MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2019/05.21.13.10-TDI

ABORDAGEM SISTEMÁTICA DE TESTES DE SOFTWARE COMUNICANTE EMBARCADO EM NANOSATÉLITES COM FOCO EM FALHAS DE INTEROPERABILIDADE

Carlos Augusto Paiva Lameirinhas da Conceição

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Maria de Fátima Mattiello-Francisco, e Leonardo Montecchi, aprovada em 30 de maio de 2019.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3TBGC2H>

INPE
São José dos Campos
2019

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21c/2019/05.21.13.10-TDI

ABORDAGEM SISTEMÁTICA DE TESTES DE SOFTWARE COMUNICANTE EMBARCADO EM NANOSATÉLITES COM FOCO EM FALHAS DE INTEROPERABILIDADE

Carlos Augusto Paiva Lameirinhas da Conceição

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pelos Drs. Maria de Fátima Mattiello-Francisco, e Leonardo Montecchi, aprovada em 30 de maio de 2019.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3TBGC2H>

INPE
São José dos Campos
2019

Dados Internacionais de Catalogação na Publicação (CIP)

Conceição, Carlos Augusto Paiva Lameirinhas da.

C744a Abordagem sistemática de testes de software comunicante embarcado em nanosatélites com foco em falhas de interoperabilidade / Carlos Augusto Paiva Lameirinhas da Conceição. – São José dos Campos : INPE, 2019.

xxix + 328 p. ; (sid.inpe.br/mtc-m21c/2019/05.21.13.10-TDI)

Tese (Doutorado em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2019.

Orientadores : Drs. Maria de Fátima Mattiello-Francisco, e Leonardo Montecchi.

1. Cubesat. 2. Dependabilidade. 3. Interoperabilidade. 4. Robustez. 5. Tratamento de falha. I.Título.

CDU 629.78:004.415.53

Aluno (a): *Carlos Augusto Paiva Lameirinhas da Conceição*

Título: "ABORDAGEM SISTEMÁTICA DE TESTES DE SOFTWARE COMUNICANTE EMBARCADO EM NANOSATÉLITES COM FOCO EM FALHAS DE INTEROPERABILIDADE"

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

Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais

Dr. Walter Abrahão dos Santos

Presidente / INPE / São José dos Campos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dra. Maria de Fátima Mattiello-Francisco

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

( ) Participação por Video - Conferência

( ) Aprovado *( ) Reprovado*

Dr. Leonardo Montecchi

Orientador(a) / UNICAMP / Campinas - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dra. Ana Maria Ambrosio

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

( ) Participação por Video - Conferência

( ) Aprovado *( ) Reprovado*

Dr. Otávio Santos Cupertino Durão

Membro da Banca / INPE / SJCampos - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Dr. Francisco Carlos Parquet Bizarria

Convidado(a) / UNITAU / Taubaté - SP

( ) Participação por Video - Conferência

(X) Aprovado *( ) Reprovado*

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 30 de maio de 2019

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

Engenharia e Tecnologia Espaciais/Eng. Gerenc. de Sistemas Espaciais

Dra. Emília Villani

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

( ) Participação por Video - Conferência

(X) Aprovado ***( ) Reprovado***

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

São José dos Campos, 30 de maio de 2019

Dedico este trabalho aos que me deram apoio para alcançar essa minha nova conquista.

“O pessimista queixa-se do vento. O otimista espera que ele mude. O realista ajusta as velas”.

Willian George Ward – Teólogo Inglês

AGRADECIMENTOS

Agradeço a Deus por permitir realizar mais uma conquista.

Aos professores Dra. Maria de Fátima Mattiello-Francisco e Dr. Leonardo Montecchi, pelo profissionalismo, competência e comprometimento na condução da orientação deste trabalho.

Aos que tiveram paciência nos momentos em que eu não estava com paciência e aos que me deram uma palavra de incentivo mesmo sem saber que eram palavras importantes no momento certo.

RESUMO

Os estudos desenvolvidos nesta tese apresentam uma abordagem sistemática de testes de software comunicante embarcado em nanosatélite com foco em tratamento de falhas de interoperabilidade. A abordagem destina-se a identificar e evitar falhas entre subsistemas comunicantes no início do processo de verificação e validação (V&V) de missões espaciais de baixo custo e curto ciclo de desenvolvimento, tais como as missões de satélites de pequeno porte que utilizam o padrão Cubesat. As plataformas Cubesats são uma evolução no setor espacial permitindo levar em órbita cargas úteis científicas e inovadoras a baixo custo. Entretanto, o baixo orçamento e tempo de desenvolvimento reduzido comprometem o processo de V&V, penalizando a qualidade requerida na realização dos testes de integração dos subsistemas de carga útil desses satélites. Considerando o fato de as funcionalidades dos sistemas embarcados implementadas por software serem crescentes, a abordagem apresenta um Sistema de Teste com Arquitetura Escalável (STAE) para apoiar o processo de V&V na integração de subsistemas embarcados intensivos em software. A abordagem apresentada nesta pesquisa de tese visa antecipar, no processo de desenvolvimento de nanosatélites, possíveis falhas de interação da plataforma com suas cargas úteis e facilita a reutilização da arquitetura de teste em diferentes fases de uma mesma missão ou em satélites da mesma família, por meio da aplicação combinada das abordagens "*Model-Driven Engineering*" e "*Model-Based Testing*". O método desenvolvido sistematiza: i) a concepção de modelos comportamentais de subsistemas comunicantes buscando verificar os requisitos de interoperabilidade dos subsistemas por meio da validação dos modelos (MIL - *Model-in-the-loop*), ii) a geração automática de código computacional a partir dos modelos validados, por ferramentas MDE, permitindo a verificação dos requisitos de interoperabilidade em ambiente simulado, com o uso do barramento de comunicação real (SIL - *Software-in-the-loop*), iii) a evolução dos modelos para que casos de testes abstratos possam ser gerados por ferramentas MBT, iv) a execução de casos de testes para validar os subsistemas reais (HIL - *Hardware-in-the-loop*) na fase de integração, e v) a possibilidade de injetar falhas por meio de um Mecanismo Emulador de Falhas (FEM), o qual permite testar a interação dos subsistemas reais em termos de requisitos de robustez especificados. Para auxiliar na especificação de requisitos de robustez, a abordagem apresentada utiliza uma planilha de dependabilidade, que aplica os conceitos da árvore de dependabilidade e análise causa-efeito para mitigações de falha. Para demonstrar sua efetividade, a abordagem desenvolvida no âmbito desta tese foi aplicada na V&V da interoperabilidade dos subsistemas intensivos em software embarcados no nanosatélite NanosatC-BR2, em desenvolvimento no INPE. Os testes de interação foram realizados nos subsistemas Computador de Bordo e a carga útil denominada Sonda de Langmuir. Os resultados obtidos proporcionaram o tratamento de falhas durante o processo de V&V de sistemas intensivos em software embarcados do nanosatélite NanosatC-BR2.

Palavras-chave: missão espacial. Cubesat. nanosatélite. verificação e validação. dependabilidade. interoperabilidade. robustez. tratamento de falha.

SYSTEMATIC TESTING APPROACH OF COMMUNICANTING SOFTWARE EMBEDDED IN NANOSATELLITES FOCUSING ON INTEROPERABILITY FAULTS

ABSTRACT

The studies developed in this thesis present a systematic testing approach of communicating software embedded in nanosatellites focusing on the treatment of interoperability faults. The approach is intended to support the verification and validation (V&V) process of low-cost space missions, which have short development cycle, such as small satellite missions based on the Cubesat standard. The Cubesats platforms are an evolution in the space sector, allowing to bring into the orbit scientific and innovative payloads at low cost. However, low budget and low development time compromise the V&V process, penalizing the quality required to perform the integration tests of the payload subsystems of these satellites. Considering the fact that the functionalities of the embedded systems implemented by software are increasing, the approach introduces a Scalable Architecture Test System (STAE) to support the V&V process in the integration of embedded software intensive subsystems. The approach proposed in this research thesis aims to anticipate, in the nanosatellite development process, common faults related to the satellite platform interaction with its payloads. Moreover, it facilitates the reuse of the test architecture in different phases of the same mission or in satellites of the same family, through the combined application of the Model-Driven Engineering and Model-Based Testing tools. The developed approach systematizes: i) the design of behavioral models of communicating subsystems in order to verify the subsystem interoperability requirements through model validation (MIL - Model-in-the-loop), ii) the automatic generation of computational code from the validated models, by MDE tools, allowing the verification of interoperability requirements in a simulated environment, with the use of the SIL (Software-in-the-loop), since the generated codes are embedded in STAE components, (iii) the evolution of models so that abstract test cases can be generated by MBT tools, iv) the execution of test cases to validate the real subsystems (HIL - Hardware-in-the-loop) in the integration phase, and v) the possibility of injecting faults through a Fault Emulator Mechanism (FEM), which allows testing the interaction of the actual subsystems in terms of specified robustness requirements. To support the specification of robustness requirements, the proposed approach uses a dependability worksheet, which applies the concepts of the dependability tree and cause-and-effect analysis for fault mitigations. In order to demonstrate the approach effectiveness, it was applied in the V&V of the software interoperability embedded into the NanosatC-BR2 nanosatellite, under development at INPE. The interactions between the On-board Computer subsystem and the payload called the Langmuir Probe were the testing target.

The results obtained allowed to anticipate the treatment of faults during the V&V process of the software intensive systems embedded in the NanosatC-BR2 nanosatellite.

Keywords: Space mission. Cubesat. nanosatellite. verification and validation. dependability. interoperability. robustness. treatment of fault.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

ACK	Reconhecido (*Acknowledge*)
ADCS	Sistema de Controle de Atitude e Determinação (*Attitude Control System and Determination*)
ANTS	Sistema de Antena (*Antenna System*)
AOCS	Sistema de Controle de Atitude (*Attitude and Orbit Control System*)
ASIC	Aplicação Específica de Circuito Integrado (*Application Specific Integrated Circuit*)
ATCM	Caso de Teste Automático Baseado em Modelos (*Automatic Test Case Based on Models*)
ATM	Caixa Eletrônico (*Automated Teller Machine*)
BOB	Placa de Saída (*Break_Out Board*)
CASE	Engenharia de Software Assistida por Computador (*Computer-Aided Software Engineering*)
CoFI	Conformidade e Injeção de Falhas (*Conformance and Fault Injection*)
COTS	Componentes Comerciais (*Commercial Off-The-Shelf*)
C&DH	Controle e Tratamento de Dados (*Control and Data Handling*)
DAS	Sistema de Determinação de Atitude (*Determination Attitude System*)
DSL	Linguagem Específica de Domínio (*Domain-Specific Language*)
EGSE	Equipamentos de Suporte em Solo para Teste Elétrico (*Electrical Ground Support Equipment*)
ESA	Agência Espacial Europeia (*European Space Agency*)
FEM	Mecanismo Emulador de Falha (*Fault Emulator Mechanism*)
FMEA	Análise dos Modos de Falha e seus Efeitos (*Failure Mode and Effect*

	Analysis)
FPGA	Matriz com portas configuráveis (*Field Programmable Gate Array*)
FMS	Máquina de Estado Finita (*Finite State Machine*)
FTA	Análise de Árvore de Falhas (*Fault Tree Analysis*)
GSE	Equipamento para Suporte em Solo (*Ground Support Equipment*)
HIL	Hardware no loop (*Hardware-in-the-loop*)
IBM	Máquinas de Negócios Internacionais (*International Business Machines*)
INCOSE	Conselho Internacional de Engenharia de Sistemas (*International Council on Systems Engineering*)
HIL	Hardware no loop (*Hardware-in-the-loop*)
IDEF0	Definição de Integração para Modelagem de Funções - (*Integration Definition for Function Modeling*)
IDM	Modelo Intermediário de Dependabilidade (*Intermediate Dependability Model*)
IGIS	Interface Genérica de Sistema (*Generic Interface System*)
INPE	Instituto Nacional de Pesquisas Espaciais
InRob	Interoperabilidade e Robustez (*Interoperability and Robustness*)
InRob-UML	Interoperabilidade e Robustez- Linguagem de Modelagem Unificada (*Interoperability and Robustness - Unified Modeling Language)*
IP	Protocolo de Internet (*Internet Protocol*)
ISIS	Inovação em Solução no Espacial (Innovative Solutions In Space)
I2C	Circuito Inter-Integrado (*Inter-Integrated Circuit*)
LSB	Bit Menos Significativo (*Less Significant Bit*)

MBT	Teste Baseado em Modelo (*Model-Based Testing*)
MBSE	Engenharia Sistemas Baseados Modelos (*Model-Based System Engineering*)
MBST	Teste de Segurança Baseado em Modelo (*Model-Based Security Testing*)
MDA	Arquitetura Dirigida por Modelo (*Model-Driven Architecture*)
MDD	Desenvolvimento Dirigido por Modelo (*Model-Driven Development*)
MDE	Engenharia Dirigida por Modelos (*Model-Driven Engineering*)
MIL	Modelo no loop (*Model-in-the-loop*)
MME	Modelador de Máquina de Estados
MOF	Facilidade de Meta-Objeto (*Meta-Object Facility*)
MSB	Bit Mais Significativo (*Most Significant Bit*)
MYSQL	Banco de Dados que utiliza: Linguagem de Consulta Estruturada (*Structured Query Language*)
NACK	Não reconhecido (*No Acknowledge*)
NASA	Administração Nacional da Aeronáutica e Espaço (*National Aeronautics and Space Administration*)
OBC	Computador de Bordo (*On-Board Computer*)
OMG	Grupo de Gerenciamento de Objetos (*Object Management Group*)
PC	Computador Pessoal (*Personal Computer*)
PIL	Processador no loop (*Processor-in-the-loop*)
PIM	Modelo Independente de Plataforma (*Platform Independent Model*)
PSM	Modelo Específico de Plataforma (*Platform Specific Model*)
PHPMYADMIN	Administrador Pessoal de Página Caseiro (*Personal Home Page My*

	Admin)
R/W	Ler / Escrever (*Read / Write*)
SBS	Especificação Baseada em Sequência (*Sequence Based Specification*)
SCL	Sinal de Relógio (*Signal Clock*)
SDA	Sinal de Dados (*Signal Data*)
SIL	Software no loop (*Software-in-the-loop*)
SQL	Linguagem de Consulta Estruturada (*Structured Query Language*)
ST	Sistema de Teste
STAE	Sistema de Teste com Arquitetura Escalável
SUT	Sistemas sob Teste (*System Under Test*)
SW-FMEA	Análise dos Modos de Falha de Software e seus Efeitos (Software-*Failure Mode and Effect Analysis*)
SysML	Linguagem de Modelagem de Sistemas (*System Modeling Language*)
TC	Telecomando (*Telecommand*)
TM	Telemetria (*Telematry*)
TML	Linguagem de Modelagem de Tráfego (*Traffic Modeling Languages*)
TRXUV	Transmissão e Recepção de Dados (*Data Transmission and Reception*)
UML	Linguagem de Modelagem Unificada (*Unified Modeling Language)*
UHF	Frequência Ultra-Alta (*Ultra High Frequency*)
UX	EXperiência do Usuário (*User eXperience*)
VHF	Frequência Muito Alta (*Very High Frequency*)
V&V	Verificação e Validação

SUMÁRIO

Pág.

1 INTRODUÇÃO

Os satélites de pequeno porte foram desenvolvidos para serem utilizados em pesquisas e testes de novas tecnologias em ambiente espacial (RABASA, 2015). Esses satélites são produzidos com baixo custo comparado aos satélites de grande porte e as missões duram em torno de dois anos (HELVAJIAN; JANSON, 2008). No ciclo de desenvolvimento de satélites de pequeno porte, que utilizam o padrão Cubesat, referenciados nessa tese por nanosatélites, nem todos os tipos de testes recomendados pelas normas espaciais têm sido realizados em razão do curto período de tempo de desenvolvimento da missão e baixo orçamento. A falta de testes pode levar a ocorrências de falhas na operação e possíveis fracassos das missões. O estado da prática tem demonstrado que muitas falhas na operação da missão estão relacionadas à interação entre os subsistemas embarcados, muitos deles intensivos em software. O subsistema de gestão de bordo é um exemplo típico de sistema intensivo em software por ter a maioria das suas funcionalidades realizadas por software embarcado no computador de bordo da plataforma do satélite (ASUNDI; FITZ-COZY, 2013).

Em nanosatélites que utilizam plataforma padrão Cubesat, cabe ao subsistema de gestão de bordo interagir com os subsistemas da carga útil da missão para fins de manipulação dos dados da missão. Esses subsistemas frequentemente são desenvolvidos por equipes ou organizações diferentes (RABASA, 2015), sendo que os desenvolvedores, não necessariamente, utilizam a mesma abordagem de desenvolvimento para atender aos requisitos do sistema. Esses subsistemas são testados inicialmente de forma isolada no ambiente de desenvolvimento do desenvolvedor. Cada desenvolvedor testa seu subsistema em sua plataforma de teste buscando verificar se os requisitos estão sendo atendidos (HERPEL et al., 2016). Assim, a fase de integração dos subsistemas é extremamente relevante no processo de verificação e validação (V&V) de missões espaciais. Entretanto, no ciclo de desenvolvimento de missões espaciais, essa fase de testes ocorre quando grande parte dos subsistemas da missão já foram desenvolvidos isoladamente. A realização de testes de

subsistemas comunicantes antes da fase de integração do nanosatélite tem por objetivo detectar erros ainda na fase de desenvolvimento de cada subsistema (CHARTRES et al., 2014), evitando problemas de interoperabilidade e dependabilidade dos subsistemas do nanosatélite.

Conforme alegado por Langer et al. (2016), um aspecto que vem sendo observado e analisado sob a ótica de confiabilidade é que falhas em missões de Cubesats deve-se em grande medida à ausência de testes funcionais nos subsistemas em uma fase inicial no ciclo de desenvolvimento do projeto.

De acordo com Jacklin (2015), uma pesquisa realizada, com mais de 165 artigos disponíveis em domínio público, revelou pouco uso de abordagens de verificação e validação (V&V) de software no desenvolvimento de pequenos satélites. Dentre os métodos utilizados para verificação e validação de software envolvendo Cubesats referenciados nesta pesquisa destacam-se:

- Métodos formais, os quais são técnicas matemáticas voltadas a casos específicos.
- Software tolerante a falhas e verificação por monitoramento em tempo de execução, o qual possibilita detectar erros que não foram detectados antes do lançamento ou que surgiram após o lançamento.
- Simulação e teste, sendo um método muito utilizado para V&V de software de satélites em geral. A simulação permite testar os software e algoritmos antes dos componentes reais serem testados na fase de integração. Dessa forma, erros podem ser detectados na fase inicial do ciclo de desenvolvimento. Após as simulações serem validadas os componentes reais são testados individualmente buscando a verificação e validação individual e posteriormente em integração.
- Projeto e testes de software baseado em modelo, permitem que casos de testes sejam aplicados nos modelos comportamentais do software no inicio do ciclo de desenvolvimento validando os requisitos de software mais cedo. A partir dos modelos validados, ferramentas permitem gerar códigos de forma automatizada, agilizando dessa

forma a verificação do software antes da fase de integração, além de economizar tempo e custo. Os códigos são gerados sem interferência manual. Os modelos podem ser desmembrados em metamodelos permitindo verificar se os componentes interagem conforme os requisitos antes dos códigos reais serem integrados. Os modelos também podem ser reutilizados nas fases evolutivas dos testes.

Portanto, a tese apresentada teve por objetivo conceber, desenvolver e aplicar uma abordagem sistemática de testes de software comunicante embarcado em nanosatélites com foco no tratamento de falhas de interoperabilidade. A abordagem utiliza um Sistema de Teste com Arquitetura Escalável (STAE) para sistematizar os testes de interoperabilidade entre sistemas e apoiar o processo de V&V. A abordagem sistemática é fruto do uso combinado dos conceitos e ferramentas da Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE) e Teste Baseado em Modelo (*Model-Based Testing* - MBT). A abordagem MDE auxilia transformar de forma automatizada a modelagem comportamental dos requisitos em códigos computacionais executáveis. A abordagem MBT auxilia a geração automatizada de casos de teste a partir da modelagem dos subsistemas do nanosatélite.

