a 100 melhor)

Tabela B.6 - Métricas - Robustez

Atributo: Robustez	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Com que frequência a infraestrutura de simulação falha de forma generalizada (catastrófica) em presença de perturbações? (ISO/IEC, 2002)		
M1.1	Taxa de Falhas catastróficas (*RFTCas*) adaptada de (FERNSLER; KOOPMAN, 1999)	
	Objetivo	Comparar a robustez das infraestruturas de simulação relativamente a falhas catastróficas.
	Descrição	Taxa de falhas catastróficas em presença de perturbações. Falhas catastróficas são aquelas que terminam a simulação, parando a infraestrutura de simulação (processo *RTIExec*) ou o sistema como um todo (sistema operacional).
	Fórmula	$RFTCas = \frac{FCas}{P}$ FCas: número de falhas catastróficas P: número de perturbações injetadas
	Interpretação	0< RFTCas (quanto menor, melhor)
Q2 Com que frequência a infraestrutura de simulação falha causando o travamento da infraestrutura em presença de perturbações.		
M2.1	Taxa de Falhas *Restart* (*RFTRes*) adaptada de (FERNSLER; KOOPMAN, 1999)	
	Objetivo	Comparar a robustez das infraestruturas relativamente a falhas do tipo *restart*.
	Descrição	Taxa de falhas de *restart* em presença de perturbações. Falhas de *restart* exigem a reinicialização da RTI ou da federação depois de um travamento.
	Fórmula	$RTFRes = \frac{FRes}{P}$ FRes: número de falhas restart P: número de perturbações injetadas
	Interpretação	0< RTFRes (quanto menor, melhor)
Q3 Com que frequência a infraestrutura de simulação falha causando o abortamento dos federados em presença de perturbações?		
M3.1	Taxa de Falhas Aborto (RTFAbort) adaptada de (FERNSLER; KOOPMAN, 1999)	
	Objetivo	Comparar a robustez das infraestruturas de simulação relativamente a falhas do tipo abortamento.
	Descrição	Taxa de falhas de abortamento em presença de perturbações. Falhas de abortamento causam a parada abrupta dos federados sem que haja o acionamento de nenhum mecanismo de tratamento de exceção.
	Fórmula	$RTFAbort = \frac{FAbort}{P}$ FAbort: número de falhas abortamento P: número de perturbações injetadas
	Interpretação	0< RTFAbort (quanto menor, melhor)

(Continua)

Tabela B.6 – Conclusão.

Q4 Com que frequência a infraestrutura de simulação não detecta um erro?		
M4.1	Taxa de Falhas Silenciosas (*RFTSientl*) adaptada de (FERNSLER; KOOPMAN, 1999)	
	Objetivo	Comparar as infraestruturas de simulação em função dos mecanismos de tolerância a falhas.
	Descrição	Taxa de falhas de silenciosas em presença de perturbações. Falhas silenciosas são as falhas não detectadas.
	Fórmula	$RTFSilent = \frac{FSil}{P}$ FSil: número de falhas não detectadas P: número de perturbações injetadas
	Interpretação	0< RFTSilent (quanto menor, melhor)
Q5 Com que frequência a infraestrutura de simulação trata exceções específicas através de tratamento de falha genérico escondendo o motivo do erro?		
M5.1	Taxa de Falhas Escondidas (*RFTHinder*) adaptada de (FERNSLER; KOOPMAN, 1999)	
	Objetivo	Comparar as infraestruturas de simulação em função dos mecanismos de tolerância a falhas.
	Descrição	Taxa de falhas escondidas em presença de perturbações. Falhas escondidas são as falhas tratadas por exceções genéricas.
	Fórmula	$RFTHinder = \frac{FHinder}{P}$ FHinder: número de falhas tratadas por exceções genéricas P: número de perturbações injetadas
	Interpretação	0< RFTHinder (quanto menor, melhor)
Q6 Quantos padrões de tratamentos de falhas foram utilizados em presença de perturbações? (ISO/IEC, 2002)		
M6.1	Mecanismos de tolerância (ISO/IEC, 2002)	
	Objetivo	Comparar as infraestruturas de simulação relativamente ao tratamento correto das exceções.
	Descrição	Taxa de tratamentos de falhas executados em presença das perturbações.
	Fórmula	$RTxExc = \frac{TF}{Ne}$ TF: padrões executados Ne: Número de exceções previstas
	Interpretação	0 < RTxExc (quanto maior, melhor)