A abordagem sistemática concebida nessa tese é aplicada na fase inicial do processo de V&V do NanosatC-BR2 (SCHUCH et al. 2017) visando identificar e tratar possíveis falhas de interoperabilidade de seus subsistemas antes da fase de integração. No ciclo de desenvolvimento de um projeto, falhas identificadas nas fases iniciais minimizam gastos e esforços de retrabalho, inviáveis em cenários de baixo orçamento. A utilização da abordagem sistemática desenvolvida nessa tese permite antecipar a identificação eo tratamento de falhas dos subsistemas comunicantes do satélite no ciclo de desenvolvimento. O tratamento de falhas adotado nesta tese compreende identificar possíveis falhas de especificação de requisitos, por meio de simulação dos subsistemas do nanosatélite e implementar procedimentos para evitar que essas falhas ocorram em ambiente real, utilizando conceitos de dependabilidade, interoperabilidade e robustez. Para a validação de robustez o STAE conta com Mecanismo Emulador de Falhas (FEM) que atua no canal de comunicação

entre dois subsistemas reais comunicantes. As possíveis falhas identificadas são relacionadas em uma planilha de dependabilidade e uma tabela FMEA, desenvolvidas com base nos conceitos de dependabilidade da Taxonomia de Avizienis et al. (2004). Essa planilha permite analisar outras possíveis falhas decorrentes daquelas detectadas durante os testes. A tabela relaciona mecanismos e mitigações para evitar as falhas identificadas. A arquitetura de teste que apoia o STAE possui o atributo de ser expansível e reutilizável em diferentes fases do projeto conforme a evolução dos testes. A arquitetura é utilizada para apoiar a execução de códigos que simulam o comportamento dos subsistemas comunicantes (software embarcado em hardware do STAE) verificando se os requisitos de interoperabilidade são atendidos conforme especificados. O propósito é testar o comportamento desses subsistemas simulados, porém com o uso do barramento real de comunicação. Falhas identificadas são tratadas com melhorias no modelos simulados até que a comunicação entre os modelos simulados se comporte como esperado. Posteriormente, a medida que cada subsistema real (*harware-in-the-loop*) se torna disponível, o respectivo modelo simulado é substituído pelo subsistema real no STAE. Requisitos de robustez implementados pelos subsistemas reais são testados com o uso do FEM. Assim, de forma incremental a interoperabilidade dos subsistemas reais no barramento são validados. Essa sistematização na construção de modelos, geração de testes e execução dos testes em ambientes simulado e real no STAE, evidencia que a arquitetura STAE é reutilizável em várias fases do processo de V&V de um dado satélite bem como de outros satélites da mesma família.

1.1. **Motivação**

Esta tese tem como motivação buscar soluções que sistematizam os testes de integração de subsistemas comunicantes intensivos em software embaraçados em missões espaciais de baixo custo e curto ciclo de desenvolvimento, e seu reuso no processo de V&V.

Com o propósito de diminuir os custos de projetos de missões de nanosatélites, desenvolvidas geralmente em ambiente universitário, laboratórios simplificados e tempo de teste reduzido têm sido adotados. Desta forma, nem todos os testes recomendados em projetos espaciais são realizados, levando à ocorrência de casos de fracasso da missão (ASUNDI; FITZ-COZY, 2013). Teste é reconhecida como uma técnica efetiva de verificação para garantir a qualidade e evidências de dependabilidade na operação de uma missão (AVIZIENIS et al., 2004).

A realização de testes é um requisito fundamental no desenvolvimento de satélite, sendo aplicado em várias fases do projeto, em especial na concepção, integração, verificação e inspeção do satélite. A execução dos testes em cada fase de projeto é altamente dependente do ambiente ou sistema de teste adequado. Por exemplo, o teste elétrico é categoria de destaque nos projetos de satélites, pois necessita de equipamentos e instalações específicas para serem realizados, os quais são denominados Equipamentos de Suporte em Solo para Teste Elétrico (*Electrical Ground Support Equipment* - EGSE). Esses equipamentos são desenvolvidos de acordo com os requisitos particulares de cada tipo de satélite. No desenvolvimento de missões de nanosatélites que utilizam o padrão CubeSat, o EGSE é parte no modelo de engenharia do satélite e pode ser utilizado para apoiar a execução dos testes elétricos e funcionais de subsistemas que compõem a missão. Entretanto, o uso de EGSE nos testes funcionais de software comunicantes embarcados nos subsistemas do satélite tem sido pouco explorado e sistematizado.

Assim, um dos desafios desta tese foi conceber um sistema de teste possível de ser reutilizado em diferentes fases do desenvolvimento de um dado nanosatélite para apoiar os testes funcionais de interoperabilidade de subsistemas intensivos em software ao longo do processo de V&V, tanto de um nanosatélite quanto no desenvolvimento de outros nanosatélites da mesma missão como no caso de constelação de satélites.

1.2. **Objetivo geral**

Desenvolver uma abordagem que permita sistematizar a identificação e tratamento de falhas de interoperabilidade entre subsistemas intensivos em software comunicantes, embarcados em missões de nanosatélites. A abordagem destina-se a apoiar o processo V&V de missões espaciais de baixo custo e curto ciclo de desenvolvimento, tais como as missões de satélites de pequeno porte que utilizam o padrão Cubesat.

1.2.1. **Objetivos específicos**

Com o propósito de realizar e cumprir o objetivo geral da abordagem proposta os objetivos específicos desta tese são:

- Conceber um Sistema de Teste com Arquitetura Escalável (STAE), o qual possa ser reutilizado em diferentes fases de uma mesma missão ou em satélites da mesma família, para apoiar a identificação e tratamento de falhas de subsistemas intensivos em software comunicantes embarcados em nanosatélite durante o ciclo de projeto da missão.
- Sistematizar o uso do referido Sistema de Teste (STAE) buscando antecipar no ciclo de desenvolvimento a verificação de requisitos por meio de simulação e testes funcionais de software embarcado. Para tanto, a combinação das abordagens MBT e MDE se mostraram de grande valia, quando associadas aos conceitos de dependabilidade. A partir de modelos comportamentais de software comunicante, a abordagem MDE proporciona sistematizar a geração automatizada de códigos computacionais executáveis no STAE e a abordagem MBT permite derivar casos de testes, os quais poderão ser aplicados em diferentes fases do desenvolvimento e integração do nanosatélite.
- Implementar o Sistema de Teste com Arquitetura Escalável (STAE) em quatros Cenários evolutivos (A, B, C, D) de modo que códigos computacionais gerados de forma automatizada possam ser

embarcados em processadores disponibilizados pelo STAE. Nesta configuração, casos de testes gerados a partir de modelos de interoperabilidade serão executados e validados para serem aplicados nas fases subsequentes do processo de V&V, quando os processadores providos pelo STAE forem substituídos pelo hardware real dos subsistemas comunicantes.

- Validar a abordagem de sistematização dos testes e o uso do STAE em um estudo de caso real. O objetivo é analisar a efetividade da abordagem desenvolvida na tese em no mínimo uma fase do processo de V&V, com o uso das ferramentas necessárias e um protótipo do STAE visando verificar aspectos de interoperabilidade e dependabilidade do software embarcado alvo de teste em ambiente integrado com o hardware.

1.3. **Metodologia de desenvolvimento da tese**

A metodologia adotada para o desenvolvimento da abordagem de sistematização de testes com o o uso do STAE compreendeu as seguintes etapas:

- Pesquisa bibliográfica referente às abordagens que vêm sendo utilizadas para identificação e tratamento de falhas em missões espaciais que utilizam Cubesats e software complexos. Algumas abordagens MBT para geração de casos de teste foram encontradas em (GRADES, 2009; MATTIELLO-FRANCISCO, 2009; ALI et al., 2010; MATTIELLO-FRANCISCO, 2012; PELESKA, 2013; PETRENKO; SCHINGLOFF, 2013; PINHEIRO, 2014; SINGH; RAMASAMY, 2015; HERPEL, 2016), entre outros.
- Pesquisa bibliográfica para conhecimento e análise do avanço das técnicas e métodos utilizados no desenvolvimento das abordagens que auxiliam na modelagem de requisitos funcionais de sistemas de software complexos (SOUZA, 2009; MONTECCHI et al., 2011;

PUENTE et al., 2014; FERNÁNDEZ-ISABEL et al., 2015; ABRAHÃO et al., 2017). Estas abordagens buscam representar de forma mais clara o comportamento esperado do software, agilizando a geração de códigos e facilitando a integração de sistemas (PERROTIN; YABAR, 2014). As abordagens tradicionais de desenvolvimento de software não apresentam tanta clareza na representação de software complexo. Importância também foi dada às abordagens que focam aspectos de interoperabilidade e robustez entre os sistemas comunicantes intensivos em software (MATTIELLO-FRANCISCO, 2012; HERPEL et al., 2016; BATISTA et al., 2018). Conforme apresentado em Gundy-Burlet (2013), diversas publicações referentes aos testes de Cubesats destacam a verificação de requisitos de gerenciamento de energia, controle térmico, controle de atitude, porém, há pouco relato sobre testes de interoperabilidade entre subsistemas comunicantes intensivos em software embarcado em Cubesats.

- Nas pesquisas sobre V&V em missões espaciais foi analisada a Norma (ECSS-Q-HB-80-03, 2012), a qual descreve conceitos de dependabilidade e segurança (RABASA, 2015) no desenvolvimento de software de sistemas.
- Após estas pesquisas foram observadas possíveis lacunas existentes entre as abordagens utilizadas para tratamento de falhas em missões envolvendo Cubesats. Também foram observadas abordagens utilizadas para gerar casos de teste e códigos computacionais executáveis buscando melhorar o desenvolvimento e a confiabilidade dos subsistemas comunicantes intensivos em software complexos embarcados nestas missões.
- A taxonomia de dependabilidade de Avizienis et al. (2004) foi analisada para o desenvolvimento de uma planilha de dependabilidade que permite identificar, classificar e tratar possíveis falhas durante a aplicação da sistemática de testes.

- Houve a necessidade de analisar os requisitos funcionais dos subsistemas embarcados em nanosatélites (ISIS, 2011; ISIS, 2013; ALMEIDA, 2016; NOVAL et al., 2016).
- Uma análise foi realizada referente às ferramentas computacionais que auxiliam na modelagem de requisitos e geração de códigos de forma automatizada.
- Outro fator para o desenvolvimento e aplicação da abordagem apresentada foi a análise dos componentes (placas computacionais) passíveis de serem utilizados por um Sistema de Teste com Arquitetura Escalável (STAE), para implementação dos códigos computacionais gerados pelas ferramentas de modelagem.

Após as pesquisas bibliográficas e análises do estado da arte e da prática, a concepção da abordagem sistemática de testes desenvolvida nesta tese foi iniciada. A sistemática de testes utiliza a abordagem MBT para modelar e verificar os requisitos comportamentais de interação entre software comunicante embarcado em um nanosátelite. Baseado nessa modelagem é realizado a transformação de modelos utilizando a abordagem MDE para gerar códigos computacionais de forma automatizada e embarcar esses códigos em placas computacionais programáveis. Esses códigos irão simular o comportamento dos subsistemas sob testes em interação no barramento de comunicação. A planilha de dependabilidade desenvolvida para a abordagem apresentada é utilizada para identificar, classificar e tratar possíveis falhas de interoperabilidade entre software comunicante. Casos de testes de robustez são gerados baseados nas possíveis falhas identificadas nos testes dos subsistemas simulados. Para testar a robustez dos subsistemas reais é utilizado um Mecanismo Emulador de Falha conectado entre os subsistemas reais, o qual injeta falhas no canal de comunicação de modo a provocar situações que demandem ação de robustez por parte dos subsistemas comunicantes.

Para validar a sistematização de teses desenvolvida nessa tese e explorar seu potencial de escalabilidade foi utilizado um estudo de caso real envolvendo

Cubesat. O estudo de caso foi o nanosatélite NanosatC-BR2 que está sendo desenvolvido e testado no laboratório do INPE.

1.4. **Estrutura da tese**

Esta tese está estruturada em sete Capítulos, conforme a descrição a seguir:

- Capítulo 1 – Introdução: é apresentada a contextualização, o objetivo geral, os objetivos específicos, a metodologia de desenvolvimento da tese, o estado da arte da pesquisa realizada e contribuições da tese.
- Capítulo 2 – Fundamentação teórica: artigos técnicos e literatura que guardam relação direta com os temas relacionados à missão espacial de nanosatélites, modelagem de sistemas e realização do processo de V&V de subsistemas intensivos em software, principalmente, nas fases iniciais do projeto.
- Capítulo 3 - Revisão bibliográfica das abordagens e conceitos para tratamento de falhas de sistemas complexos de software: os artigos selecionados estão relacionados com as técnicas utilizadas na abordagem apresentada nesta tese para tratamento de falhas de subsistemas intensivos em software embarcados em Cubesats, com foco em testes de interoperabilidade e análise de dependabilidade.
- Capítulo 4 - Desenvolvimento da abordagem sistemática de testes para tratamento de falhas de sistemas intensivos em software: apresenta a utilização das abordagens MBT e MDE, e os conceitos de interoperabilidade e dependabilidade, conjuntamente, para melhoria no tratamento de falhas nos subsistemas do nanosatélite NanosatC-BR2.
- Capítulo 5 - Sistema de Teste com Arquitetura Escalável (STAE) para apoiar a identificação e o tratamento de falhas de subsistemas comunicantes: apresenta a utilização do STAE nas diferentes fases de desenvolvimento dos subsistemas comunicantes intensivos em

software do NanosatC-BR2 no início do processo de verificação e validação desta missão.

- Capítulo 6 - Aplicação da abordagem desenvolvida: apresenta o desenvolvimento e experimentação da abordagem no tratamento de falhas no estudo de caso NanosatC-BR2.
- Capítulo 7 - Conclusões: apresentação das principais conclusões e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Em pesquisas realizadas entre os anos de 2009 e 2019 foram observados artigos técnicos e literaturas que guardam relação direta com os temas relacionados a missão espacial de nanosatélites, modelagem de sistemas e realização de V&V principalmente nas fases iniciais do projeto.

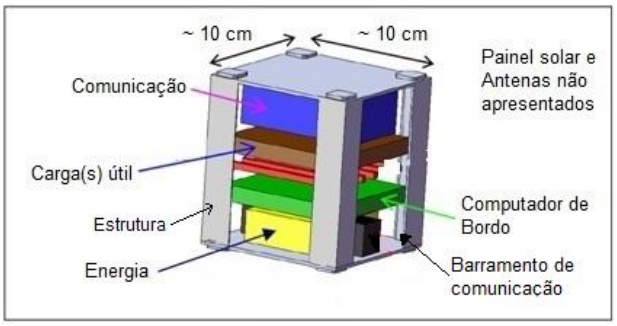
2.1. **Cubesats**

As primeiras propostas de satélites padrão Cubesats surgiram em torno de 15 anos atrás com o objetivo de permitir que estudantes projetassem, desenvolvessem e testassem no espaço satélites com capacidades análogas ao primeiro satélite colocado em órbita. Esses satélites vêm sendo utilizados em ambientes acadêmicos e empresas emergentes no setor buscando a baixo custo e curto prazo de desenvolvimento qualificar em voo tecnologias espaciais inovadoras (HELVAJIAN; SIEGFRIED, 2008).

Conforme as especificações e utilizações de Cubesats, esses satélites são posicionados em órbitas baixas (de 200 a 10.000 km de altitude) com uso de Componentes Comerciais de baixo custo (*Commercial Off-The-Shelf* - COTS), por serem submetidos a baixos níveis de radiação (ASUNDI; FITZ-COZY, 2013). Entretanto, no dia 26 de novembro de 2018 a NASA recebeu informações de Cubesats enviados ao planeta Marte (NASA, 2019), demonstrando que os satélites padrão Cubesats permitem serem utilizados em outras órbitas.

Os Cubesats denominados nanosatélites possuem uma composição básica sendo: estrutura geralmente de alumínio, computador de bordo, carga (s) útil (eis), fornecimento de energia, painel solar, transmissor e receptor de informações entre o satélite e segmento solo, antenas, barramento de comunicação e alguns nanosatélites possuem controle de atitude. A Figura 2.1 apresenta a estrutura de um Cubesat de forma generalizada.

Figura 2.1 - Estrutura de um nanosatélite.



Fonte: Adaptado de Addaim et al. (2010).

A comunicação entre os subsistemas computador de bordo, cargas úteis e outros componentes do nanosatélite é feita por meio do barramento de comunicação e segue um protocolo estabelecido.

O processo de V&V no ciclo de desenvolvimento de satélites é importante para tratamento de falhas na missão espacial. Um dos procedimentos a ser realizado é o V&V em subsistemas intensivos em software que se comunicam nos nanosatélites (MATTIELLO, 2009).

2.2. **Verificação e validação**

A norma (ECSS-S-ST-00-01C, 2012) define verificação como um processo que comprova por meio do fornecimento de evidência objetiva que o produto é projetado e produzido de acordo com suas especificações e validação um processo que comprova que o produto é capaz de realizar seu uso no ambiente operacional pretendido.

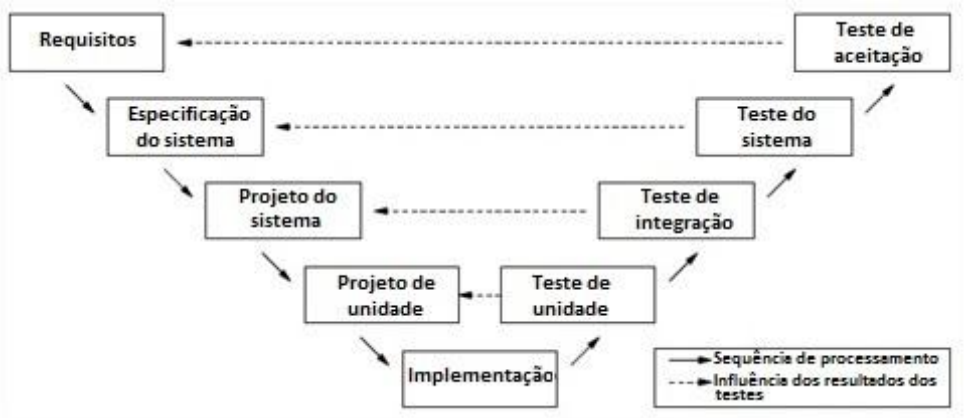
De acordo com Mattiello-Francisco (2003), o uso de componentes existentes no mercado (*COTS*) nas plataformas de satélites somado à padronização dos subsistemas a bordo do satélite permitiu reduzir significativamente o ciclo de desenvolvimento de missões espaciais viabilizando qualificar em voo novas tecnologias espaciais a baixo custo. Entretanto, as atividades de Verificação e

Validação (V&V) nas diferentes fases do ciclo de vida do projeto espacial, que busca melhoria, segurança e confiabilidade das missões, ainda se fazem necessárias e continuam onerosas.

Conforme dissertação de Grades (2009), as diferentes técnicas de V&V existentes abrangem de maneiras diferentes a eficácia de atender aos testes e requisitos necessários para as diferentes fases no desenvolvimento de um sistema não devendo ser menosprezadas principalmente no início do ciclo. As primeiras fases de desenvolvimento devem ter uma atenção especial aos requisitos. Geralmente dois terços do tempo no desenvolvimento de um sistema são empenhados na fase inicial dando atenção em análise, teste, revisão e validação quando apenas um terço dos custos está sendo gasto. Havendo lacunas nas definições dos requisitos e não realização dos testes necessários para validar as etapas do ciclo de desenvolvimento, poderá acarretar em falha de algum componente, falha no sistema, perda do projeto, problema ambientais ou até perda de vidas. Deverá haver preocupação e previsão quanto à necessidade de realizar manutenções em tempo real sem prejudicar o sistema.

Existem muitos modelos diferentes que descrevem como gerenciar o desenvolvimento do sistema. Um representante de destaque é o modelo V. No lado esquerdo do modelo, o desenvolvimento do sistema é mostrado como uma abordagem de cima para baixo: A partir de requisitos, o sistema é especificado, projetado, dividido em unidades, e implementadas. No lado direito, o teste é mostrado com o processo de baixo para cima. Os resultados dos testes têm um impacto sobre as fases iniciais de desenvolvimento. O modelo V é apresentado na Figura 2.2.

Figura 2.2 - Modelo V.



Fonte: Adaptado de Grades (2009).

Segundo Mattiello-Francisco (2009), um dos fatores iniciais e fundamentais para a execução de testes nas abordagens MBT é definir uma arquitetura de teste adequada aos modelos utilizados para gerar os casos de teste. Nesta arquitetura são estabelecidos os pontos de entrada, o tratamento das informações por um Sistema de Teste (ST) e os resultados que serão analisados. Os modelos formais aplicados para a geração de casos de teste na fase de integração de subsistemas intensivos em software visam aumentar a confiabilidade do serviço. Um dos meios para testar a interoperabilidade e robustez entre os subsistemas de satélites é a técnica de injeção de falhas.

Alguns testes a serem realizados durante as atividades de V&V são (SRINIVAS et al., 2014):

- MIL (*Model-in-the-loop*) - Refere-se ao tipo de teste feito para verificar a precisão / aceitabilidade de um modelo ou sistema de controle. Teste de MIL significa que o modelo e seu ambiente são simulados na estrutura de modelagem sem nenhum componente de hardware. A MIL permite testes nos estágios iniciais do ciclo de desenvolvimento.
- SIL (*Software-in-the-loop*) - Refere-se ao tipo de teste feito para validar o comportamento do código do sistema gerado automaticamente e usado em ambiente controlado. O software pode ser embarcado em um ambiente simulado que pode contar com

componentes de hardware real do sistema. O SIL também permite verificar a abrangência do código.

- HIL (*Hardware-in-the-loop*) - Refere-se ao tipo de teste feito para validar o sistema sob teste real que é executado em ambiente simulando ou real. O nível de teste do HIL pode revelar falhas causadas pelo ambiente real.

Segundo Shekoofa et al. (2011), a realização de testes é fundamental no desenvolvimento de satélite. As fases de teste correspondem a 30% ou até 50% do custo total do desenvolvimento do sistema e, portanto, representam um ponto importante para possíveis melhorias. A falta de abordagens sistemáticas e automáticas no processo de verificação e validação levam muitas vezes a se subestimar as atividades necessárias ao processo e sua terceirização para reduzir custos. Assim os testes acabam priorizando a validação dos requisitos funcionais em detrimento dos requisitos de dependabilidade, considerados não-funcionais. Esses fatores afetam a qualidade do produto.

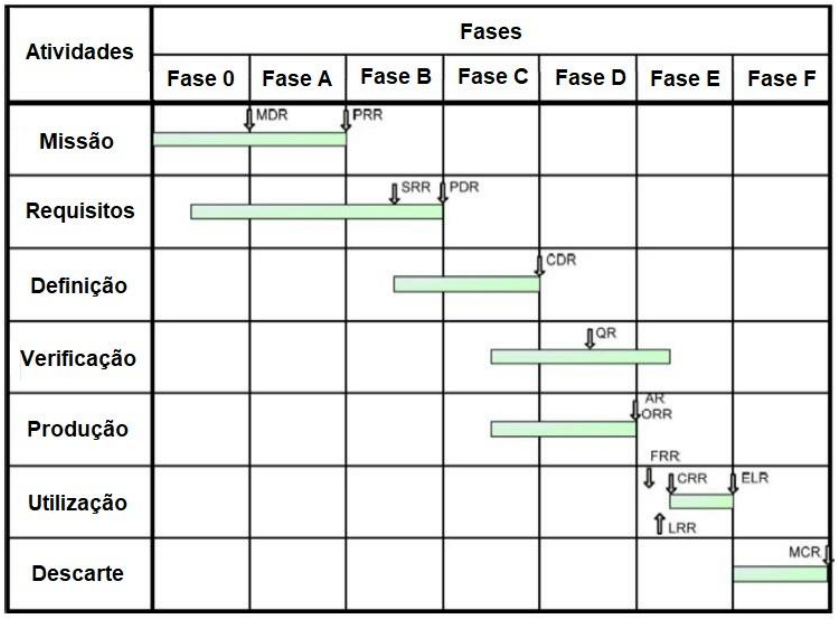
2.2.1. **Ciclo de vida de projetos**

O ciclo de vida de um projeto define as fases que conectam seu início ao seu final. O ciclo de vida proposto pela (ECSS-M-ST-10 Rev. 1, 2009) para projetos espaciais é tipicamente dividido em sete fases, as quais são:

- Fase 0: Análise de missão / Identificação de necessidades.
- Fase A: Viabilidade.
- Fase B: Definição Preliminar.
- Fase C: Definição Detalhada.
- Fase D: Qualificação e Produção.
- Fase E: Operações.
- Fase F: Descarte.

As fases do projeto estão intimamente ligadas às atividades no nível do sistema e do produto. Dependendo das circunstâncias específicas de um projeto e da excitação de risco envolvido, as atividades podem se sobrepor às fases do projeto. A Figura 2.3 apresenta as fases do projeto representadas nas colunas, as atividades representadas nas linhas e as revisões de projeto destacadas nas células por setas. A cada revisão de projeto é associado um conjunto de entregas que podem ser documentos e/ou produtos.

Figura 2.3 – Ciclo de vida de projetos (ECSS).



Fonte: Adaptado de ECSS (2009).

As fases 0, A e B são focadas principalmente em:

- Elaboração de requisitos funcionais e técnicos do sistema e identificação dos conceitos do sistema para cumprir as necessidades da missão, observando os problemas técnicos e programáticos identificados pelo desenvolvedor do projeto e pelo cliente.
- Identificação de todas as atividades e recursos a serem usados para desenvolver segmentos espaciais e terrestres da missão espacial.

- Avaliações iniciais do risco técnico e programático.
- Início de atividades de pré-desenvolvimento.

As Fases C e D compreendem todas as atividades a serem realizadas para desenvolver e qualificar os segmentos espacial e terrestre e seus produtos.

A Fase E compreende todas as atividades a serem executadas para lançamento, comissionamento, utilizar e manter os elementos orbitais do segmento espacial e utilizar e manter o segmento terrestre associado.

A Fase F compreende todas as atividades a serem realizadas para eliminar com segurança o produto lançado no espaço, bem como desativa a infraestrutura do segmento terrestre alocada à operação da missão.

A revisões de projeto recomendadas na ECSS são:

- MDR: *Mission Definition Review* (Revisão de Definição de Missão).
- PRR: *Preliminar Requirements Review* (Revisão de Requisitos Preliminares).
- SRR: *System Requirements Review* (Revisão de Requisitos de Sistema).
- PDR: *Preliminary Design Review* (Revisão de Projeto Preliminar).
- CDR: *Critical Design Review* (Revisão de Projeto Crítico).
- QR: *Qualification Review* (Revisão de Qualificação).
- AR: *Acceptace Review* (Revisão de Aceitação).
- ORR: *Operational Readiness Review* (Revisão de Maturidade Operacional).
- FRR: *Flight Readiness Review* (Revisão de Maturidade de Voo).
- LRR: *Launch Readiness Review* (Revisão de Maturidade de Lançamento).
- CRR: *Comissioning Result Review* (Revisão de Resultado de Comissionamento).

- ELR: *End-of-Life Review* (Revisão de Final de Vida).
- MCR: *Mission Close-out Review* (Revisão de Fechamento de Missão).

2.3. **Dependabilidade e segurança de software**

Segundo Avizienis et al. (2004), a realização de testes é um dos meios primários para garantir a dependabilidade (confiança no funcionamento) e a segurança de um sistema cujo objetivo é prover serviços que, justificadamente, necessitam ser confiáveis. A dependabilidade de uma missão espacial deve zelar para que a operação um sistema não afete o desempenho de outro sistema ambos parte da mesma missão. A dependabilidade de uma missão busca evitar que as falhas de serviço sejam mais frequentes e mais graves do que o aceitável e abrangem a verificação dos requisitos de confiabilidade, disponibilidade, segurança, integridade e capacidade de manutenção. A segurança abrange preocupações com confidencialidade, disponibilidade e integridade.

Os casos de ameaças à dependabilidade e segurança que necessitam ser observados são:

a) Execução incorreta não intencional ou imprevista ocasionando falhas.

b) Configuração errônea de software gerando erros que acarretaram falhas no sistema.

c) Incapacidade de realizar uma função ocasionando defeitos no sistema. A utilização de um mecanismo de tolerância a falhas pode impedir a propagação de erros. Medidas necessitam ser realizadas observando e identificando onde as precauções precisam ser aplicadas, considerando as técnicas existentes para prevenção de falhas, tolerância a falhas, remoção de falha e previsão de falha.

A manutenção em longo prazo inclui não apenas reparos, mas também todas as modificações do sistema que ocorrem durante a fase de utilização de um

sistema, envolvendo reparos para correção e/ou prevenção e/ou modificações para adaptação e/ou acréscimos (AVIZIENIS et al., 2004).

A Norma (ECSS-Q-HB-80-03, 2012), preconiza que a quantidade de testes realizados em um componente do sistema deve estar relacionada com a sua criticidade. A classificação da criticidade do sistema é principalmente destinada a identificar quais os componentes são críticos podendo falhar de modo a causar ou contribuir para falhas críticas, permitindo assim que haja uma concentração de esforço no projeto sobre os componentes críticos. Esses componentes são submetidos a atividades de engenharia e garantia de produto de forma mais exigente, pois se for aplicado em todo o sistema o seu desenvolvimento ficará oneroso, em termos de orçamento, mão de obra e tempo. Componentes menos críticos significam menos esforço despendido em medidas de mitigação de risco. Diversos testes podem ser utilizados para melhorar a dependabilidade e a segurança do software, por exemplo, teste de robustez, o qual é utilizado para apresentar a capacidade do software de funcionar corretamente na presença de entradas inválidas ou condições ambientais estressantes.

As ações de dependabilidade são incorporadas na taxonomia de falhas buscando medidas unificadas de dependabilidade e segurança. A utilização da taxonomia (prática e a ciência da classificação de coisas ou conceitos, incluindo os princípios que fundamentam tal classificação) leva em consideração a interação do sistema com outros sistemas, com o computador, aplicativos computacionais, seres humanos e o meio físico com seus fenômenos naturais. O comportamento de um sistema é descrito por um conjunto de uma sequência de estados sendo estes: computação, comunicação, informação armazenada, interconexão, e condição física.

2.3.1. **Árvore de dependabilidade**

A fase de utilização de um sistema, denotada por ciclo de vida do sistema, inicia quando esse é aceito para uso e começa o fornecimento dos seus serviços para os usuários, interagindo com o ambiente real de operação e

podendo ser afetado por falhas de natureza diversa. Os ambientes de uso podem ser constituídos pelos seguintes elementos: mundo físico, administradores, usuários, prestadores de serviços, intrusos podendo ser humano ou sistema, ferramentas de desenvolvimento e infraestrutura (AVIZIENIS et al., 2004).

Tendo como alvo analisar os atributos de dependabilidade, as categorias de falhas, os ambientes e manutenção do sistema, uma taxonomia poderá ser utilizada para avaliar as falhas que podem afetar um sistema durante sua vida. Essa taxonomia pode ser classificada, segundo (AVIZIENIS et al., 2004), em oito classes de falhas, que consideram classes elementares de falha, sendo cada classe dividida em duas subclasses de falhas, conforme definido a seguir:

1. Fase da criação ou ocorrência: I) Falhas de Desenvolvimento, II) Falhas Operacional.
2. Limites do sistema: III) Falhas Interna, IV) Falhas Externa.
3. Causa fenomenológico: V) Falhas Evento Natural, VI) Falhas Humana.
4. Dimensão: VII) Falhas de Hardware, VIII) Falhas de Software.
5. Objetivo: IX) Falhas Não Maliciosa, X) Falhas Maliciosa.
6. Intenção: XI) Falhas Não Intencional, XII) Falhas Intencional.
7. Capacidade: XIII) Falhas Acidental, XIV) Falhas por Incompetência.
8. Persistência: XV) Falhas Permanente, XVI) Falhas Transitória.

Nem todas as combinações dessas classes estão relacionadas. Um efeito natural não está relacionado com capacidade, assim como alguns outros relacionamentos. Realizando certas exclusões podem ser identificadas trinta e uma (31) combinações de falhas. Essas combinações podem ser representadas e agrupadas em nove (9) tipos de falhas que em alguns casos se sobrepõem parcialmente. Esses nove tipos de falhas podem ser representados por: A) Falhas de Software [combinações 1a 4], B) Bombas de Lógica [combinações 5 e 6], C) Erro de Hardware [combinações 7 a 10], D)

Defeitos de Produção [combinações 7 a 11], E) Deterioração Física [combinações 12 e 13], F) Interferência Física [combinações 14 a 21], G) Tentativas de Intrusão [combinações 22 a 24], H) Vírus e *Worms* [combinação 25] e I) Erros de Entrada [combinações 26 a 31].

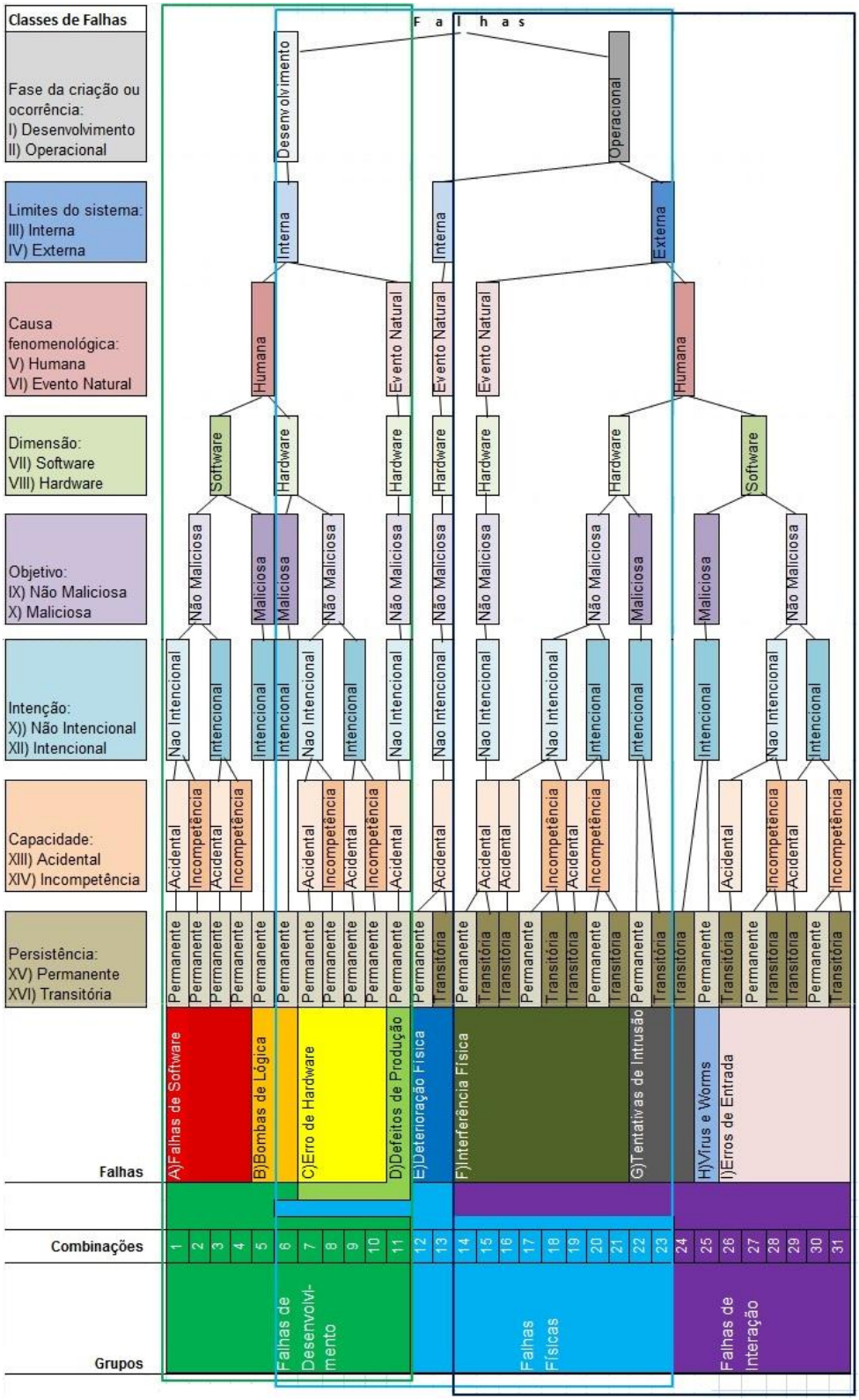
Desta forma as classes podem ser divididas e representadas em três (3) grandes grupos que se sobrepõem parcialmente: falhas de desenvolvimento [combinações 1 a 11], falhas físicas [combinações 6 a 23] e falhas de interação [combinações 14 a 31].

Baseado nas informações relacionadas à dependabilidade e segurança será apresentado na Figura 2.4 uma árvore de dependabilidade identificando cada item de falha definido anteriormente com seus respectivos relacionamentos.

As classes de falhas estão representadas em diferentes níveis do topo para baixo.

Na base da árvore estão os três (3) grandes grupos de falhas que contém os nove (9) tipos de falhas identificados de A a I. As combinações de falhas estão numeradas de 1 a 31.

Figura 2.4 - Árvore de dependabilidade.



Fonte: Adaptado de Avizienis et al. (2004).

Observando as ramificações da árvore, a ocorrência de uma falha em determinada localização da árvore poderá ocasionar um ou mais tipos de falhas dependendo do seu relacionamento com a fase anterior ou subsequente. Analisando uma das ramificações da árvore pode ser observado que uma falha física, por rastreabilidade, pode ocasionar uma falha de software se não houver remoção desse tipo de falha de dependabilidade. Exemplificando: se durante a comunicação entre dois subsistemas houver uma falha física e esta falha não estiver prevista na configuração do software, os sistemas poderão ficar travados aguardando o retorno da comunicação.

2.4. **Injeção de falhas**

Conforme delineado por Hsueh et al. (1997), a injeção de falhas é importante para avaliar a dependabilidade de sistemas computacionais. Pesquisadores e engenheiros desenvolveram novos métodos para injetar falhas. As falhas podem ser implementadas em hardware e/ou software dependendo da necessidade de análise a ser realizada. As técnicas de injeção de falhas por software acabam geralmente sendo mais atraentes por não necessitarem de hardwares caros, porém dependendo da necessidade dos testes necessitam ser por hardware (BATISTA et al., 2018).

Conforme definido em Avizienis et al. (2004), a avaliação de dependabilidade envolve o estudo de falhas (*faults*), erros (*errors*) e defeitos (*failures*). Uma falha (*fault*) está ativa quando produz um erro (*error*), caso contrário está inativa. Uma falha ativa pode ser interna, que estava anteriormente inativa e foi acionada pelo processo de computação ou condições ambientais, ou pode ser externa. A propagação de erro dentro de um componente pode ocasionar uma propagação externa e provocar um defeito de serviço (*failure*).

As causas das falhas devido a erros é um fator difícil de ser identificado no ambiente operacional. É particularmente difícil recriar um cenário de falha de um sistema grande e complexo. Para identificar e compreender as falhas potenciais de um sistema, uma abordagem baseada em experimento pode ser utilizada para analisar a dependabilidade de um sistema. Tal abordagem pode

ser aplicada não apenas durante as fases de concepção e desenvolvimento, mas também durante o protótipo e fases operacional (HSUEH, 1997).

A utilização de uma abordagem baseada em experimento requer que primeiramente possam ser compreendidos a arquitetura, estrutura e comportamento do sistema. Especificamente, precisam ser conhecidos a sua tolerância a falhas e possíveis erros. Para isso, são utilizadas ferramentas para injetar falhas e erros específicos, bem como mecanismos para detecção, recuperação e monitoramento de seus efeitos.

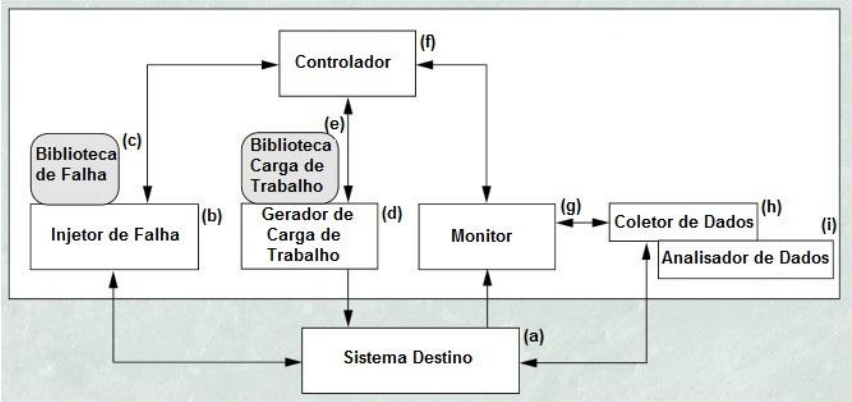
Engenheiros na maioria das vezes usam ferramentas de baixo custo (*low-cost*) para injeção de falhas baseado em simulação buscando avaliar a dependabilidade de um sistema que está nas fases de concepção e desenvolvimento. A injeção de falhas baseada em simulações busca ocasionar erros ou falhas de acordo com o arranjo predeterminado, sendo útil para avaliar a eficácia dos mecanismos de tolerância a falhas e dependabilidade do sistema. O teste de um protótipo permite que o sistema seja avaliado sem quaisquer suposições sobre o projeto do sistema (HSUEH, 1997).