Tabela B.7 - Métricas - Confiabilidade

Atributo: Confiabilidade	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Quão frequentemente a infraestrutura falha (*failure*) em operação?		
M1.1	Tempo médio entre falhas (MTBF) (ISO/IEC, 2002)	
	Objetivo	Avaliar a confiabilidade da infraestrutura (referência e mudanças)
	Descrição	Computa o número de falhas ocorridas durante um período definido de operação do sistema e calcula o intervalo médio entre as falhas.
	Fórmula	$Ctx = \mathrm{T} / \mathrm{F}$ T = Tempo de Operação F: Número de falhas observadas
	Interpretação	0 < Ctx (quanto maior, melhor)

Tabela B.8 - Métricas - Estabilidade

Atributo: Estabilidade	**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Qual a estabilidade do tempo dos passos de simulação dos modelos (federados)?		
M1.1.	Estabilidade do passo de simulação do modelo (SFedStep)	
	Objetivo	Comparar a estabilidade das infraestruturas em relação ao tempo de execução de um passo de simulação do modelo incluindo chamadas aos serviços de *callback*.
	Descrição	Desvio padrão da duração (tempo de processamento) dos passos de simulação do modelo.
	Fórmula	$$SFedStep = \sqrt{\frac{1}{S-1}\sum_{i=1}^{S}(x_i - \bar{X})^2}$$ S: Número total de passos de simulação $\bar{X}$: Média do tempo de processamento de um passo de simulação no modelo x_i: Tempo de processamento do passo da i-ésima execução do modelo
	Interpretação	$0 \leq \mathrm{SFedStep}$ (quanto menor, melhor)
Q2 Qual a estabilidade dos serviços de atualização de atributos?		
M2.1.	Taxa de perda de mensagens (SUpdLoss)	
	Objetivo	Comparar a estabilidade do sistema no que tange à perda de mensagens durante a atualização de atributos.
	Descrição	Quantifica o número de atualizações perdidas.
	Fórmula	$$SUpdLoss = \frac{R}{A}$$ A: objetos atualizados R: objetos recebidos
	Interpretação	$\mathrm{SUpdLoss} \leq 1.0$ (quanto mais próximo de um, melhor)
Q3 Qual a estabilidade da execução dos serviços?		
M3.1.	Taxa de perda de mensagens (SIntLoss)	
	Objetivo	Comparar a estabilidade do sistema no que tange à perda de mensagens durante a execução de serviços.
	Descrição	Quantifica o número de chamadas a serviços perdidas.
	Fórmula	$$SLoss = \frac{I}{E}$$ I: interações realizadas E: interações executadas
	Interpretação	$\mathrm{SLoss} \leq 1.0$ (quanto mais próximo de um, melhor)

Tabela B.9 - Métricas - Agendabilidade

Atributo: Agendabilidade		**Objetivo**: Avaliar	**Ponto de Vista**: Federação
Q1 Qual a frequência de desvio do tempo no agendamento de tarefas da federação?			
M1.1	Frequência de atrasos (SCDesv)		
	Objetivo	Comparar os mecanismos de agendamento.	
	Descrição	Mede a frequência de desvios (atrasos ou adiantamentos) no agendamento de tarefas da federação.	
	Fórmula	$SCDesv = \mathrm{D} \ / \ \mathrm{Ag}$ D: Número de desvios Ag: Número de execuções agendadas	
	Interpretação	0< SCDesv (quanto menor, melhor)	
Q2 Qual a média de tempo de desvio nos agendamentos?			
M2.1	Desvio médio de agendamento - SCDesvMed		
	Objetivo	Comparar os mecanismos de agendamento.	
	Descrição	Média do tempo de desvio ocorrido durante os agendamentos.	
	Fórmula	$SCDesvMed = \frac{1}{A} \sum_{i=1}^{A} T$ T: tempo de desvio A: Número de execuções agendadas	
	Interpretação	0< $\mathrm{SCDesvMed}$ (quanto menor, melhor)	
	Unidade	ms (milissegundos)	
Q3 Qual a taxa de *deadlocks* no agendamento de tarefas dos federados em presença de perturbações?			
M3.1	Taxa de *deadlocks* (SCDead)		
	Objetivo	Comparar os mecanismos de agendamento.	
	Descrição	Número de ocorrências de *deadlocks* no agendamento de tarefas.	
	Fórmula	$SCDead = \mathrm{A} \ / \ \mathrm{P}$ A: Número de *deadlocks* ocorridos P: Número de Publicações	
	Interpretação	0< SCDead (quanto menor, melhor)	

B.3 Requisitos do Sistema Gerenciador de Benchmark

Conforme indicado na metodologia, o conjunto de requisitos do SGB é uma das saídas do passo de *Definição do Sistema Gerenciador de Benchmark*. A Tabela B.10 apresenta a lista de requisitos gerais da ferramenta de benchmark que está projetada no Capítulo 6.

Os requisitos aqui apresentados são requisitos genéricos do sistema de gerenciamento de benchmark no que tange a infraestrutura HLA, requisitos adicionais poderão ser criados a partir dos requisitos de instanciação das mudanças e/ou carga de trabalho de benchmarks concretos.

Tabela B.10 - Requisitos Funcionais - SGB

Requisitos Funcionais (Gerais) do SGB	
GE1	O SGB deve ser capaz de ler o arquivo de definição e instanciação do benchmark (RBDL).
GE2	O SGB deverá ser capaz de gerar os federados da carga de trabalho automaticamente a partir do arquivo de definição e instanciação do benchmark (RBDL).
GE3	O SGB deverá ser capaz de gerar automaticamente as cargas de mudança a partir do arquivo de definição e instanciação do benchmark. (RBDL).
GE3	O SGB deverá ser capaz de gerar automaticamente o federado de monitoramento e controle (M&C) e os federados injetores a partir do arquivo de definição e instanciação do benchmark (RBDL).
CO1	O SGB deve ser capaz de, no início do experimento, iniciar o processo *RTIExec* na máquina definida no Cenário Base (RBDL).
CO2	O SGB deve ser capaz de iniciar a execução da federação, incluindo o federado de monitoramento e controle (M&C) e os federados injetores de falha/mudança, nas máquinas indicadas nos cenários de instanciação (RBDL)
CO3	O SGB deve ser capaz de terminar a execução dos federados de uma federação ao final do experimento.
CO4	O SGB deve ser capaz de terminar o processo *RTIExec* no final do experimento.
CO5	Os resultados dos experimentos devem ser armazenados localmente no final de cada rodada de execução.
CO6	O SGB deve ser capaz de coletar os resultados no final de cada experimento para posterior análise e geração das medidas.
AN1	O SGB deve ser capaz de calcular as medidas de cada uma das métricas definidas para o experimento por meio do arquivo de definição e instanciação do benchmark (RBDL).
AN2	O SGB deve ser capaz de gerar o arquivo de resultados (RBDL).

B.4 Carga de Trabalho

Esta seção apresenta as cargas de trabalho à qual uma infraestrutura de simulação estaria exposta no domínio de simuladores de satélites. Como cada benchmark de resiliência concreto instanciado a partir da especificação pode ter objetivos diferentes, nesta seção nós apresentamos cargas de trabalho que podem ser diretamente utilizadas em benchmarks concretos por meio da definição dos parâmetros da carga de trabalho (Tabela 5.4).