Algumas injeções de falhas podem ser realizadas em tempo de compilação modificando as instruções do código do programa antes da sua execução. Esse método injeta erros no código de origem ou destino para emular o efeito no hardware, software e falhas transitórias. Os mecanismos para injetar falhas em tempo de execução podem ser usados para provocar os seguintes erros: a) tempo esgotado, predeterminando a expiração de um tempo de execução, b) exceções e armadilhas, inserindo no software falhas que ocasionam certos eventos e condições de erros ou injetando uma falha antes da execução de uma determinada instrução, e c) inserção de código, adicionando instruções de falhas ao invés de alterar as instruções originais.

Um ambiente para um sistema de injeção de falhas pode ser representado conforme a Figura 2.5: O injetor de falta (b) injeta falhas no sistema destino (a) enquanto executa comandos do gerador de carga de trabalho (d). O monitor (g) rastreia a execução dos comandos e inicia a coleta de dados sempre que necessário. O coletor de dados (h) realiza a coleta de dados *on-line* e o

analisador de dados (i) executa o processamento e a análise dos dados. O controlador (f) controla a experimento. Fisicamente, o controlador é um programa que pode ser executado no sistema destino ou em um computador separado. O injetor de falha pode ser hardware ou software customizado. O próprio injetor de falha pode suportar diferentes tipos de falhas, localizações de falhas, tempos de falha e semântica de hardware ou estrutura de software apropriados - cujos valores são extraídos de uma biblioteca de falhas (c). A biblioteca de falhas é um componente separado, o que permite uma maior flexibilidade e portabilidade. O gerador de carga de trabalho (d), monitor (g) e outros componentes podem ser implementados da mesma maneira.

Figura 2.5 - Sistema de Injeção de Falhas.



Fonte: Adaptado de Hsueh (1997).

A análise das medições usando dados reais contendo falhas e erros devem ser armazenados durante um longo período, pois essas ocorrências podem ocorrer com pouca frequência, sendo um dos fatores as condições ambientais.

Um dos meios para injetar falhas no sistema é por meio do barramento de comunicação entre os subsistemas do satélite, o qual segue um protocolo (BATISTA et al., 2018).

2.5. **Análise dos modos de falha e seus efeitos**

A Norma (ECSS-Q-HB-80-03, 2012), apresenta que a contribuição na utilização de técnicas de análise de sistema baseado em modelos voltados a dependabilidade e segurança é um fator importante, especialmente tendo em vista a crescente complexidade dos componentes de software utilizados na área espacial em situações críticas, juntamente com as crescentes restrições de custo e cronograma. Produtos de hardware e software são classificados de acordo com sua criticidade, a fim de concentrar as atividades de engenharia e garantia de produto sobre os itens mais críticos.

Assim, conhecer a complexidade da aplicação do software é importante para se adotar métodos e técnicas específicas que implementam verificações sistemáticas de falhas. A previsão de falhas visa estimar o número atual, a incidência futura e as prováveis consequências de falha. Essa estimativa pode ser qualitativa, por exemplo, por meio de técnicas como a Análise dos Modos de Falha e seus Efeitos (*Failure Mode and Effect Analysis* - FMEA) e Análise de Árvore de Falhas (*Fault Tree Analysis* - FTA), e quantitativa, por exemplo, modelos estatísticos.

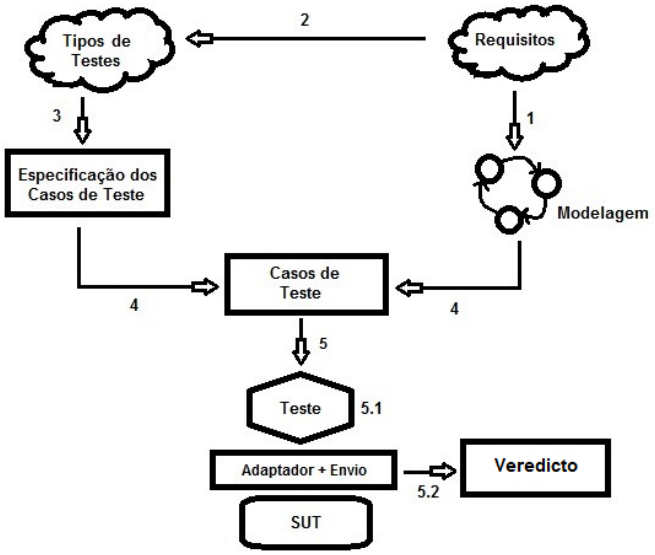
Uma tabela FMEA poderá conter informações que irão permitir relacionar possíveis origens das causas de falhas, quais os efeitos que as falhas poderão ocasionar, que mecanismos poderão ser realizados para evitar o problema, pontuarem conforme o grau do problema e que mitigações poderão ser realizadas para evitar ou prevenir o defeito no sistema (ECSS-Q-HB-80-03, 2012).

Identificadas as possíveis falhas e mitigações a serem realizadas, poderão ser modelados em um software os modelos de falhas com as mitigações (SW-FMEA). Quando aplicados no sistema seus efeitos permitem analisar aspectos de robustez requeridos do sistema. Esse teste será comparado com os resultados do sistema em estado nominal e analisado a satisfação dos resultados. Os testes poderão ser utilizados em várias fases de verificação no desenvolvimento do sistema ou em outros sistemas similares (MONTECCHI, 2016).

2.6. **Teste baseado em modelo**

Segundo Utting et al. (2006), novas abordagens vêm sendo aplicadas devido à necessidade de detectar e antecipar possíveis falhas que possam ocorrer no desenvolvimento de sistemas de software cada vez mais complexos. Uma dessas abordagens é o Teste Baseado em Modelo (*Model-Based Testing* - MBT). A abordagem MBT propõe o uso de modelos para realizar a geração automática de casos de teste com a finalidade de avaliar o desempenho de um sistema, possibilitando a detecção de possíveis falhas. Esses testes propõem o uso de modelos explícitos em notação formal para especificar o comportamento esperado do Sistema sob Teste (*System Under Test* - SUT). Os modelos devem contemplar as propriedades relevantes do sistema que se deseja testar. Modelos comunicantes são usados para testes de interoperabilidade visando auxiliar na verificação se duas ou mais implementações interagem corretamente fornecendo os serviços especificados. A abordagem MBT permite a automatização dos testes a serem realizados antes da fase de integração do projeto. A utilização dos benefícios da MBT também requer uma arquitetura extensível e configurável que a torne adaptável a vários contextos de aplicação (AVIZIENIS et al., 2004). A Figura 2.6 apresenta um processo de Teste baseado em modelo. Neste processo, nas etapas 1 e 2 os requisitos são analisados sob a ótica de modelagem comportamental do sistema e sob a ótica de testes, respectivamente. Na etapa 3 serão especificados os casos de teste que deverão ser realizados para validar o sistema. Na etapa 4, com a modelagem e as especificações, serão gerados os casos de teste. Na etapa 5 os casos de teste gerados são executados no Sistema Sob Teste (SUT). Em 5.1 serão executados os casos de teste em diferentes níveis de abstração, por meio de um adaptador. Em 5.2 os resultados dos testes irão para veredicto, por meio do adaptador.

Figura 2.6 - Processo de Teste Baseado em Modelo.



Fonte: Adaptado de Utting et al. (2006).

Segundo Mattiello-Francisco et al. (2012), outra necessidade é verificar se as implementações interoperam na presença de falhas utilizando testes de robustez para determinar o grau em que as implementações podem funcionar corretamente em presença de entradas inválidas ou sob condições ambientais adversas. A utilização dos Testes Baseados em Modelos auxilia nas fases de V&V em termos de especificação e geração dos casos de teste mais adequados à verificação dos requisitos, do sistema, identificando os componentes que podem ocasionar falhas e buscando reduzindo esforços de testes.

A abordagem MBT fornece uma técnica para a geração automática de casos de teste usando modelos extraídos de artefatos de software conforme descrito em Dias et al. (2007). Um modelo formal seria, por exemplo, Máquina de Estado Finita (*Finite State Machine* - FSM) ou a Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML), as quais descrevem o comportamento do software ou do sistema utilizando diagramas. A automação utilizando MBT depende de três elementos fundamentais: a) o modelo utilizado para a descrição do comportamento do software, b) o algoritmo de geração de teste, e c) ferramentas que auxiliam e geram informações, como casos de teste

entre outras informações, que possibilitem dar apoio à execução dos testes. Diferentes níveis de testes requerem diferentes estratégias de geração de casos de teste. A possibilidade de reutilizar ou extrair um modelo de teste do modelo de comportamento do software melhora a produtividade devido à economia de tempo.

Utilizar ferramentas proprietárias de apoio MBT ou ferramentas de integração pode ser proveitoso em recursos, mas as taxas de licenciamento limitam o seu uso em alguma organização. Alguns requisitos para redução e aumento da automação podem ser: a) usar uma linguagem simples e conhecida, como UML, para a modelagem de software, b) usar as ferramentas de testes integradas com o processo de desenvolvimento de software, c) automatizar o máximo de etapas possíveis em uma abordagem MBT.

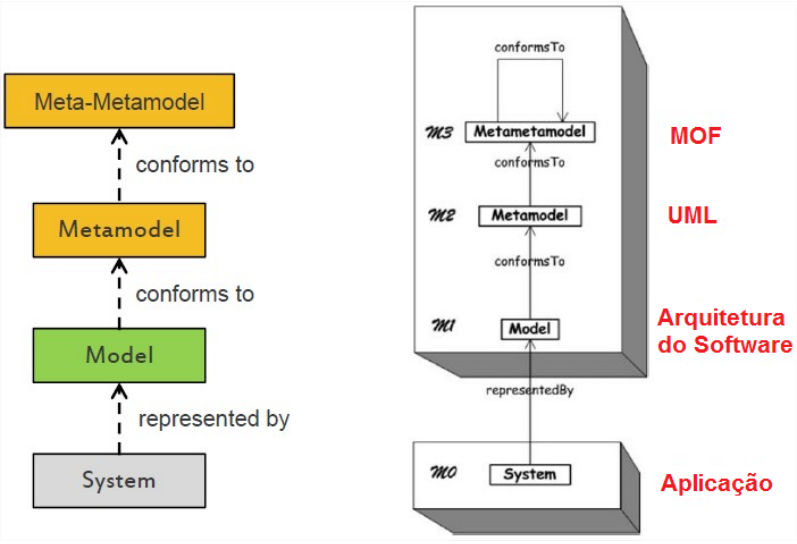
Uma modelagem bem elaborada, capaz de derivar de modelos abstratos especificações detalhadas, irá gerar casos de teste abrangentes permitindo uma validação coerente com as definições dos requisitos. As restrições devem estar representadas na modelagem, pois são elas que permitem identificar falhas no sistema real. Gerar os casos de teste manualmente para sistemas de software complexos requer horas de desenvolvimento e podem ocorrer falhas nas definições. Existem diversos recursos computacionais que permitem a geração de casos de teste de forma automatizada economizando tempo de desenvolvimento. Quanto melhor os detalhes dos requisitos estiverem representados na modelagem, maior será a efetividade dos casos de teste gerados. Outra vantagem da geração de casos de teste de forma automatizada é a possibilidade de haver alterações nos requisitos de forma mais dinâmica do que a manual, requerendo menos tempo de desenvolvimento (ALI et al., 2010).

2.7. **Engenharia dirigida por modelos**

Conforme apresentado por Montecchi (2011), a abordagem da Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE) vem vencendo o desafio do uso continuado de modelos nas diferentes fases do ciclo da missão, com o propósito de apoiar o processo de V&V de sistemas complexos. Esta

abordagem busca transformar de forma automática códigos de uma determinada linguagem de modelagem em uma linguagem de programação computacional executável. Para isso é preciso entender e definir a estrutura, os relacionamentos e restrições das entidades de um domínio específico, transformando essas informações em modelagens denominadas meta-modelos. A identificação dos meta-modelos separa as preocupações facilitando configurações e expansões (SCHIMIDT, 2006). Uma sequência resumida de passos a serem realizados seria analisar o sistema, representar em uma modelagem e transformar em meta-modelo. Os meta-modelos darão uma visão e definição mais detalhada da representação do sistema. Se houver necessidade de abstração das informações em mais baixo nível os meta-modelos poderão ser detalhados em meta-metamodelos. Esse nível de abstração permite testar níveis isolados do projeto antes de haver integração como um todo. Uma ferramenta computacional existente para definir os meta-metamodelos é a linguagem denominada Serviço de Meta-Objeto (*Meta-Object Facility* - MOF). A Figura 2.7 apresenta níveis de meta-modelos.

Figura 2.7 - Níveis de meta-modelos.



Fonte: Montecchi (2016).

Esse diagrama apresenta as abstrações do sistema para níveis mais detalhados até o *meta-metamodel*. Primeiramente a aplicação do sistema (M0) é analisado e sua arquitetura é modelada (M1). O próximo passo é modelar os *metamodel* em uma linguagem, no caso representado com a UML (M2). Posteriormente os *metamodel* serão abstraídos em mais um nível, *meta-metamodel* (M3), sendo representada a utilização do MOF.

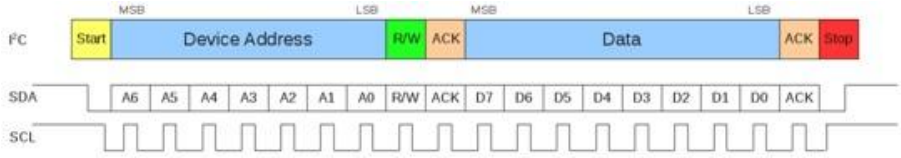
2.8. **Protocolo de comunicação I2C**

O protocolo I2C é altamente utilizado para a comunicação entre subsistemas no padrão Cubesat. O protocolo de comunicação possui os condutores elétricos SCL e SDA cuja finalidade é realizar troca de informações no barramento de comunicação. SCL é a linha de relógio, sendo usado para sincronizar todas as transferências de dados sobre o barramento. SDA é a linha de dados. As linhas SCL e SDA são conectadas aos dispositivos do barramento. Existe um terceiro fio sendo o Terra (*Ground*) e um quarto podendo ter 5 ou 3.3 volt de alimentação que deverá ser distribuído para os dispositivos.

O pacote I2C de transmissão de informações começa com a identificação de inicio de transmissão (*Start*) seguido pelo endereço do dispositivo que irá receber ou fornecer o pacote (*Device Address*). O endereço possui sete bits (A6, A5, A4, A3, A2, A1, A0). O bit mais significativo a ser transmitido primeiro está à esquerda (*Most Significant Bit* - MSB) e o menos significativo à direita (*Less Significant Bit* - LSB). A próxima identificação representa se o pacote será lido do dispositivo ou escrito no dispositivo (*R/W*). Na sequência vem a indicação se o pacote foi recebido e reconhecido (*ACK*) ou se o pacote não foi recebido e/ou não reconhecido (*NACK*). No caso do pacote ter sido recebido, um byte de informação será transmitido no barramento *(Data*: D7, D6, D5, D4, D3, D2, D1, D0), seguido pela identificação de recebido (*ACK*) ou não recebido (*NACK*). Será enviado um byte por vez seguindo pela identificação de *ACK* ou *NACK*. Após o final da transmissão haverá a identificação de fim e parada de

transmissão (*Stop*) (PHILIPS, 2003). O formato do protocolo I2C é apresentado na Figura 2.8.

Figura 2.8 - Formato do protocolo I2C.



Fonte: Philips (2003).

Existem analisadores lógicos de protocolo I2C que permitem analisar o tráfego de informações nesse formato de protocolo. Essa análise permite verificar se as informações estão trafegando conforme estabelecido ou se está havendo alguma falha de comunicação (PHILIPS, 2003).

3 REVISÃO BIBLIOGRÁFICA DAS ABORDAGENS PARA TRATAMENTO DE FALHAS EM SOFTWARE COMPLEXO

A engenharia de software tem evoluído necessitando atender aos avanços tecnológicos exigidos na área computacional. Na década de 1980 surgiram as ferramentas de Engenharia de Software Assistida por Computador (*Computer-Aided Software Engineering* - CASE). Essas ferramentas surgiram para auxiliar as atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes. Os desenvolvedores representavam os projetos como máquinas de estado, diagramas de estrutura e fluxo, entre outras representações gráficas. Essas formas auxiliavam reduzir esforços de codificação manual, depuração e portabilidade de programas. Porém, as ferramentas CASE não foram amplamente utilizadas devido à linguagem gráfica não ser compatível com todas as plataformas de sistema operacional.

Os avanços nas linguagens de programação e plataformas durante as últimas duas décadas elevaram o nível de abstrações de software disponíveis para os desenvolvedores. Novas plataformas de desenvolvimento e bibliotecas reutilizáveis minimizaram os esforços computacionais auxiliando nas notificações de eventos, análise de tolerância a falhas, segurança e gerenciamento de recursos distribuídos. A evolução das linguagens de terceira geração e plataformas reutilizáveis auxiliara no desenvolvimento de software complexo. Entretanto, as plataformas evoluíram mais rapidamente do que a capacidade das linguagens de terceira geração, havendo necessidade de transferir manualmente os códigos dos aplicativos para diferentes plataformas ou versões mais recentes da mesma plataforma. Esse esforço demanda tempo, especialmente para realizar as atividades relacionadas à integração, como implantação do sistema, configuração e garantia de qualidade.

Essas evoluções dificultaram os desenvolvedores terem uma ampla visão das alterações de requisitos. A falta de uma visão integrada juntamente com o perigo de efeitos colaterais imprevistos muitas vezes obriga os desenvolvedores a implementar soluções de duplicidade de códigos,

comprometendo os princípios arquitetônicos de sistemas e garantia da qualidade (SCHMIT, 2006).

Em decorrência das necessidades no desenvolvimento de sistemas complexos novas abordagens de modelagem surgiram buscando facilitar e agilizar o desenvolvimento de sistemas integrados o que possibilitou uma visão mais clara e ampla. A modelagem de sistemas passou a ser utilizada na indústria espacial no início dos anos 2000 (PERROTIN; YABAR, 2014).

Preocupações referentes à confiabilidade dos componentes que integram um sistema vem de longa data. A origem do conceito de dependabilidade data da primeira geração de computadores eletrônicos, entre os anos 40 e 50, onde técnicas práticas foram empregadas para melhorar a confiabilidade dos componentes. O aumento do interesse na tolerância a falhas e na confiabilidade dos sistemas durante os anos de 1960 e 1970 levou ao estudo da dependabilidade de forma mais abrangente, não apenas focado em confiabilidade, mas também de outros atributos de funcionamento do sistema alvo. Em 1980, Jean-Claude Laprie começou a usar a dependabilidade como um conceito para estudar tolerância a falhas e confiabilidade de sistemas (RABASA, 2015). O trabalho foi sintetizado pelo mesmo autor em 1992 no livro Dependabilidade: Conceitos Básicos e Terminologia de Rabasa. As principais inovações foram a adição de segurança como um atributo e da classe de falhas intencionais maliciosas na taxonomia das falhas. Muitos conceitos foram refinados e elaborados. O próximo passo importante foi o reconhecimento da segurança como uma combinação dos atributos de confidencialidade, integridade e disponibilidade e a adição da classe de falhas intencionais não maliciosas, juntamente com uma análise dos problemas de especificações inadequadas do sistema, embora forneça apenas uma classificação resumida de ameaças de dependabilidade (AVIZIENIS et al., 2004).

3.1. **Abordagens e métodos de teste baseados em modelos**

Com o avanço tecnológico espacial houve uma evolução nas abordagens utilizadas buscando tratamento de falhas em missões espaciais. Nesta tese

foram selecionados alguns trabalhos relacionados com as abordagens pertinentes a MBT, MDE e ao processo de V&V em subsistemas intensivos de software embarcados em Cubesats e sistemas complexos buscando tratamento de falhas. A seguir são apresentados alguns destes trabalhos visando destacar apenas as abordagens utilizadas com os mesmos objetivos da abordagem apresentada nesta tese e realizar comparações.

Segundo Herpel et al. (2016), a engenharia de software vem enfrentando nas últimas décadas o aumento da complexidade das aplicações computacionais pela necessidade de autonomia e robustez em aplicações mais sofisticadas. A complexidade dificulta a clareza da apresentação e comprovação que um projeto poderá atingir o objetivo como esperado. Uma das etapas para atender os objetivos é analisar as atividades de desenvolvimento e testes de um projeto do software. A abordagem baseada em modelos vem sendo desenvolvida com objetivo de cada vez mais existir meios para facilitar e melhorar o desenvolvimento, apresentação e validação de um projeto de software.

Os testes são atividades essenciais para a validação de um sistema, porém testes complexos demandam tempo e orçamento. Gerar um conjunto de testes curto e eficaz geralmente requer trabalho manual e conhecimento especializado. O desenvolvimento de software tem evoluído para abordagens baseadas em modelos com o uso extensivo de geradores de códigos de forma automatizada. A abordagem de Teste Baseado em Modelo (*Model Based Testing* - MBT) permite gerar casos de teste a partir do modelo comportamental do sistema. Na ocorrência de mudanças nos requisitos a utilização dessa modelagem possibilita gerar novos casos de teste de forma mais dinâmica comparada na geração manual. A Linguagem Unificada de Modelagem (*Unified Modeling Language* - UML) possibilita a geração de casos de teste baseados nos modelos. Outra ferramenta que auxilia na MBT é a Especificação Baseada em Sequência (*Sequence Based Specification* - SBS).