Foram estabelecidas Cargas de Trabalho que representam satélites de diferentes escalas: (i) CT1, representa um simulador de satélite do tipo FES para um satélite experimental, pequeno, com apenas 6 federados, 32 atributos intercambiados e tempo real flexível; (ii) CT2, representa um Simulador de Satélites Operacional, de tempo real flexível, para uma satélite típico de Coleta

de Dados (baseado nos SCDs do INPE); e (iii) CT3, representa um Simulador de Satélites Operacional para um satélite de sensoriamento remoto do porte do satélite CBERS.

A Tabela B.11 apresenta as características das cargas de trabalho que podem ser usadas em benchmarks concretos.

Tabela B.11 - Cargas de Trabalho - Simuladores de Satélite.

Satélite	**CT1**	**CT2**	**CT3**
	SAT_1	**SAT_2**	**SAT_3**
Tipo de Satélite	**Experimental (base)**	**Coleta de Dados**	**Sensoriamento Remoto**
Tipo de Simulador	FES	Operacional	Operacional
Número de modelos	6	8	16
Dados intercambiados	32	100	300
Tipo dos dados intercambiados	Double	Double	Double
Serviços assíncronos	Não	Sim	Sim
Parâmetros dos serviços assíncronos	-	10	10
Tipo de dados dos parâmetros	-	Double	Double
Ligação entre modelos	Pares	Circular	Circular
Repetibilidade	Não	Sim	Sim
Frequência do passo de simulação	1 Hz	1 Hz	1 Hz
Comportamento do modelo	Não	Não	Sim
Classes de Objetos - tipos	1	1	1
Classes de Interação - tipos	1	1	1
Classes de Interação - número	1	1	1
Classes de Objetos - instâncias	1	1	1
Modelo de *callback*	HLA_Evoked	HLA_Evoked	HLA_Evoked
Serviços HLA	Não	Não	Não
Linguagem	C++	C++	C++
Versão da biblioteca RTI	RTI 1.3	RTI 1.3	RTI 1.3

B.5 Mudanças

Esta seção do apêndice apresenta a composição da Carga de Mudanças e os requisitos dos métodos de geração/instanciação de mudanças especificados para na execução dos estudos de caso apresentados no Capítulo 7.

B.5.1 Carga de Mudanças

A Tabela B.12 apresenta a evolução da carga de mudanças à medida que foram executados os passos indicados pela metodologia para a sua composição.

Tabela B.12 - Evolução da Carga de Mudanças.

#	Passo	Número de Mudanças
1	Identificar e Classificar Mudanças	109
2	Normalizar as mudanças – parâmetros	72
3	Normalizar as mudanças – efeitos	61
4	Avaliar as mudanças (>= Alto)	51

O primeiro levantamento de mudanças elencou um conjunto de 109 mudanças, que foram normalizadas compondo uma carga de 61 mudanças. Em seguida, quatro especialistas fizeram a análise de exposição das mudanças, conforme indicado na metodologia, considerando um nível de corte “Alto” e chegou-se a uma carga composta por 51 mudanças. A Tabela B.13 apresenta as 61 mudanças normalizadas, o resultado da análise de exposição, bem como os parâmetros usados para a geração ou execução da mudança. Na tabela as mudanças implementadas para a execução dos estudos de caso apresentados no Capítulo 7 estão destacadas em azul.

Tabela B.13 - Carga de Mudanças.

ID		MUDANÇA	SUBCLASSE	EXPOSIÇÃO	FONTE	PARÂMETROS		
						Geração	Instanciação	Ambos
CM1	FImp	RTILib de outro fabricante	Implementação	**Baixo**	Versão da biblioteca RTI		numDeModelos	implementação
CM2	FLang	Federados desenvolvidos em diferentes linguagens de programação	Implementação	**Alto**	Linguagem de programação da biblioteca RTI		numDeModelos	linguagem
CM3	FRTIVer	Mudança na versão da RTILib	Versão	**Alto**	Versão da biblioteca RTI		numDeModelos	versão
CM4	FHard	Inserção de hardware na malha de simulação	Objetivo	**Mandatório**	Template HLA	*template*		numDeModelos
CM5	FStepDur	Mudança na definição de tempo da simulação (tempo real restrito, tempo real flexível, *as-fast-as-possible*)	Objetivo	**Mandatório**	Frequência do passo de simulação			frequência
CM6	FAssinc	Inserção de eventos assíncronos na simulação	Objetivo	**Mandatório**	Serviços assíncronas		numDeModelos	numClassesInt
CM7	FNModels	Aumento no número de modelos que compõem um simulador	Escala	**Muito Alto**	Número de modelos			numDeModelos
CM8	FObject	Aumento no volume de dados intercambiados entre federados (publicação e subscrição de dados) - número de objetos	Escala	**Alto**	Classes de Objetos		numDeModelos	numObjetos
CM9	FAttrib	Aumento no volume de dados intercambiados entre federados (publicação e subscrição de dados) - número de atributos	Escala	**Alto**	Dados intercambiados		numDeModelos	numAtributos
CM10	FNInter	Aumento no número de chamadas a serviços de outros federados	Escala	**Alto**	Classes de Interação - número		numDeModelos	numClassesInt

(Continua)

Tabela B.13 – Continuação.

ID		MUDANÇA	SUBCLASSE	EXPOSIÇÃO	FONTE	PARÂMETROS		
						Geração	Instanciação	Ambos
CM11	FNPar	Aumento no número de parâmetros nas chamadas a serviços de outros federados	Escala	**Baixo**	Parâmetros dos serviços assíncronos		numDeModelos	numParametros
CM12	FAttType	Mudanças nos tipos de dados dos atributos publicados	Projeto/Escala	**Alto**	Tipo dos dados intercambiados		numDeModelos	attTipoDado
CM13	FParType	Mudanças nos tipos de dados dos parâmetros dos serviços chamados	Projeto/Escala	**Baixo**	Tipo dado Parâmetros		numDeModelos	paramTipoDado
CM14	FObjects	Mudanças nos tipos de objetos publicados/atualizados	Projeto/Escala	**Alto**	Classe de Objeto tipos		numDeModelos	numObjetos
CM15	FInter	Mudanças nos tipos de serviços chamados	Projeto/Escala	**Alto**	Classes de Interação - tipos		numDeModelos	numClassesInt
CM16	FCallback	*Threads* de federados para *callback*	Projeto	**Alto**	Template HLA, Modelo de *callback*	*template*	numDeModelos	tipoCallback
CM17	FService	Mudança no uso de serviços HLA (serviço de tempo, etc.)	Projeto	**Alto**	Template HLA	*template*	numDeModelos	serviço
CM18	FStack	Uso da fila de mensagens da RTILib	Projeto	**Médio**	Template HLA	*template*	numDeModelos	nFila
CM19	FStackInt	Uso da fila de mensagens da RTILib - serviços	Projeto	**Baixo**	Template HLA	*template*	numDeModelos	nFila
CM20	Fload	Mudança na carga do modelo	Objetivo	**Muito Alto**	Comportamento do modelo	*template*	numDeModelos	nomeLib
CM21	FCallMOM	Chamadas a serviços MOM	Projeto	**Baixo**	Template HLA	*template*	numDeModelos	tipoChamada numChamada
CM22	FCallBackProj	Uso do serviço *Evoke Callback*	Projeto	**Alto**	Template HLA	*template*	numDeModelos	callMin callMax

(Continua)

Tabela B.13 – Continuação.