A indústria espacial adota o modelo V para desenvolver sistemas intensivos em software. Esse modelo visa desenvolver com qualidade um software complexo e a utilização de testes para comprovar conformidade do software com os

requisitos especificados. A MBT é uma abordagem promissora para controlar o esforço gasto em verificação e validação (V&V), sendo um processo utilizado para aplicações embarcadas em missões espaciais. A utilização dessa abordagem abrange rever os requisitos cuidadosamente e realizar casos de teste verificando se os requisitos estão sendo atendidos.

O artigo de Grades (2009) apresenta abordagens para combinar critérios de cobertura, usar transformações de modelo para teste e combinar máquinas de estado com outros modelos de teste, utilizando a abordagem MBT. Também é apresentado um algoritmo de geração de teste que permite combinar critérios de cobertura baseados em fluxo de controle, fluxo de dados ou transição com critérios de cobertura baseados em limites. O estudo de caso apresentado foi em um caixa eletrônico ATM (*Automated Teller Machine*).

Segundo Ali et al. (2010), nos últimos anos a abordagem Teste Baseado em Modelo tem atraído um interesse cada vez maior da indústria e da academia estudantil. Essa abordagem, no entanto, requer ferramentas que não apenas automatizem o processo de teste, mas que também dependam de uma arquitetura extensível e configurável tornando-as adaptáveis a vários conjuntos de aplicações.

O artigo de Ali at el. (2010) apresenta uma abordagem para projetar e desenvolver ferramentas de MBT baseadas na tecnologia de transformação de modelos (*TRansformation-based tools for Uml-baSed Testing* - TRUST) baseado em máquinas de estado modeladas em UML. O autor relata que realizar a modelagem de um sistema não é uma tarefa trivial e requer que a especificação seja cuidadosamente estudada. Mesmo que construções como simultaneidade e hierarquia permitam facilitar a capacidade de compreensão de grandes máquinas de estado, tais construções podem confundir o desenvolvedor. A simultaneidade se não for cuidadosamente aplicada, permite introduzir erros de modelagem na prática. Foi realizado um estudo de caso em um sistema de conferência de multimídia e controle de segurança.

No artigo apresentado em Peleska (2013), o Teste Baseado em Modelo é considerado uma tecnologia de ponta na indústria. Essa abordagem é

atualmente aplicada, com ênfase especial nos domínios aviônico, ferroviário e automotivo. Uma das facilidades da MBT é a geração de casos de teste de forma automatizada do sistema sob teste (SUT). O gerador de casos de teste MBT utiliza algoritmo e fórmulas lógicas que identificam cálculos nos modelos para determinar os casos de teste. Os engenheiros de teste são deslocados da programação de procedimentos para a modelagem do sistema. Esse procedimento resulta em um retorno de investimento, pois os testes são gerados de forma automatizada facilitando a observação dos resultados. Os processos de V&V também são facilitados porque os casos de teste refletem os requisitos dos sistemas e podem ser automaticamente rastreados. Nesse artigo foram relatadas as seguintes observações: i) o comportamento do sistema deve ser estruturado em vários modos operacionais de níveis superiores, ii) os modos de falhas não são determinísticos, mas podem ser iniciados de forma determinística para fins de teste por meio de injeção de falhas pré-planejadas, iii) pode ser aplicado restrições durante os testes incrementais, e iv) o comportamento não é completamente determinístico, mas podendo ser capturado por conjunto de restrições. Nesse artigo é apresenta a ferramenta RT_Tester2, a qual suporta as linguagens de modelagem UML e SysML em um estudo de caso industrial. Essas ferramentas foram utilizadas para representar os diagramas de blocos do comportamento das maquinas de estado e operações do sistema.

Conforme relatado em Singh (2015), o teste consome cerca de 50% dos custos totais de desenvolvimento de software. O objetivo do teste é averiguar a exatidão de qualquer software desenvolvido, se está funcionando de acordo com o esperado e o que deveria ser realizado para atender aos requisitos especificados, caso seu funcionamento não corresponda ao comportamento esperado. O teste de software é um processo de execução de um programa ou aplicativo com o objetivo de encontrar defeitos. O objetivo do teste de software não é apenas encontrar os defeitos, mas também descobrir as situações que podem causar impacto negativo ao cliente. Segundo o autor, existem diversas técnicas e maneiras de testar o aplicativo e o software, sendo destacada a abordagem MBT como uma das metodologias eficientes e eficazes. Essa

abordagem facilita a geração de casos de teste efetivos a partir do modelo desenvolvido do software. Um modelo descreve a funcionalidade e o comportamento do sistema sob teste (SUT). O estudo de caso aplicado neste artigo foi em um robô. Foram utilizados os diagramas da UML para modelar e representar o comportamento do robô por meio de Máquina de Estado Finita (*Finite State Machine* - FSM).

O artigo selecionado de Herpel (2016) relata que o desenvolvimento e teste de software complexo exige tempo de desenvolvimento. Dessa forma o artigo apresenta a utilização da abordagem MBT para modelar Máquina de Estado do comportamento formal do software de bordo de um satélite e gerar casos de teste. No estudo de caso deste artigo foi utilizada a ferramenta *SBS-Super* para construir o modelo, a ferramenta *JUMBL* para gerar os casos de teste abstratos e as ferramentas *Airbus Defence and Space* para gerar os casos de teste executáveis. Foi relatado que a modelagem e geração de casos de teste auxiliam na representação dos requisitos e a realização dos testes respectivamente. Porém, essa cadeia de ferramentas pode dificultar a aplicação da abordagem aos projetos de negócios. O estudo de caso foi aplicado no *Model-bAsed Testing of saTellite on-board Software* (MATTS).

Conforme relatado no artigo de Schieferdecker et al. (2012), o teste de segurança visa validar os requisitos do sistema de software relacionados às propriedades de segurança, como confidencialidade, integridade, autenticação, autorização, disponibilidade e não repúdio. Durante os últimos anos surgiram poucas abordagens que permitem a especificação de casos de teste em um nível mais alto de abstração capaz de possibilitar orientação sobre identificação e especificação de teste, assim como geração automatizada de teste. O Teste de Segurança Baseado em Modelo (*Model-Based Security Testing* - MBST) é um campo relativamente novo e especialmente dedicado à especificação e documentação sistemáticas e eficientes dos objetivos nos testes de segurança, casos de teste e casos de teste de segurança, assim como a sua geração automatizada ou semi-automatizada. A combinação de modelagem de segurança e abordagens de geração de teste ainda é um desafio em pesquisa e de alto interesse para aplicações industriais. A MBST inclui testes funcionais

de segurança, técnica *fuzzing* baseado em modelo, testes orientados a riscos e ameaças, e o uso de padrões de teste de segurança. *Fuzzing* é uma técnica de testes de software, frequentemente automatizada ou semi-automatizada, que envolve prover dados inválidos, inesperados e aleatórios como entradas para programas de computador.

No artigo de Schieferdecker et al. (2012) foram relatados os seguintes procedimentos para gerar teste de segurança com uma abordagem baseada em modelo: i) identificar os objetivos e métodos do teste de segurança, ii) projetar um modelo de teste funcional, iii) determinar critérios de geração de testes, e iv) gerar os testes. Esse método foi utilizado no projeto europeu ITEA2-DIAMONDS (*Development and Industrial Application of Multi-Domain Security Testing Technologies*).

De acordo com Mattiello-Francisco (2009), o aumento da complexidade de funcionalidades de sistemas computacionais implementadas por software e necessidades de soluções mais robustas geram dificuldade cada vez maior no processo de V&V de sistemas intensivos em software. Atender aos requisitos necessita uma visão ampla do sistema com suas interações. Outra necessidade de análise é verificar se as implementações interoperam na presença de falhas temporais e de comunicação. A abordagem denominada InRob contribui com um processo de testes de integração baseado em modelos para V&V de propriedades de interoperabilidade e robustez de sistemas comunicantes intensivos em software.

Em Mattiello-Francisco (2012) a abordagem InRob é aplicada em um estudo de caso de integração de software embarcado em subsistemas comunicantes em satélites. A InRob possui cinco elementos chave: i) perfil do serviço, ii) modelo nominal do serviço, iii) perigos de tempo, iv) modelo aumentado do serviço, e v) propósito de teste. O InRob guia a construção de modelos formais de interoperabilidade, os quais representam o comportamento de um serviço em um dado estágio de integração. A geração automática de testes foi realizada por uma ferramenta acadêmica, denominada Hit-or-Jump to IF (HJ2IF) a partir

de modelos de interoperabilidade dos subsistemas comunicantes representados no formalismo *Timed Input Output Automata* (TIOA).

A abordagem InRob foi adaptada por Weller at al. (2015) para a Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML) para simplificar a modelagem dos sistemas sob teste e permitir gerar de forma automatizada os casos de teses de interoperabilidade e robustez. Denominado InRob-UML o método incluiu nos modelos de interoperabilidade a modelagem do Mecanismo Emulador de Falhas (*Failure Emulator Mechanism* - FEM) não detalhado na InRob original. Esse mecanismo recebe informações de um subsistema, injeta falha e envia para o subsistema de destino analisando seu comportamento na presença de falha, permitindo assim testar propriedades de robustez dos subsistemas comunicantes.

Um estudo de caso realizado com a InRob-UML foi um sistema de controle de passagem de nível em uma ferrovia, amplamente utilizado na literatura. O método utilizou a linguagem UML para modelar o estado de um sistema ferroviário, utilizando a abordagem MBT para gerar casos de teste, testar a interoperabilidade entre os sistemas envolvidos e a robustez desses sistemas por meio do FEM.

O artigo de Batista et al. (2018) apresenta uma arquitetura de testes, a qual utiliza um Mecanismo Emulador de Falha, denominada FEM, para realizar testes de robustez de subsistemas intensivos em software a bordo de nanosatélites. O FEM atua no canal de comunicação realizando testes de integração entre os subsistemas comunicantes sob teste nas fases de: i) especificação de requisitos de robustez usando modelo no loop (MIL), e ii) validação de robustez usando hardware no loop (HIL). Foi utilizado o formalismo timed automata para realizar a modelagem dos subsistemas comunicantes e gerar os casos de teste. O nanosatélite denominado NanosatC-BR2 que está em desenvolvimento no Instituto Nacional de Pesquisas Espaciais (INPE) foi utilizado para ser realizado testes com o FEM.

De acordo com Pinheiro et al. (2014), o avanço da tecnologia espacial propiciou o desenvolvimento de software mais complexo nas aplicações de

satélites. A integração de sistemas complexos levou à demanda de novas abordagens para a realização de V&V nas missões espaciais. Uma dessas abordagens é a de Testes Baseados em Modelos (*Model-Based Testing* - MBT). A realização da modelagem é um trabalho minucioso que busca apontar os requisitos necessários para atender à missão e também identificar possíveis falhas que possam ocorrer.

Uma metodologia de teste que busca identificar falhas de um sistema por meio da geração de casos de teses de forma automática denomina-se Conformidade e Injeção de Falhas (*Conformance and Fault Injection* - CoFI) (AMBROSIO, 2005). Os passos dessa metodologia compreendem identificar os estados normais, caminhos furtivos, exceções especificadas e tolerâncias a falhas na modelagem do sistema em teste (*System Under Test* - SUT).

A utilização da CoFI visa gerar de forma automática os casos de teste a partir do modelo de Máquina de Estado Finita (*Finite State Machine* - FSM). Algumas ferramentas computacionais auxiliam na geração dos casos de teste. No trabalho apresentando sobre CoFI, a modelagem foi elaborada utilizando a ferramenta Modelador de Máquinas de Estado (MME) e a geração dos códigos de teste foi por meio da ferramenta CONDADO (MARTINS, 1999).

3.2. **Abordagens e métodos de engenharia orientada a modelos**

Segundo Puente et al. (2014), a abordagem Engenharia Dirigida por Modelo (*Model-Driven Engineering* - MDE) vem sendo cada vez mais utilizada por apresentar um meio de elevar o nível de abstração no desenvolvimento de software, reduzindo custos e aumentando a eficiência. Essa abordagem de desenvolvimento de software permite aos engenheiros utilizarem abstrações de alto nível para definir os componentes de um sistema ao longo do ciclo do desenvolvimento do software.

A MDE utiliza a modelagem dos requisitos para descrever e analisar o comportamento do sistema auxiliando na lógica das propriedades do sistema em um nível abstrato. Os modelos fornecem valia para descrição de conceitos,

validação desses conceitos com base em técnicas de verificação e análise, geração de código e outros componentes de implementação. Os sistemas de controle são habitualmente projetados usando modelos abstratos baseados em fundamentos matemáticos bem estabelecidos. Algumas ferramentas de desenvolvimento para modelagem e de engenharia de software possibilitam a geração de forma automatizada dos códigos computacionais da grande parte do software. A utilização da MDE possibilita representar a arquitetura do software e dos principais detalhes do projeto em um nível abstrato. A utilização de testes baseados em modelos implementados na validação do *Hardware-in-loop* (HIL), também apresenta ser uma estratégia plausível, economizando tempo significativo e facilitando problemas em uma fase inicial de desenvolvimento do software.

O artigo selecionado de Souza (2009) utiliza a abordagem MDE para apresentar o desenvolvimento de uma estrutura de Caso de Teste Automático Baseado em Modelos (*Automatic Test Case based on Models* - ATCM). O objetivo é minimizar a injeção de erros durante a geração de casos de teste buscando garantir a qualidade do software. Essa estrutura apresenta dois processos distintos: i) processo de geração de software de teste, e ii) processo de geração do código fonte do software por meio de transformação de Modelo Independente de Plataforma (*Platform Independent Model* - PIM) para Modelo Específico de Plataforma (*Platform Specific Model* - PSM).

Outro artigo relacionado foi o de Montecchi et al. (2011), o qual apresenta um fluxo de trabalho de transformação de modelos para análise de dependabilidade de uma infraestrutura industrial em termos de qualidade da especificação, análise e verificação de propriedades extra-funcionais do sistema. As seguintes definições são apresentadas: i) definição das etapas de transformação necessárias para avaliar automaticamente as propriedades de dependabilidade do sistema a partir de uma linguagem de modelagem denominada CHESS, ii) definição de um novo Modelo Intermediário de Dependabilidade (*Intermediate Dependability Model* - IDM) atuando como ponte entre a linguagem de modelagem e os modelos de análise de baixo

nível, e iii) definição de transformações da linguagem de modelagem para modelos de IDM.

A filosofia da linguagem de modelagem CHESS utilizada refere-se a uma iniciativa MDE específica, a Arquitetura Dirigida por Modelo (*Model-Driven Architecture* - MDA) definida pelo Grupo de Gerenciamento de Objetos (*Object Management Group* - OMG). No fluxo de trabalho promovido pelo MDA, o desenvolvedor do sistema elabora um modelo PIM. A partir do modelo PIM, enriquecido com informações de implementação, um modelo PSM é gerado por transformações automatizadas. A partir do PSM, a geração do código pode ser acionada para obter uma implementação do sistema. Esse fluxo de trabalho da MDA foi utilizado no projeto ARTEMIS-JU CHESS.

O artigo de Puente et al. (2014) apresenta a utilização de algumas ferramentas MDE para modelar e gerar os códigos de forma automatizada do comportamento dos subsistemas de um microssatélite. Os procedimentos iniciais realizados foram: i) Modelo Independente de Plataforma (PIM) utilizando alguns meios para definir os tipos dos dados (notação ASN.1), ii) descrever o comportamento funcional dos componentes dos sistemas, e iii) descrever a arquitetura de integridade dos sistemas de software. Posteriormente foi realizada a definição do Modelo Específico de Plataforma (PSM).

Esse método foi utilizado no projeto do microssatélite UPMSat-2 (desenvolvido pela *Technical University of Madrid*) para realizar modelagem, simulação e geração de códigos visando à validação de software em diferentes níveis do processo, incluindo o HIL.

O artigo de Fernández-Isabel et al. (2015) apresenta um processo de dois estágios utilizando a abordagem MDE para um projeto de simulação de tráfego automotivo. O primeiro voltado para especialistas em tráfego e outro para projetistas de simulação. O primeiro estágio usa uma linguagem específica de modelagem definida como Linguagem de Modelagem de Tráfego (*Traffic Modelling Languages* - TML) e ferramentas do ambiente de desenvolvimento Eclipse para modelar e simular o comportamento dos agentes do ambiente e a

categoria dos elementos participantes. Isso possibilita o fornecimento de ferramentas personalizadas para especialistas. A segunda parte associa a MDE com a Engenharia de Software para os projetistas de baixo nível, orientado para a plataforma. As etapas do processo estão resumidas na seguinte sequencia: i) analisar o modelo abstrato, ii) construir o modelo teórico, ii) desenvolver as transformações, iv) construir a simulação dos modelos, e v) utilizar ferramentas MDE (exemplo, Eclipse) para simular os modelos.

No artigo de Abrahão et al. (2017) foi realizado pesquisas sobre a utilização da abordagem MDE e suas ferramentas para desenvolvimento de projetos com foco nas facilidades e dificuldades da utilização desses recursos pelos desenvolvedores. A integração das ferramentas MDE com outras ferramentas também é outro ponto que deve ser observado. Nesse artigo foi realizado um estudo de caso na pesquisa relacionada à eXperiência do Usuário (*User eXperience* - UX). Esse artigo também relata que a utilização de padrões e métodos de desenvolvimento conforme o usuário e o projeto poderia ser um caminho que a abordagem MDE também deveria abranger.

Conforme Kaslow et al. (2015), a Engenharia de Sistemas Baseados em Modelos (*Model-Based System Engineering* - MBSE) é uma abordagem que visa apoiar a engenharia de sistemas. Essa abordagem permite beneficiar o desenvolvimento das missões que utilizam Cubesats desenvolvendo um modelo para auxiliar na integração de outros modelos de engenharia e simulação. Os modelos MBSE apresentam informações consistentes dos requisitos do sistema e do projeto auxiliando na análise e verificação.

O Conselho Internacional de Engenharia de Sistemas (*International Council on Systems Engineering* - INCOSE) instituiu um comitê de desenvolvimento em Engenharia de Sistemas Baseados em Modelos em 2007, a qual contém uma equipe averiguando a aplicabilidade do MBSE para projetar Cubesats desde 2011. O objetivo é fornecer um modelo de referência suficientemente completo que possa ser adaptada à variedade de projetos que envolvem Cubesats. A linguagem de modelagem que vem sendo utilizada na MBSE é a Linguagem de Modelagem de Sistemas (*Systems Modeling Language* - SysML), a qual é uma

modelagem gráfica. Essa modelagem é uma extensão da Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML).

A SysML abrange recursos com facilidades de especificações de requisitos, estrutura, comportamento, projetos, verificação e validação. A abordagem MBSE está em desenvolvimento para integrar outros modelos de simulações de engenharia. A modelagem MBSE e o uso de linguagens padrão como SysML e UML permitem uma comunicação mais integrada entre as equipes de engenharia.

O estudo de caso realizado no artigo de Kaslow et al. (2015) foi a Missão Radio Aurora Explorer (RAX), o qual é um projeto da Cal Poly Cubesat (*Califórnia Polytechnic State University*).

3.2.1. **Modelagem e geração de códigos**

A modelagem e geração de códigos computacionais de forma automatizada têm evoluído com o passar dos anos devido ao avanço tecnológico. Os sistemas tem sido cada vez mais complexos e precisam ser desenvolvidos o mais rápido possível com alta confiabilidade (PERROTIN; YABAR, 2014). Alguns erros no desenvolvimento do sistema podem surgir devido às especificações errôneas, projeto ineficiente, erros de programação, linguagem de programação inadequada, processos mal definidos, entre outros.

A Agência Espacial Europeia (*European Space Agency* – ESA) está buscando definir um padrão para desenvolver modelos e códigos usando a família das ferramentas MATLAB e Simulink. O objetivo é produzir códigos reutilizáveis por meio de padrões e diretrizes. Essas ferramentas estão sendo utilizadas em simuladores e software de voo.

No ciclo de desenvolvimento do software deve ser analisado o comportamento do sistema, as informações que serão utilizadas, os algoritmos utilizados, a arquitetura do sistema, os requisitos, entre outras analises e representações que permitam identificar e atingir o objetivo do projeto. A modelagem de sistemas passou a ser utilizada na indústria espacial no início dos anos 2000. A

primeira especificação de satélite e software baseado em modelagem e geração automatizada de códigos foi utilizada no Smart-1, sendo lançado em 2002. Com o avanço tecnológico diversas ferramentas de modelagem e geração de códigos foram desenvolvidas entre elas: Matlab, Simulink, SCADE, Eclipse, entre outras (PERROTIN; YABAR, 2014).