ID		MUDANÇA	SUBCLASSE	EXPOSIÇÃO	FONTE	PARÂMETROS		
						Geração	Instanciação	Ambos
CM23	FStartStop	Entrada e saída de Federados na federação	Perfil de uso	**Médio**	Federação			script numDeModelos
CM24	FAccel	Variação na frequência do passo de simulação	Perfil de uso	**Mandatório**	Frequência do passo de simulação		numDeModelos	frequência
CM25	FAdvisory	Habilitar as chaves de configuração (*advisory keys*)	Configuração	**Alto**	Chaves de configuração		numDeModelos	chave
CM26	FTransport	Mudança no tipo de transporte (confiável e não-confiável)	Configuração	**Alto**	Tipo de transporte		numDeModelos	tipoTransporte porAtributo
CM27	FUpdate	Variações no período de atualização de dados e serviços pela RTI	Configuração	**Muito Alto**	Taxa de Atualização		numDeModelos	taxaAtualiza
CM28	FEnaMOM	Serviços MOM	Configuração	**Alto**	Federação		nomeArquivo	
CM29	FErrSaveFile	Erro na configuração do diretório do arquivo de salvamento de estado	Falhas Operacionais	**Mandatório**	Federação		nomeArquivo	
CM30	FDelSaveFile	Deleção do arquivo de salvamento de estado	Falhas Operacionais	**Muito Alto**	Federação		nomeArquivo	
CM31	FStopFed	Término forçado de um federado da federação	Falhas Operacionais	**Muito Alto**	Federação		numDeModelos	tipoProcesso
CM32	FStopRTI	Término forçado do processo RTIExec	Falhas Operacionais	**Mandatório**	Federação			
CM33	FErrIni	Erro na inicialização de federados	Falhas Operacionais	**Muito Alto**	Federação		numDeModelos	
CM34	FDelMIM	Corromper (remover) o arquivo de inicialização RID	Falhas Operacionais	**Mandatório**	Federação		nomeArquivo	

(Continua)

Tabela B.13 – Continuação.

ID		MUDANÇA	SUBCLASSE	EXPOSIÇÃO	FONTE	PARÂMETROS		
						Geração	Instanciação	Ambos
CM35	FDelFom	Corromper (remover) o arquivo de configuração FOM de um federado	Falhas Operacionais	**Mandatório**	Federação		numDeModelos	
CM36	FErrCall	Erro na configuração do tipo de *callback*	Falhas Operacionais	**Muito Alto**	Federação			callbackConf numDeModelos
CM37	FSoftFault	Falhas de software ODC (operador) – Obs.: operadores definidos por Durães e Madeira (2006)	Falhas de Software	**Muito Alto**	Federados			operator numDeModelos
CM38	FIntFault	Falhas de software - Interface	Falhas de Software	**Mandatório**	Federados		numRodadas	arquivoTipos
CM39	FCallBckFault	Falha no uso do serviço *Evoke Callbac*k (frequência)	Falhas de Software	**Alto**	Federados	template	numDeModelos	Frequencia
CM40	CNetType	Mudança no tipo de canal de comunicação (Lan, Wan, Wireless, memória compartilhada)	Recurso	**Muito Alto**	Tipo			tipoRede
CM41	CNetBand	Mudança na banda da rede (aumento, diminuição)	Recurso	**Mandatório**	Banda da rede			banda
CM42	CTraf	Variação no tráfego da rede	Perfil de uso	**Muito Alto**	Canal de comunicação		nomeScript	
CM43	CNetPar	Mudança na configuração de parâmetros de rede	Configuração	**Muito Alto**	Canal de comunicação			operator value
CM44	CNetFail	Falhas na interface da rede	Falhas de Hardware	**Alto**	Canal de comunicação		nomeScript	
CM45	CNetCon	Falhas na conexão da rede	Falhas de Hardware	**Muito Alto**	Canal de comunicação		nomeScript	
CM46	CNetCorr	Corrupção da rede	Falhas de Software	**Alto**	Canal de comunicação		nomeScript	

(Continua)

Tabela B.13 – Continuação.

ID		MUDANÇA	SUBCLASSE	EXPOSIÇÃO	FONTE	PARÂMETROS		
						Geração	Instanciação	Ambos
CM47	CSem	Falha na semântica da troca de mensagens	Falhas de Software	**Médio**	Canal de comunicação		numDeModelos	
CM48	OSPatchExec	Atualização no Sistema Operacional - RTIExec	Versão	**Alto**	Sistema Operacional - patches			patch
CM49	OSPatchLib	Atualização no Sistema Operacional - RTILib	Versão	**Alto**	Sistema Operacional - versão		numDeModelos	patch
CM50	OSVerExec	Diferentes sistemas operacionais (versões, outros sistemas, outras classes) - RTIExec	Versão /Implementação	**Muito Alto**	Sistema Operacional - tipo			sistemaOp
CM51	OSVerLib	Diferentes sistemas operacionais (versões, outros sistemas, outras classes) - RTILib	Versão /Implementação	**Mandatório**	Sistema Operacional da biblioteca RTI		numDeModelos	sistemaOp
CM52	FUnavailFed	Indisponibilidade de máquina executando um federado	Falhas Operacionais	**Muito Alto**	Hardware		numDeModelos	maquinas
CM53	FUnavailRTI	Indisponibilidade de máquina executando RTIExec	Falhas Operacionais	**Muito Alto**	Hardware			
CM54	OSSofTFault	Falhas de software ODC (operador) Obs.: operadores definidos por Durães e Madeira (2006)	Falhas de Software	**Médio**	Sistema Operacional			operator
CM55	HProc	Mudança na capacidade do processador (aumento, diminuição)	Recurso	**Mandatório**	Hardware - processador			
CM56	HNProc	Mudança no número de CPUs (aumento, diminuição)	Recurso	**Mandatório**	Hardware - processador			

(Continua)

Tabela B.13 – Conclusão.

ID		MUDANÇA	SUBCLASSE	EXPOSIÇÃO	FONTE	PARÂMETROS		
						Geração	Instanciação	Ambos
CM57	HMem	Mudança na capacidade de memória da máquina (aumento, diminuição)	Recurso	**Muito Alto**	Hardware - memória			
CM58	HNType	Mudança no tipo de hardware utilizado (*tablets*, celulares)	Recurso	**Médio**	Hardware - categoria			arquitetura
CM59	HNMach	Mudança no número de máquinas na simulação (aumento, diminuição)	Escala	**Muito Alto**	Hardware - cardinalidade		máquinas	numMáquinas
CM60	HProcUse	Variação no padrão de uso dos processadores	Perfil de uso	**Muito Alto**	Hardware		nomeScript	numProcessos
CM61	HMemUse	Variação no padrão de uso da memória	Perfil de uso	**Muito Alto**	Hardware		nomeScript	numProcessos

B.5.2 Métodos de Geração e Instanciação de Mudanças

Como visto na metodologia, cada mudança tem associada a ela um conjunto de parâmetros e métodos usados na sua geração e instanciação. Vale salientar que um mesmo método pode ser utilizado para gerar ou instanciar mais de uma mudança. Esta subseção apresenta os requisitos dos métodos associados às mudanças que foram implementadas para a execução dos estudos de caso apresentados no Capítulo 7. As Tabelas B.14 a B.20 apresentam os requisitos de métodos de geração e execução de mudanças.

a) Métodos de Geração de Mudanças

Tabela B.14 - Método - *FEDModification.*

Método	***FEDModification***
Operadores	FRTIVer, FStepDur, FAssinc, FNModel, FObjects, FAttrib, FNInter, FNPar, FAttType, FParType, FObject, FInter, FUpdate, FAdvisory
Descrição	Modificação em parâmetros da federação.
Parâmetros	operador: operador de mudança tipo: tipo do dado da origem da mudança mudança: valor da mudança a ser aplicada
Requisitos	
R.1	Usando a informação sobre tipo de dado, o serviço deve ser capaz de converter o valor da mudança;
R.2	O serviço deve ser capaz de, usando o operador de mudança (*operator*), modificar a fonte de mudança com o valor recebido.

Tabela B.15 - Método - *FEDModificationChange.*

Método	***FEDModificationChange***
Operadores	FAccel
Descrição	Modificação de parâmetros da federação e geração de federado instanciador.
Parâmetros	operador: operador de mudança tipo: tipo do dado da origem da mudança mudança: mudança a ser aplicada (classe mudança)
Requisitos	
R.1	Usando a informação sobre tipo de dado, o serviço deve ser capaz de converter o valor da mudança;
R.2	O serviço deve ser capaz de, usando o operador de mudança, modificar a fonte de mudança com o valor recebido.
R.3	O serviço deve ser capaz de gerar o federado para a instanciação de cada mudança (instanciador) que considera: valor do parâmetro, gatilho, duração e cardinalidade.
R.4	O federado gerado deve ser capaz de alterar o passo de simulação a cada aplicação da mudança de acordo com o valor do parâmetro de mudança.