3.3. **Dependabilidade e mitigação**

Nos últimos anos aumentou a quantidade de satélites de pequeno porte que estão sendo utilizados em experimentos no espaço de curta duração a baixo custo comparado aos satélites de maior porte. Diversas universidades no mundo estão utilizando plataformas Cubesat como programa educacional espacial. Essas plataformas também estão sendo utilizadas para validação de novas tecnologias espaciais em órbita, em situações como demonstração tecnológica, aplicações na ciência, observação da terra, entre outras pesquisas espaciais.

Com o uso dessas plataformas em diversas aplicações existe a necessidade de avanço tecnológicos para aumentar o desempenho dos Cubesats. Outra preocupação é a confiabilidade na missão (RABASA, 2015). Com a necessidade de aumentar a confiabilidade nessa evolução tecnologia, novas metodologias e conceitos estão sendo desenvolvidos e aplicados. Essas metodologias e conceitos foram agrupadas em três blocos principais:

- O primeiro bloco tem como foco o estudo geral da dependabilidade do ponto de vista teórico. O estudo envolve analisar os atributos que podem afetar a dependabilidade de um sistema, técnicas usadas para mitigar as possíveis falhas, parâmetros para medir a dependabilidade e modelos e técnicas para a modelagem da dependabilidade.
- O segundo bloco foca analisar as falhas ocorridas durante as missões envolvendo Cubesats nos últimos dez anos e analisando a confiabilidade. Com esses resultados foi desenvolvida uma base de dados contendo informações de Cubesats lançados até 2013. Essas

informações foram obtidas a partir de fontes da *Web*, de projetos de Cubesats publicados, entre outros meios de consulta. A base de dados contém informações como missão, data de lançamento, objetivos da missão, possíveis falhas, entre outras, relevante para o estudo em questão. A utilização dessas informações é para realizar uma análise de confiabilidade quantitativa.

- O terceiro bloco foca os conceitos para prevenção e tolerância a falhas analisados objetivando dependabilidade, e consequentemente a confiabilidade, em missões envolvendo Cubesats. Para buscar atingir esse objetivo foram aplicados três métodos diferenciados:
  - O primeiro realiza uma análise das atividades executadas durante o ciclo de vida no desenvolvimento de um Cubesat. Diversas atividades foram determinadas visando aumentar o sucesso da missão.
  - O segundo destina-se a aumentar a confiabilidade por meio de adaptar a missão aos padrões internacionais ECSS.
  - O terceiro é a implementação de uma arquitetura de missão distribuída ao invés de arquiteturas monolíticas clássicas.

No artigo de Rabasa (2015) o estudo de caso foi aplicado em um projeto espacial real em desenvolvimento da *Politécnico di Torino* no programa *Cubesat e-st@r*. Parte da pesquisa foi conduzida no *European Space research and Technology Centre* (ESTEC) da *European Space Agency* (ESA).

3.4. **Resumo dos métodos e abordagens selecionados**

A partir das informações obtidas nas referências bibliográficas apresentadas na seção anterior foi elaborado uma tabela que sintetiza as contribuições das publicações por ano de publicação e respectivo autor. Destacam-se na tabela os métodos e ferramentas utilizadas pelos autores nos artigos, bem como um resumo do artigo com o estudo de caso. A Tabela 3.1 apresenta as abordagens selecionadas.

Tabela 3.1 - Abordagens para tratamento de falhas em sistemas complexos.

Artigo	Ano	Autor	Alguns dos recursos utilizados	Resumo
1	2009	Grades	Linguagem UML: Diagramas e modelagem	Utiliza a abordagem MBT relatando a utilização de um algoritmo para geração de casos de teste no estudo de caso em caixa eletrônico ATM.
2	2009	Souza	Linguagem UML: Ambiente Eclipse (JUnit, NUnit	Utiliza a abordagem MDE buscando qualidade de software e minimizar injeção de falhas. Apresenta uma estrutura de Caso de teste Automático Baseado em Modelos (ATCM).
3	2009 2012	Mattiello-Francisco	Formalismo TIOA Ferramenta Hit-or-Jump to IF (HJ2IF)	InRob orienta a construção de modelos formais de interoperabilidade os quais representam o comportamento de um serviço em um dado estágio de integração. Utiliza a abordagem MBT para gerar casos de teste em um estudo de caso de sistemas espaciais intensivos em software.
4	2010	Ali et al.	Ferramenta Transformation-based for UM-baSed Testing (TRUST) Linguagem UML	Apresenta uma abordagem para projetar e desenvolver ferramentas da abordagem MBT baseadas na tecnologia de transformação de modelo (*TRansformation-based tools for Uml-baSed Testing* - TRUST) baseado em máquinas de estado modeladas em UML. Estudo de caso em um sistema de conferência de multimídia e controle de segurança.
5	2011	Montecchi et al.	Linguagem UML. Análise de dependabilidade. Modelagem CHESS	Apresenta um fluxo de trabalho de transformação para análise de dependabilidade que faz parte de uma infraestrutura de qualidade industrial para a especificação, análise e verificação de propriedades extra-funcionais. Utiliza a abordagem MDE.
6	2012	Schieferdecker et al.	Linguagem UML	Este artigo fornece uma pesquisa sobre as técnicas do MBST e os modelos relacionados, bem como exemplos de métodos e ferramentas que estão sendo desenvolvidos no projeto europeu ITEA2-DIAMONDS (*Development and Industrial Application of Multi-Domain Security Testing Technologies*).
7	2013	Peleska	Linguagem: UML, SysML Ferramenta: Matlab, Simulink	Apresenta a ferramenta RT_Tester2, a qual suporta UML e SysML em um estudo de caso industrial. Também descreve a capacidade da abordagem MBT apresentando as abordagens que ajudam a atender aos requisitos relacionados a teste de V&V.
8	2014	Puente et al.	Ferramenta: Simulink, SDL, TASTE Linguagem: Ada	Apresenta a utilização de algumas ferramentas MDE para modelar e gerar os códigos de forma automatizada do comportamento dos subsistemas de um microssatélite. Estudo de caso realizado no projeto do software de tempo real do microssatélite UPMSat-2 desenvolvido pela *Technical University of Madrid* (UPM).
9	2014	Perrotin e Yabar	Ferramentas Matlab, Simulink	A ESA está buscando definir um padrão para criar modelos e códigos usando a família das ferramentas MATLAB e Simulink. O objetivo é produzir códigos reutilizáveis por meio de padrões e diretrizes. As ferramentas em desenvolvimento estão sendo utilizadas em simuladores e software de voo.
10	2014	Pinheiro et al.	JPlavisFSM	Apresenta a metodologia Conformidade e Injeção de Falhas (CoFI), a qual realiza testes buscando identificar falhas por meio da geração de casos de teses de forma automática (MBT). Os passos desta metodologia é identificar os estados normais, caminhos furtivos, exceções especificadas e tolerâncias a falhas na modelagem.
11	2015	Fernández-Isabel et al.	Linguagem UML, Ambiente: Ecore (Eclipse), Delphi	O artigo apresenta um processo de dois estágios utilizando a abordagem MDE para um projeto de simulação de tráfego automotivo.

continua

Tabela 3.1 - Conclusão.

12	2015	Singh e Ramasamy	Linguagem UML	Aplicação da abordagem MBT para gerar casos de teste em um estudo de caso de um robô (*ROBOT*). O objetivo do teste de software não é apenas encontrar os defeitos, mas também descobrir as situações que podem causar impacto negativo ao cliente.
13	2015	Kaslow et al.	Linguagem SysML	A abordagem utilizada foi a MBSE utilizando a linguagem SysML com seus diagramas para representar os modelos do projeto do Cubesat a ser testado. Foi realizado um esforço para desenvolver um modelo de referência Cubesat que outros projetos possam usar como ponto de partida para seu modelo Cubesat específico da missão. O estudo de caso foi a Missão Radio Aurora Explorer (RAX), o qual é um projeto da Cal Poly Cubesat (*Califórnia Polytechnic State University*).
14	2015	Rabasa	Tabela FMECA	A pesquisa do artigo objetiva estudar abordagens para análise de dependabilidade conduzida por pequenos satélites com atenção voltada à confiabilidade das missões e mitigações a serem realizadas. Melhorar a atividade de verificação, avaliar os atributos de dependabilidade e utilizar padrões internacionais, como ECSS, são alguns dos meios para aumentar a confiabilidade da missão. Estudo de caso aplicado em um projeto espacial real em desenvolvimento da *Politécnico di Torino* no programa Cubesat e-st@r.
15	2016	Herpel	Ferramenta SBS-Super (Sequence Based Specification)	O artigo relata que o desenvolvimento e teste de software complexo exige tempo de desenvolvimento. Dessa forma o artigo apresenta a utilização da abordagem MBT para modelar Máquina de Estado do comportamento formal do software de bordo de um satélite e gerar casos de teste. O estudo de caso foi realizado no *Model-bAsed Testing of saTellite on-board Software* (MATTS).
16	2017	Abrahão et al.	Linguagem: UML, SysML Ferramenta: Papyrus, DOORS	O artigo pesquisa sobre a utilização da abordagem MDE e suas ferramentas para desenvolvimento de projetos com foco nas facilidades e dificuldades da utilização desses recursos pelos desenvolvedores. O artigo relata que a utilização de padrões e métodos de desenvolvimento conforme o usuário e projeto poderia ser um caminho que a abordagem MDE também deveria abranger. O estudo de caso foi realizado na pesquisa relacionada à eXperiência do Usuário (*User eXperience* - UX).
17	2018	Batista et al.	FEM e formalismo *timed automata*	Utiliza uma arquitetura de testes denominada FEM para realizar testes de robustez em subsistemas intensivos em software de nanosatélites. Estudo de caso o nanosatélite denominado NanosatC-BR2 em desenvolvimento no INPE.

Fonte: Produção do autor.

Pode ser observado que houve uma evolução nas abordagens de modelagem buscando facilidade e melhor confiança no desenvolvimento de software complexo conforme as necessidades tecnológicas. Porém, as abordagens em geral são apresentadas sendo utilizadas separadamente. A utilização de algumas dessas abordagens de forma que um possa complementar a outra poderá permitir aprimorar o tratamento de falhas em software complexo.

4 ABORDAGEM SISTEMÁTICA DE TESTES COM ARQUITETURA ESCALÁVEL

Nos satélites, sejam de grande ou pequeno porte, a mesma filosofia de testes é adotada ao longo do desenvolvimento da missão. No caso dos satélites que utilizam plataformas padronizadas tipo Cubesats, pequenos satélites com estrutura modular em forma cúbica, a infraestrutura de Equipamento para Suporte em Solo (*Ground Support Equipments* – GSE) para apoio aos testes funcionais dos nanosatélites é ainda a mesma utilizada nos testes de satélites, cujo desenvolvimento está baseado em abordagens mais conservadoras. Os laboratórios existentes para apoiar os testes de satélites de grande porte são onerosos para os projetos de nanosatélites de baixo custo (JACKLIN, 2015).

A abordagem sistemática de testes desenvolvida e apresentada nesta tese permite a reutilização da arquitetura do Sistema de Teste STAE de forma escalável em diferentes fases do desenvolvimento de uma mesma missão ou de satélites da mesma família. A abordagem apoia o processo de V&V de missões espaciais desde o início do ciclo de desenvolvimento reduzindo a baixo custo o tempo de desenvolvimento.

Tal feito se deve ao foco da abordagem nas questões de interoperabilidade entre dois subsistemas comunicantes de um nanosatélite, sendo um o computador de bordo e o outro uma carga útil. O tratamento das falhas identificadas é feito por meio da adição de requisitos de robustez na especificação comportamental de cada subsistema comunicante ainda na fase inicial de projeto.

Partindo das especificidades do protocolo de comunicação que será utilizado na interação de subsistemas sob teste, cujas funcionalidades serão implementadas por software, a abordagem apresentada utiliza diagramas de Máquina de Estado Finita para modelar e simular os estados desses subsistemas em interoperabilidade baseadas nos requisitos do comportamento nominal funcional de cada subsistema. São representadas duas máquinas de estado em interoperabilidade. Cada máquina representa os estados de um subsistema.

A abordagem desenvolvida utiliza a técnica MBT para modelar e simular o comportamento dos subsistemas comunicantes em interação, baseado nas especificações de requisitos, com objetivo de tratamento de falhas de interoperabilidade por simulação antes da fase de integração. As falhas identificadas são utilizadas como estímulos para gerar casos de testes possibilitando identificar e realizar o tratamento de outras possíveis falhas que possam ocorrer. Uma ferramenta MBT deve ser selecionada para modelar o comportamento dos subsistemas comunicantes em máquinas de estado finita que representam os requisitos da interação. A partir do modelo, a ferramenta deve gerar casos de teste de forma automatizada representando os estados percorridos, e ser gratuita.

Outra técnica utilizada para complementar os testes é a MDE. Os modelos especificados com a ferramenta MBT são transformados para serem utilizados por ferramentas MDE, cujo objetivo é gerar de forma automatizada códigos computacionais que simulam por software os subsistemas comunicantes. As ferramentas MDE selecionadas para apoiar a abordagem desenvolvida além de atenderem os requisitos de gerar códigos computacionais de forma automatizada, deve permitir que os códigos gerados possam ser embarcados em placas computacionais programáveis COTS, e serem ferramentas gratuitas.

As possíveis falhas identificadas durante os testes são classificadas em uma planilha de dependabilidade e uma tabela FMEA, desenvolvidas com base nos conceitos de dependabilidade da Taxonomia de Avizienis et al. (2004). As informações dessa planilha e a tabela são utilizadas no tratamento de falhas, cujo objetivo é evitar que as falhas identificadas em ambiente simulado ocorram durante a interação entre os subsistemas comunicantes reais.

A utilização das técnicas MBT e MDE de forma complementar, em conjunto com a taxonomia de dependabilidade, possibilita sistematizar a identificação e o tratamento de falhas por meio de testes de interoperabilidade e robustez dos subsistemas comunicantes ao longo do processo de V&V. Dessa forma, as falhas identificadas durante os testes em ambiente simulado auxiliam no

tratamento de possíveis falhas nos subsistemas reais antes de serem submetidos aos testes de integração.

A abordagem sistemática desenvolvida nesta tese foi utilizada para apoiar o processo de V&V do NanosatC-BR2, um nanosatélite 2U em desenvolvimento no INPE, em cooperação com a UFSM e outras universidades Brasileiras, com previsão de lançamento para o final de 2019. Um protótipo do Sistema de Teste com Arquitetura Escalável (STAE) foi desenvolvido pelo autor para aplicar os casos de testes gerados pela abordagem sistemática nas diferentes fases de desenvolvimento dos subsistemas comunicantes intensivos em software embarcados no NanosatC-BR2.

4.1. **Procedimento para a implementação da abordagem da tese**

Para melhor representar os principais elementos considerados na abordagem sistemática desenvolvida nessa tese um diagrama IDEF0 (Definição de Integração para Modelagem de Funções - *Integration Definition for Function Modeling*) foi utilizado, categorizando os elementos em termos de entradas, mecanismos, controles e saídas (SERIFI, 2008).

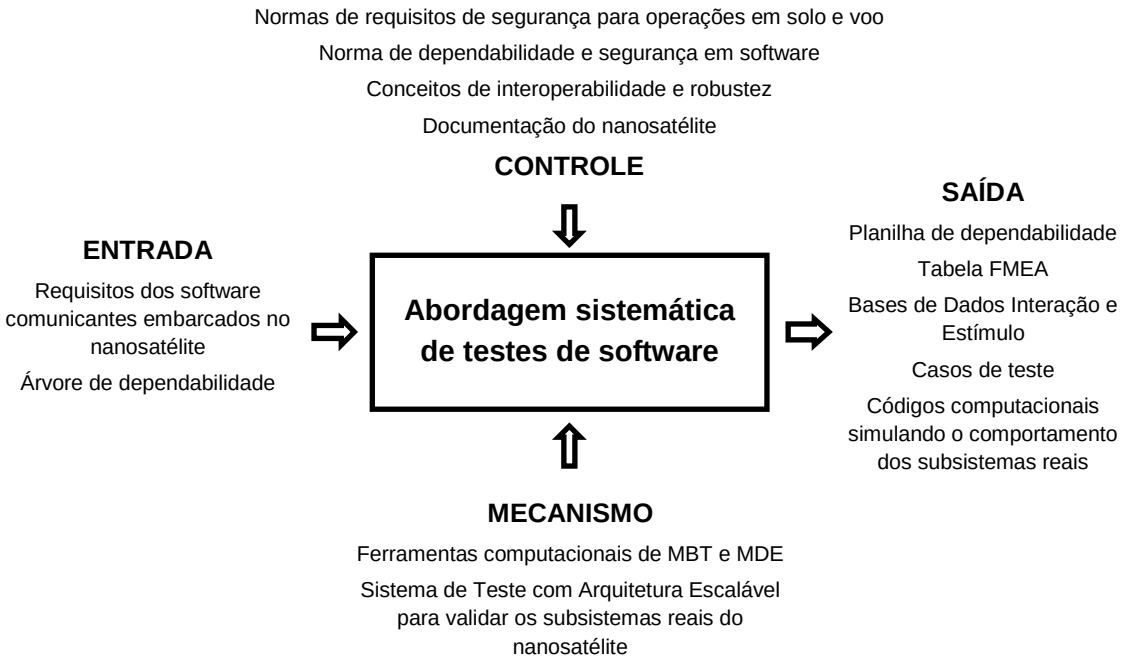
O diagrama apresentado na Figura 4.1 contém as seguintes informações:

- Na parte superior são considerados elementos de controle: normas, conceitos e documentos que servirão de controle na abordagem.
- Do lado esquerdo estão as informações consideradas entradas para a aplicação da abordagem.
- Na parte inferior encontram-se os mecanismos que auxiliam na execução da abordagem, tais como as ferramentas MBT e MDE.
- Do lado direito estão as informações de saída geradas pela sistemática de testes.

Assim, a abordagem utiliza as entradas, os controles e os mecanismos de auxílio para realizar as atividades para tratamento de falhas e a geração de códigos computacionais de forma automatizada simulando o comportamento

dos subsistemas comunicantes reais. Esses códigos serão gerados a partir da modelagem comportamental dos requisitos dos subsistemas do nanosatélite.

Figura 4.1 - Diagrama IDEF0 do STAE no processo de V&V para nanosatélite.



Fonte: Produção do autor.

Os códigos gerados serão utilizados para simular o comportamento dos subsistemas reais auxiliando na realização de V&V no nanosatélite.

4.1.1. **Controle**

As missões espaciais, sejam de pequenos ou grandes satélites, seguem algumas normas internacionais apresentadas na seção 1.3:

- ABNT NBR ISO 14620-1, 2 e 3 (2009): referente aos requisitos de segurança para realizar operações em solo e voo.
- ECSS-Q-HB-80-03 (2012): descreve conceitos de dependabilidade e segurança no desenvolvimento de software de sistemas.

Essas normas foram analisadas e utilizadas como referência para aplicação na abordagem apresentada.

Outros documentos analisados foram:

- Conceitos de interoperabilidade e robustez: utilizado para especificar, gerar, executar e analisar os testes entre os subsistemas intensivos em software que se comunicam.
- Documentação do nanosatélite necessária para identificar e analisar os requisitos da missão e dos subsistemas alvo dos testes.

4.1.2. **Entrada**

As informações de entrada da abordagem apresentada serão os requisitos de interoperabilidade do software comunicante de cada subsistema do nanosatélite em V&V e a árvore de dependabilidade para analisar o tratamento de falhas.

Esses requisitos serão utilizados para definir a modelagem comportamental dos subsistemas em interação que será representada graficamente por Máquina de Estado Finita utilizando aplicativos computacionais que possibilitem a sua simulação e verificação.

4.1.3. **Mecanismo**

Os mecanismos utilizados para a execução da abordagem apresentada são as ferramentas computacionais que permitem a utilização das técnicas MBT e MDE e do Sistema de Teste com Arquitetura Escalável (STAE) para executar os testes gerados pela abordagem produzida nesta tese.

Dentre diversos ambientes de desenvolvimento MDE, foi escolhido o Eclipse (2016), por possuir uma ferramenta denominada Yakindu (2017), a qual possibilita gerar códigos computacionais de forma automatizada em diversas linguagens computacionais de terceira geração. Um diferencial da ferramenta

Yakindu é gerar os códigos para o ambiente de desenvolvimento Arduino (MCROBERTS, 2011). As placas computacionais Arduino são *COTS*, permitem serem configuradas com códigos computacionais e existem diversas bibliotecas gratuitas disponíveis na internet que facilitam a implementação de recursos como protocolo de comunicação, acesso a base de dados e conexão com ethernet.

A ferramenta MBT escolhida foi a Uppaal. Essa ferramenta além de ser gratuita possui os seguintes recursos: i) utiliza o formalismo *timed automata* (conjunto finito de relógios de valor real), ii) permite simular as interações entre os subsistemas comunicantes modelados, iii) gera diagrama de sequência permitindo verificar a mudança dos estados durante a simulação, e iv) gera de forma automatizada casos de teste, os quais irão permitir verificar se a implementação real se comporta conforme esperado, de acordo com a modelagem.

E por fim, é utilizado um Sistema de Teste com Arquitetura Escalável que permite embarcar os códigos gerados para fins de execução dos casos de testes em ambiente simulado.

4.1.4. **Abordagem sistemática de testes de software**

A abordagem sistemática consiste na construção evolutiva de modelos iniciada a partir dos requisitos comportamentais dos subsistemas comunicantes do satélite tendo como foco a modelagem desses subsistemas em interação. O objetivo da abordagem é identificar de forma sistemática possíveis falhas de interoperabilidade por simulação antes dos subsistemas reais serem integrados ao satélite. As falhas identificadas são classificadas na planilha de dependabilidade e tabela FMEA desenvolvidas nesta tese. Essa classificação é utilizada no tratamento de falhas buscando evitar que as falhas identificadas ocorram na interação entre os subsistemas reais. As falhas identificadas são utilizadas para gerar estímulos de falhas nos subsistemas simulados gerando novos casos de teste.. A ocorrência de novas falhas leva ao refinamento dos

modelos com requisitos de robustez, fruto da análise da falha classificada na planilha de dependabilidade e a tabela FMEA desenvolvidas na tese.

4.1.5. **Saída**

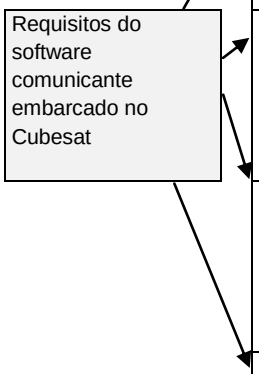
Após a execução das atividades de testes utilizando as entradas, controles e mecanismos descritos anteriormente, têm-se como saídas a geração dos casos de teste abstratos, os códigos computacionais que simulam o comportamento dos subsistemas, a planilha de dependabilidade, a tabela FMEA e as bases de dados integração e estímulo.

4.2. **Verificação da implementação da abordagem da tese**

A abordagem desenvolvida é estruturada em seis módulos de teste os quais contam com o Sistema de Teste STAE para serem executados. .

A Figura 4.2 sintetiza os procedimentos associados a cada módulo e suas vantagens.

Figura 4.2 - Módulos de teste com procedimentos e vantagens.



Procedimentos	Módulos	Vantagens
Testar e validar modelagem (MIL). Testar e validar sistemas simulados (SIL) e casos de teste. Verificar os resultados conforme a taxonomia de dependabilidade. Elaborar mitigações para remoção de falhas.	1 2 3	Facilidade e agilidade em alterações e remodelagem dos requisitos no início do ciclo do desenvolvimento do projeto. Gerar de forma automatizada os códigos dos subsistemas modelados e casos de teste.
Testar e validar os subsistemas reais no barramento (HIL) separadamente. Verificar os resultados conforme a taxonomia de dependabilidade. Elaborar mitigações para tratamento de falhas.	4	Identificar possíveis falhas e qual (is) subsistemas (s) necessita (m) alteração (ções).
Testar e validar interoperabilidade entre os subsistemas reais no loop (HIL). Verificar os resultados conforme a taxonomia de dependabilidade. Elaborar mitigações para tratamento de falhas.	5	Verificar se a interoperabilidade das informações entre os subsistemas comunicantes está condizente com as especificações dos requisitos.
Testar e validar os requisitos de robustez entre os subsistemas reais no loop (HIL). Verificar os resultados conforme a taxonomia de dependabilidade. Elaborar mitigações para tratamento de falhas.	6	Verificar os requisitos de robustez dos subsistemas reais por meio de injeção de falhas.

Fonte: Autor.

Em todos os módulos os resultados dos procedimentos de testes realizados são analisados conforme os conceitos de dependabilidade (AVIZIENIS et al., 2004).

Em todos os módulos é possível alterar requisitos no modelo original realizando a remodelagem de forma dinâmica com o uso das ferramentas de modelagem MBT e MDE. Essas ferramentas possibilitam gerar novamente os códigos computacionais e casos de teste de forma automatizada. Os testes serão realizados ainda em uma fase inicial do ciclo de desenvolvimento do projeto permitindo verificar se algum subsistema necessita ser alterado antes da fase de integração com o nanosatélite.

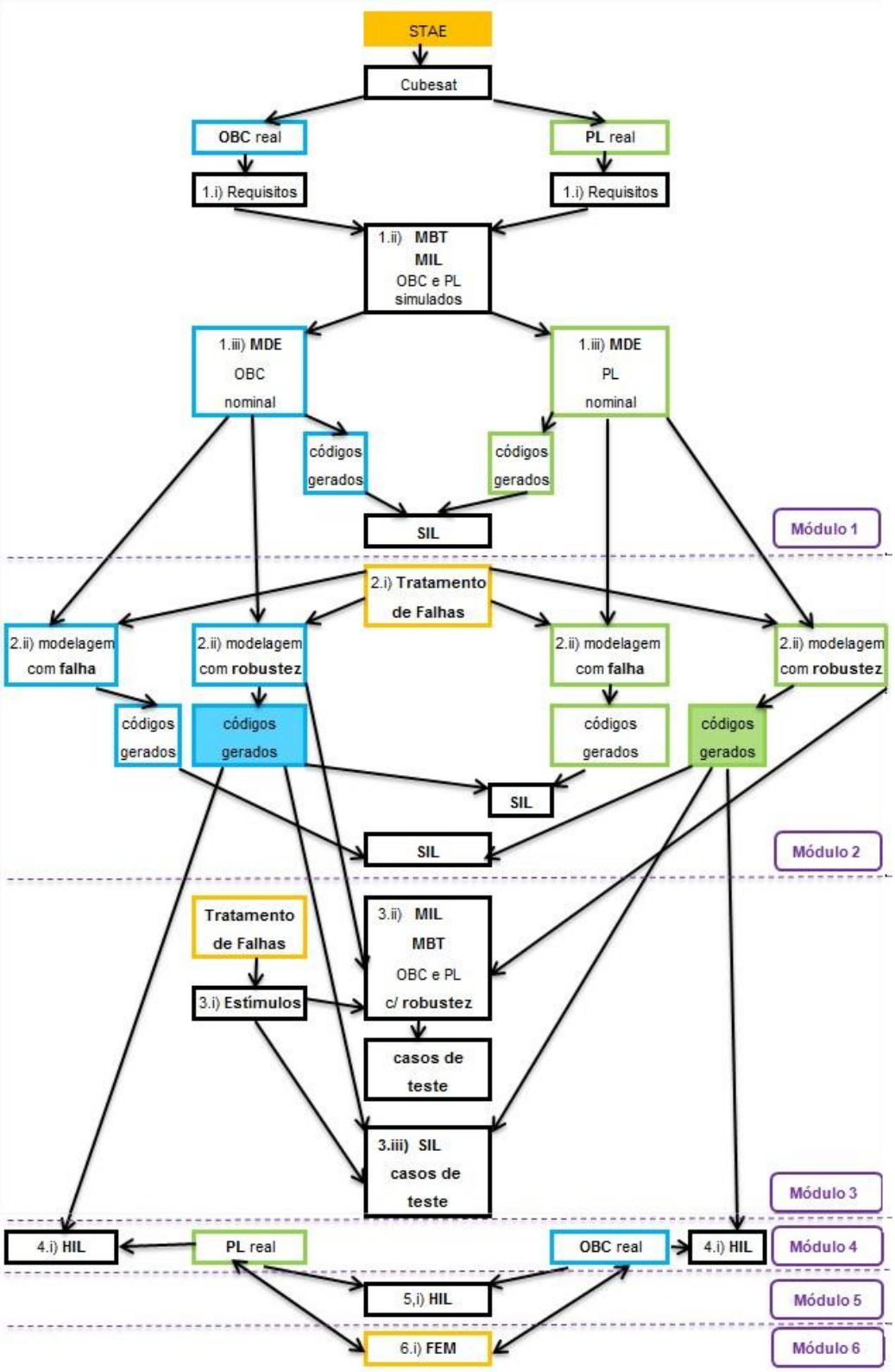
4.3. **Sistemática de testes**

A abordagem apresentada sistematiza testes de interoperabilidade entre subsistemas intensivos em software comunicantes em missões espaciais de nanosatélites. A abordagem permite a reutilização dos testes de forma sistemática estruturada em seis módulos. Os módulos estão associados às transformações dos modelos e às validações necessárias alcançadas para avançar com os testes. A Figura 4.3 apresenta essa sistemática, sendo que, a identificação utilizada na figura deve ser interpretada da seguinte forma: i) o primeiro número em algarismo numérico identifica o módulo de teste a ser realizado, e ii) o segundo número em algarismo romano identifica o procedimento dentro do referido módulo. Os procedimentos realizados na sistemática de testes são: 1.i) especificar os requisitos do software comunicante no comportamento nominal funcional de interação dos subsistemas sob teste, 1.ii) baseado nos requisitos, realizar a modelagem nominal comportamental desses subsistemas, cujo objetivo é simular e verificar o comportamento do OBC e de uma carga útil em interoperabilidade por meio da validação dos modelos (MIL - *Model-in-the-loop*), 1.iii) gerar os códigos computacionais desses modelos simulados utilizando ferramentas MDE, cujo objetivo é testar, verificar e validar os requisitos de interoperabilidade dos subsistemas modelados embarcados em ambiente simulado, com uso do barramento de comunicação real (SIL- *Software-in-the-loop*), 2.i) utilizar a taxonomia de dependabilidade (amarelo) de (AVIZIENIS et al., 2004), cujo objetivo é identificar e classificar possíveis falhas que possam ocorrer na interação entre os subsistemas comunicantes e analisar causa-efeito para mitigações das falhas identificadas, 2.ii) implementar falhas e robustez nos modelos simulados do OBC e da carga útil, cujo objetivo é identificar e evitar possíveis falhas e gerar modelos com robustez de ambos os subsistemas simulados, 3,i) utilizar as falhas identificadas para gerar estímulos de falhas nos modelos com robustez, cujo objetivo é gerar casos de teste abstratos, 3.ii) utilizar a técnica MBT para gerar os casos de teste a partir dos estímulos implementados nos modelos com robustez, cujo objetivo é verificar a interoperabilidade e robustez desses modelos em interação (MIL), 3.iii) utilizar

a técnica MDE para gerar e embarcar em placas computacionais os códigos desses modelos com robustez, cujo objetivo é validar os casos de teste gerados realizando estímulos de falhas nessas placas conectadas ao barramento de comunicação (SIL), 4.i) testar separadamente os modelos com robustez simulados no barramento com os subsistemas reais, cujo objetivo é verificar e validar os requisitos de interação dos subsistemas reais no barramento (HIL - *Hardware-in-the-loop*) separadamente, 5.i) testar conjuntamente no barramento os subsistemas reais (HIL), cujo objetivo é verificar e validar os requisitos de interoperabilidade desses subsistemas, e 6.i) injetar falhas nos subsistemas reais por meio de um Mecanismo Emulador de Falhas (FEM), cujo objetivo é testar a interação dos subsistemas reais em termos de requisitos de robustez especificados.

O subsistema OBC está representado em azul e a carga útil em verde.

Figura 4.3 – Sistemática de testes.



Fonte: Produção do autor.

A sistemática de testes é evolutiva conforme as validações são realizadas permitindo a reutilizando dos resultados de um módulo para o módulo subsequente.

4.4. **Evolução da sistemática de testes**

Antes de realizar os testes de interoperabilidade dos subsistemas comunicantes em nível de aplicação, isto é na camada de mais alto nível do protocolo de comunicação, são realizados testes nas camadas mais baixas como na camada física. Os testes iniciais são realizados para verificar cabeamento, meio de transmissão, pinagem, corrente elétrica, tensão elétrica, entre outros testes para verificar se a conexão física está realizada conforme as especificações. Esses testes são realizados utilizando voltímetro, osciloscópio, entre outros equipamentos necessários. Na sequência os testes realizados são para verificar o protocolo de comunicação, a taxa de transmissão, os endereçamentos dos subsistemas a serem testados, entre outros testes referentes ao protocolo. Nesses testes é utilizado analisador lógico, osciloscópio e alguns outros componentes de software que permitam verificar o protocolo de comunicação. Posteriormente são realizadas as evoluções dos testes em conformidade com a sequencia de atividades descritas em cada um dos seis módulos a seguir:

- **Módulo 1:** i) especificar os requisitos dos subsistemas de software comunicantes no comportamento nominal funcional de interação dos subsistemas sob teste, ii) construção dos modelos desses subsistemas em diagramas de Máquina de Estado Finita utilizando conceito MBT, cujo objetivo é verificar os requisitos de interoperabilidade dos subsistemas por meio da validação dos modelos (*Model-in-the-loop* - MIL), e iii) simular esses requisitos no barramento realizando transformação de modelos e utilizando conceito MDE para gerar e embarcar em placas computacionais programáveis distintas os códigos computacionais para os dois subsistemas modelados, cujo objetivo é verificar e validar os

requisitos de interoperabilidade do comportamento nominal dos subsistemas comunicantes embarcados em ambiente simulado com o uso do barramento de comunicação real (*Software-in-the-loop* - SIL).

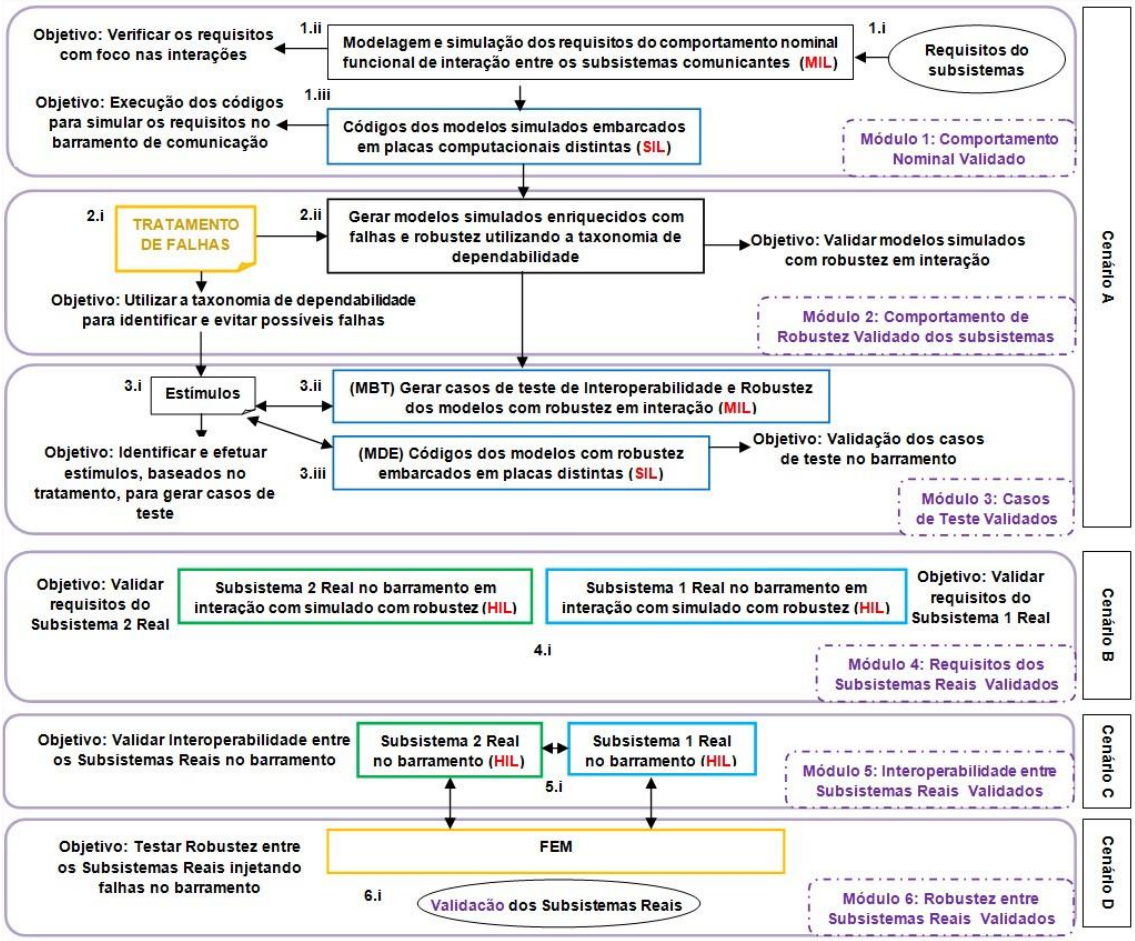
- **Módulo 2**: i) utilizar a taxonomia de dependabilidade de (AVIZIENIS, 2004), cujo objetivo é identificar e classificar possíveis falhas que possam ocorrer na interação entre os subsistemas comunicantes sob teste e relacionar mitigações para evitar as falhas identificadas, e ii) enriquecer os modelos simulados, baseado no tratamento de falhas, com falhas e robustez para gerar modelos mutantes enriquecidos com falhas e modelos com inclusão de robustez. Cada modelo enriquecido com falhas é utilizado para testar a robustez do outro modelo com robustez. O objetivo é testar a robustez do comportamento dos subsistemas simulados.
- **Módulo 3**: i) utilizar as falhas identificadas como estímulos de falhas, cujo objetivo é gerar casos de teste abstratos, ii) utilizar a técnica MBT para gerar os casos de teste de forma automatizada, cujo objetivo é verificar a interoperabilidade e robustez dos modelos com robustez em interação (MIL), e iii) utilizar a técnica MDE para gerar e embarcar de forma automatizada os códigos dos modelos com inclusão de robustez em placas distintas conectadas no barramento (SIL), cujo objetivo é validar os casos de teste no barramento.
- **Módulo 4**: i) conectar um subsistema real no barramento por vez com o subsistema com robustez simulado, o qual faz interação, cujo objetivo é validar os requisitos do software de cada subsistema real separadamente (*Hardware-in-the-loop* - HIL).
- **Módulo 5**: i) conectar os subsistemas reais no barramento, cujo objetivo é verificar e validar a interoperabilidade entre os subsistemas reais (*Hardware-in-the-loop* - HIL).
- **Módulo 6**: i) injetar falhas nos subsistemas reais por meio de um Mecanismo Emulador de Falhas (FEM) conectado entre os subsistemas, cujo objetivo é testar e validar a interação dos

subsistemas reais em termos de requisitos de robustez especificados. Nos módulos anteriores era possível introduzir as falhas nos próprios modelos simulados. Neste módulo, os subsistemas interoperantes são reais e portanto não podem ser modificados para injetar falhas. Assim os estímulos de falhas são injetados pelo FEM.

A sistemática de testes utiliza quatro cenários com arquitetura evolutiva. As evoluções dos módulos 1, 2 e 3 são realizadas em um cenário A, no qual são utilizadas duas placas computacionais programáveis para simular o comportamento nominal funcional do software dos dois subsistemas modelados em interação no barramento. O módulo 4 é realizado em um cenário B, no qual cada subsistema real é testado separadamente no barramento em interoperabilidade com uma placa contendo os códigos computacionais do software do subsistema simulado correspondente ao qual faz interação. Ou seja, um teste é realizado com o OBC real em interoperabilidade com uma carga útil simulada e o outro teste é uma carga útil real com o OBC simulado. O módulo 5 é realizado em um cenário C, no qual os subsistemas reais são testados em interoperabilidade no barramento. O módulo 6 é realizado em um cenário D, no qual é testado a robustez dos subsistemas reais injetando estímulos de falhas no software por meio de uma placa contendo um Mecanismo Emulador de Falhas (FEM) conectado no barramento entre os subsistemas reais.

O fluxo de procedimentos para a realização da abordagem desta tese é ilustrado conforme a Figura 4.4. A identificação da sequência das evoluções das modelagens dos subsistemas a serem testados está representada da seguinte forma: i) a primeira identificação em algarismo numérico representa o módulo, e ii) a segunda identificação em algarismo romano representa o procedimento do módulo.

Figura 4.4 - Evolução da sistemática de testes.



Fonte: Produção do autor.

Pode ser observado que a sistemática de testes evolui utilizando os conceitos MBT e MDE conjuntamente abordando os níveis MIL, SIL e HIL.