Tabela B.16 - Método - *LoadProcessor.*

Método	***LoadProcessor***
Operadores	HProcUse HMemUse
Descrição	Instanciação de processos de mudança.
Parâmetros	operador: operador de mudança tipo: tipo do dado da origem da mudança mudança: mudança a ser aplicada nomeScript: processo a ser executado numProcessos: número de processos
Requisitos	
R.1	O serviço deve ser capaz de obter o nome do script a ser executado.
R.2	O serviço deve ser capaz de gerar um federado instanciador de processos que: a) Leva em consideração o gatilho, a duração e a cardinalidade. Executa o processo indicado no parâmetro nomeScript como um processo a parte; b) Deve executar tantos processos quanto indicar o parâmetro numProcessos. c) No final da rodada deve executar script de recuperação que elimina todos os processos remanescentes.

Tabela B.17 - Método - *InterfaceFaultInjection.*

Método	***InterfaceFaultInjection***
Operadores	FIntFault
Parâmetros	arquivoTipos: nome do arquivo de configuração de tipos
Requisitos	
R.1	O serviço deve ser capaz de ler um arquivo XML de configuração de tipos recebido como parâmetro.
R.2	O serviço deve ser capaz de ler o arquivo de configuração de serviços HLA da versão da RTI em uso (indicada pelo valor do parâmetro *versão* da carga de trabalho)
R.3	Para cada serviço HLA, este serviço deve ser capaz de gerar um federado de falha que chama o serviço HLA em questão. O federado com falha, a cada execução, deve ser capaz de chamar o serviço usando um valor inválido em cada um dos seus parâmetros (um erro por chamada). Deve ser possível chamar o serviço com valores aceitáveis para os parâmetros.
R.4	O serviço deve gerar um federado instanciador de federados com falha que seja capaz de: a) Executar, em um processo à parte, federados com falha indicando qual o número da rodada (o serviço/parâmetro/valor que irá falhar); b) A cada rodada de simulação, instanciar o federado com falha o número de vezes indicado na cardinalidade da mudança; c) Deixar a falha ativa pelo tempo definido na duração da mudança; d) No final da rodada, executar o script de recuperação que elimina os processos remanescentes.

b) *Métodos de Execução de Mudanças*

Tabela B.18 - Método - *Substitution.*

Método	***Substitution***
Operadores	FRTIVer, FStepDur, FAssinc, FNModel, FObject, FAttrib, FNInter, FNPar, FAttType, FParType, FObjects, FInter, FAcel, FUpdate, FAdvisory
Parâmetros	operador: operador de mudança tipo: tipo do dado da origem da mudança mudança: mudança a ser aplicada numDeModelos: número de modelos a serem substituídos na federação da carga de trabalho.
Requisitos	
R.1	O serviço deve ser capaz de substituir federados da carga de trabalho definidos pelo parâmetro numDeModelos, pelos federados gerados com a mudança indicada pelo operador de mudança associado.
R.2	O serviço deve ser capaz de executar a carga de trabalho com os federados modificados.

Tabela B.19 - Método - *Distribution.*

Método	***Distribution***
Operadores	HNMach
Parâmetros	operador: operador de mudança tipo: tipo do dado da origem da mudança mudança: mudança a ser aplicada numMaquinas: número de máquinas da federação. máquinas: nome das máquinas a serem usadas na distribuição.
Requisitos	
R.1	O serviço deve ser capaz distribuir os federados da federação nas máquinas indicadas pelos parâmetros numMaquinas e máquinas (nomes).

Tabela B.20 - Método - *InjectFailModel.*

Método	***InjectFailModel***
Operadores	FIntFault, FSoftFault
Parâmetros	operador: operador de mudança numRodada – número da rodada
Requisitos	
R.1	O serviço deve ser capaz de instanciar o federado instanciador de falhas de interface passando o número da rodada.