4.5. **Tratamento de falhas**

O termo tratamento de falhas adotado no contexto do presente trabalho compreende a identificação de possíveis falhas em ambiente simulado e os procedimentos adotados para evitar que essas falhas ocorram em ambiente real. Os seguintes procedimentos são realizados para tratamento de falhas:

- **Planilha de dependabilidade**: Analisar os requisitos para identificar operações não especificadas que possam causar comportamentos

indesejáveis, e consequentemente falhas. Relacionar na planilha de dependabilidade, desenvolvida nesta tese, as possíveis falhas identificadas nas respectivas classes de falhas baseadas no conceito de dependabilidade. Para cada possível falha identificada é associada no mínimo uma atividade para evitar a respectiva falha (mitigação). Nesta planilha também é possível identificar por dependabilidade, analisando a árvore de dependabilidade, se uma possível falha pode ocasionar outra falha por estarem relacionadas;

- **Tabela FMEA**: Análise causa-efeito para relacionar nesta tabela as possíveis falhas identificadas com os mecanismos e procedimentos que implementam as mitigações necessárias para evitar que essas falhas ocorram.

As mitigações são implementadas nas modelagens nominais realizando transformação de modelos com robustez, cujo objetivo é remover os comportamentos indesejáveis.

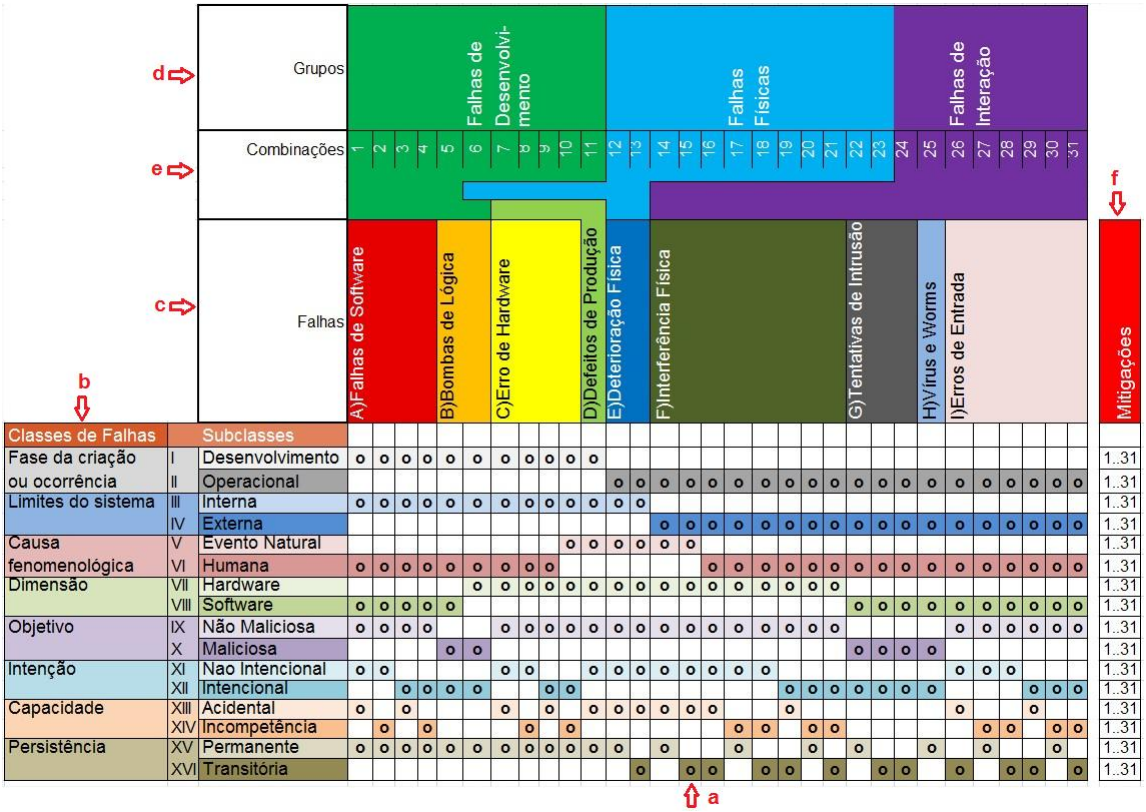
4.5.1. **Planilha de dependabilidade**

Buscando dependabilidade e segurança entre os subsistemas comunicantes de um Cubesat foi utilizada a taxonomia de dependabilidade apresentada em Avizienis et.al. (2004) para o desenvolvimento de uma planilha de dependabilidade.

O objetivo desta planilha é classificar e registrar cada possível falha, identificada na interação entre os subsistemas sob teste, na classificação de falhas da taxonomia de dependabilidade com suas respectivas mitigações. As falhas identificadas devem enriquecer o modelo de um subsistema simulado para testar o outro subsistema simulado enriquecido com as mitigações das respectivas falhas. A árvore de dependabilidade de Avizienis et.al. (2004) também deve ser observada verificando falhas que possam ocorrer por estarem relacionadas com as falhas identificadas.

A planilha representada na Figura 4.5 foi desenvolvida para a abordagem apresentada, baseadas no conceito de dependabilidade, e apresenta as seguintes informações: a) identificação de cada falha que possa ocorrer, b) classes de falhas (cada classe está subdividida em duas subclasses), c) tipos de falhas, d) grupos de falhas, e) trinta e uma combinações de falhas, e f) mitigações que deverão ser realizadas para evitar cada uma das falhas relacionadas nesta planilha. Cada possível falha identificada estará relacionada em sua subclasse de falha (linha) com sua combinação de falha (coluna). No caso de ser identificado mais de uma possível falha em uma determinada subclasse relacionada com a mesma combinação, essas falhas serão identificadas sequencialmente com a numeração .1, .2, e assim sucessivamente. Cada possível falha identificada terá no mínimo uma mitigação relacionada. Desta forma, como existem 31 combinações de falhas, cada subclasse de falha poderá ter no mínimo 31 mitigações correspondentes a cada combinação de falha.

Figura 4.5 - Planilha de dependabilidade.



Fonte: Produção do autor.

As células em branco na planilha são situações aonde não haverá falhas, como no caso de falhas naturais não podem ser classificadas nas classes objetivo, intenção e capacidade.

Neste trabalho o foco está voltado às falhas de: A) Software e F) Interferência Física.

4.5.2. **Tabela FMEA**

Os modos de falha identificados na planilha de dependabilidade deverão ser relacionados em uma tabela Análise dos Modos de Falha e seus Efeitos (*Failure Mode and Effect Analysis* - FMEA) (ECSS-Q-HB-80-03, 2012) customizados para a abordagem apresentada nesta tese. A tabela é inicialmente preenchida com as informações das possíveis falhas relacionadas

na planilha de dependabilidade. Baseado nessas informações, os efeitos de cada falha observados durante os testes são registrados na planilha, bem como os mecanismos identificados que permitirão remover essas falhas (mitigação). As informações que irão compor essa tabela são: a) procedimento identificando possível falha, b) subclasse, combinação e identificação da falha (essas informações estão relacionadas com suas respectivas mitigações na planilha), c) efeito da falha (consequência), d) local afetado pela falha, e e) mecanismo de controle para detecção da falha. A seguir é apresentada a Tabela 4.1 FMEA, na qual são relacionados os mecanismos para identificar as falhas.

Tabela 4.1 - FMEA.

a) Procedimento identificando possível falha	b) Subclasse / Combinação / Identificação da falha	c) Efeito da falha (consequência)	d) Local afetado pela falha	e) Mecanismos de controle para detecção da falha
Procedimento realizado	Subclasse (I até XVI) / Combinação (1 até 31) / Identificação (..n)	Identificar qual consequência a falha irá causar	Identificar qual subsistema ou componente será afetado pela falha	Relacionar qual (is) mecanismo (s) será (ão) realizado (s) para detecção da falha

Fonte: Produção do autor.

Durante a realização dos testes em cada módulo é verificado se os resultados obtidos estão conforme esperado ou se há necessidade de alterações na modelagem de um ou ambos os subsistemas. Não havendo conformidade, novos testes são realizados com o (s) modelo (s) alterado (s). Neste caso novas falhas podem ocorrer e deverão ser catalogadas na planilha de dependabilidade e também na tabela FMEA.

4.6. **Principais diferenças entre a abordagem da tese e as demais técnicas apresentadas na literatura**

Para o desenvolvimento da abordagem apresentada nesta tese diversos artigos foram analisados referentes a tratamento de falhas em missões espaciais de pequenos satélites e software complexo. Evolução nas técnicas utilizadas para tratamento dessas falhas foram observadas, principalmente devido ao crescente uso de modelos no desenvolvimento dessas missões.

As novas tecnologias apresentam ferramentas que auxiliam na modelagem, representação gráfica da missão e facilidade nas alterações dos requisitos no início do ciclo de desenvolvimento do projeto, entre outras facilidades.

A principal diferença da abordagem apresentada nesta tese está no uso de modelos com o propósito de sistematizar a especificação, geração automática e execução de testes de sistemas comunicantes intensivos em software. Para isso a abordagem utiliza de forma combinada as técnicas MBT e MDE orientada aos atributos da interoperabilidade e a dependabilidade para tratamento de falhas na realização de V&V de sistemas comunicantes intensivos em software embarcados em missões espaciais, inicialmente, para nanosatélites.

A Tabela 4.2 a seguir apresenta o resultado da análise de cada artigo relacionado na Tabela 3.1 sob a ótica de abrangência do artigo nas disciplinas relevantes à contribuição da abordagem sistemática proposta nesta tese, possibilitando uma comparação entre eles. O último artigo de número dezessete (18) apresenta a abrangência da abordagem sistemática desenvolvida nesta tese.

A Tabela 4.2 é constituída pelas seguintes colunas:

- Artigo: número do artigo relacionado na Tabela 3.1.
- Disciplina de modelagem: aponta o método utilizado para realizar a modelagem de sistemas empregando conceitos MBT.
- Disciplina MBT com geração de casos de teste: relaciona o método MBT utilizado para gerar casos de teste de forma automatizada.

- Disciplina de categorização do Teste: foram considerados cinco tipos de teste:
  - MIL (*Model-in-the-loop*): este teste possibilita a simulação do modelo comportamental dos subsistemas do projeto.
  - SIL (*Software-in-the-loop*): permite validar os códigos computacionais embarcados em hardware simulado e os casos de teste gerado de forma automatizada.
  - HIL (*Hardware-in-the-loop*): executa testes do subsistema real (hardware) no barramento de comunicação.
  - Inter: realiza testes de interoperabilidade entre os subsistemas comunicantes.
  - Rob: injeta falhas no barramento de comunicação com a intenção de verificar a robustez entre os subsistemas do projeto.
- Disciplina de geração de códigos computacionais de forma automatizada (MDE): aponta o método utilizado para gerar códigos computacionais de forma automatizada a partir de modelos comportamentais do sistema alvo.
- Disciplina de análise de dependabilidade: envolve uso de técnicas e ferramentas de análise de falhas para identificação de mecanismos de mitigação.

Tabela 4.2 - Abrangências dos artigos.

Artigo	Modelagem	Gerar casos de teste (MBT)	Teste					Gerar códigos computacionais de forma automatizada (MDE)	Verificar dependabilidade e mitigação
			MIL	SIL	HIL	Inter	Rob		
1	Combina máquina de estado com outros modelos	Apresenta um algoritmo de geração de teste	Teste com transformação de modelos						
2	Modela dados a partir dos requisitos do software	Gera casos de teste de forma automatizada buscando qualidade de software		Executa códigos gerados				Gera códigos executáveis a partir dos modelos	
3	Utiliza TIO para modelos formais de interoperabilidade	Uso de protótipos de testes representados por propriedades do modelo			Testes de integração de subsistemas espaciais prototipados (modelo de engenharia)	Verifica interoperabilidade entre subsistemas	Verifica robustez de subsistemas com injeção de falhas no canal de comunicação		
4	Máquinas de estado em UML	TRURT	Teste com transformação de modelos						
5	Utiliza UML	Gera casos de teste de forma automatizada	Simula comportamento do software	Executa códigos gerados				Linguagem CHESS Transformação do PIM para PSM	Verificado a partir do IDM
6	Utiliza UML	Focado na segurança	Teste funcional						
7	Utiliza UML e SysML	Teste de V&V	Injeção de falhas pré planejadas						
8	PIM Notação (ASN.1)	Utilizado Simulink e SDL	Simula comportamento do software	Executa códigos gerados	Utilizado no UPMSat-2			PSM Utilizado Ada e TASTE	
9	Definindo padrão de modelos	Utilizando Matlab e Simulink	Códigos reutilizáveis	Executa códigos gerados				Gera códigos executáveis a partir dos modelos	

continua.

Tabela 4.2 - Continuação.

10	Máquina de Estado Finita	Utiliza JPlavisFSM	Verificação por modelagem						
11	Utiliza UML, TML e Eclipse	Dois estágios: primeiro voltado para especialistas em tráfego e outro para projetistas de simulação orientado a plataforma	Simula comporta mento dos dois estágios	Executa códigos gerados				Gera códigos executáveis a partir dos modelos distintos	
12	Utiliza UML	Máquina de Estado Finita	Encontrar defeitos e impactos negativos ao cliente						
13	Utiliza SysML	Modelo de referência para Cubesat utilizando MBSE	Simula comporta mento						
14									Taxonomia de dependabili dade e FMECA
15	Utiliza SBS-Super	Máquina de estado do comportamento formal do software de um satélite	Utiliza JUMBL para gerar casos de teste abstrato e Airbus Defence and Space para gerar casos de teste executáveis						
16	Utiliza UML e SysML	Voltado a projeto e usuário	UX - User eXperience	Executa códigos gerados				Utiliza Papyrus e DOORS	
17	Modela dados a partir dos requisitos do software	Formalismo *timed automata*	Especificação de requisitos de robustez	Valida ção de robus tez		Verifica a integrida de da comuni cação entre os subsiste mas com a injeção de falhas	Injeção de falhas no barramen to de comunica ção		

continua.

Tabela 4.2 - Conclusão.

18	Utiliza Uppaal para modelar o comportamento dos subsistemas em interação	Gerado a partir da Máquina de Estado Finita	Valida requisitos do modelo de interoperabilidade	Valida os subsistemas comunicantes por meio de software embarcado em ambiente simulado	V&V na fase de integração de subsistemas reais	Verifica interoperabilidade entre subsistemas	Verifica robustez entre subsistemas com injeção de falhas	Utiliza Yakindu	Taxonomia de dependabilidade e FMEA

Fonte: Produção do autor.

Um resumo das disciplinas abrangidas nos 17 artigos analisados na Tabela 4.2 é apresentado na Tabela 4.3. O propósito é destacar a abrangência da abordagem sistemática desenvolvida nesta tese, inserida na linha 18 da Tabela 4.2, de forma comparativa aos 17 artigos analisados.

Observa-se que a abordagem de teste desenvolvida na tese contribui de forma sistematizada para o uso combinado das disciplinas consideradas de alta relevância para a integração de subsistemas intensivos em software embarcados em missões de nanosatélites que utilizam o padrão Cubesat. A limitação da abordagem para o domínio de Cubesat deve-se ao fato de a arquitetura do STAE ter sido prototipada para esse domínio.

Tabela 4.3 - Resumo da abrangência dos artigos.

Disciplinas	**Artigos**
Gerar casos de Teste (MBT)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, **18**
Teste de interoperabilidade	3, 17, **18**
Teste robustez	3, 17, **18**
Gerar códigos computacionais de forma automatizada (MDE)	2, 5, 8, 9, 11, 16, **18**
Análise de falhas e mitigações	5, 14, **18**

Fonte: Produção do autor.

5 STAE UTILIZADO NA ABORDAGEM SISTEMÁTICA DE TESTES

Para permitir a execução dos diferentes modelos dos subsistemas comunicantes construídos nos seis módulos que compõem a abordagem sistemática de testes apresentada, um Sistema de Teste com Arquitetura Escalável (STAE) foi desenvolvido. O objetivo do STAE é apoiar a verificação dos requisitos por meio da execução de testes MIL, SIL e HIL e identificar falhas durante a realização de testes.

A arquitetura do STAE é estruturada no EGSE do padrão Cubesat, o que permite estender o barramento de comunicação I2C para uso durante os testes MIL, SIL e HIL, possibilitando a reutilização do STAE em diferentes fases de desenvolvimento de uma mesma missão ou em satélites da mesma família.

5.1. **Arquitetura do STAE**

O STAE apoia o processo de V&V com foco em questões de interoperabilidade entre computador de bordo (OBC – *On-board Computer*) e cargas úteis (PL – *Payload*) de um Cubesat, auxiliadas por mecanismos de injeção de falhas. O objetivo é verificar e validar o comportamento desses subsistemas em situação nominal e de exceção, nos testes MIL, SIL e HIL.

A abordagem busca a análise da comunicação entre os subsistemas do Cubesat por meio do barramento de comunicação permitindo modificações em suas configurações e implementações de acordo com a evolução dos testes, sem grandes alterações e/ou custos. A arquitetura do STAE é simples e de baixo custo. Esse sistema é baseado em placa computacional Arduino *COTS*. Nessas placas são embarcados códigos gerados de forma automatizada por meio da técnica MDE, viabilizando os testes SIL.

O Arduino é um pequeno computador que pode ser programado para processar entradas e saídas entre o dispositivo e os componentes externos conectados a ele. No mercado existem diversos componentes de baixo custo que poderão ser acrescentados nessa arquitetura conforme a necessidade de novos testes, permitindo o reuso do STAE em diferentes fases do

desenvolvimento do satélite. Também existem diversos aplicativos computacionais disponíveis no mercado sem custo que permitem serem aprimorados para auxiliar nos testes a serem realizados, e ferramentas de análise de desempenho (MCROBERTS, 2011).

Existem diversas placas computacionais programáveis no mercado. A seguir é apresentada a Tabela 5.1 contendo a comparação entre algumas placas computacionais programáveis.

Tabela 5.1 - Comparativo entre placas computacionais programáveis.

Comparativo / Placas	Configuração	Custo	Bibliotecas gratuitas desenvolvidas	Compatível com ambiente de geração de códigos (MDE/Eclipse)	Aplicações no mercado usando as referidas placas	Utilização em geral	Vatagem	Desvantagem
Arduino	Mega 2560 R3 (Clock 16MHz, Flash Memory 256KB, SRAM 8kb)	~R$ 40,00	I2C, SQL, Sniffer I2C, RTC, entre outras diversidades	Códigos gerados para Arduino	Diversas aplicações incluindo desenvolvimento de Cubesat	Projetos de automação sendo facilitado por existir diversos projetos já desenvolvidos de livre acesso	Custo, facilidade e tempo de desenvolvimento de aplicações e ambientes de desenvolvimento compatíveis	Dependendo da aplicação tem pouca memória e clock
Raspberry	Pi 3 Modelo B(Clock 1.2GHz, RAM 1GB, SDRAM 256MB, RAM)	~R$ 210,00	I2C, entre outras, porém diversas em desenvolvimento	Códigos gerados em C	Diversas aplicações em desenvolvimento	Projetos que necessitam uma configuração mais robusta e programação específica	Configuração sendo praticamente um micro-computador	Dependendo da aplicação necessitará ser configurado manualmente
Placa de outros fabricantes	Conforme a necessidade será a configuração	> R$ 300,00	Necessitam serem compradas para serem compatíveis com o fabricante	Códigos gerados em C	Aplicações específicas	Projetos específicos necessitando programação específica	Especificidade na aplicação	Tempo e custo no desenvolvimento

Fonte: Produção do autor.

As placas computacionais Arduino *COTS* atendem às necessidades do STAE por serem processadores de baixo custo capazes de embarcar códigos gerados automaticamente e possuírem diversas bibliotecas gratuitas.

A arquitetura do STAE é composta de: i) uma placa Arduino para emular o computador de bordo, ii) uma placa Arduino para emular as cargas úteis, iii)

uma placa Arduino para analisar o tráfego das informações no barramento I2C, iv) uma placa Arduino para injetar falhas com a finalidade de verificar o comportamento dos subsistemas do Cubesat em condições normais e adversas, v) placas dos subsistemas reais do satélite em teste (computador de bordo e cargas úteis), vi) um computador (*Personal Computer* - PC 1) com os ambientes de desenvolvimento necessários para a realização da evolução dos testes do STAE, vii) um computador (PC 2) com o ambiente para realizar as operações do EGSE (Equipamento de Suporte Elétrico de Solo - *Electrical Ground Support Equipment*) fornecido pelo desenvolvedor do satélite em teste, viii) um EGSE, e ix) uma placa denominada *protoboard*, a qual é uma placa com orifícios e conexões condutoras para montagem de circuitos elétricos experimentais.

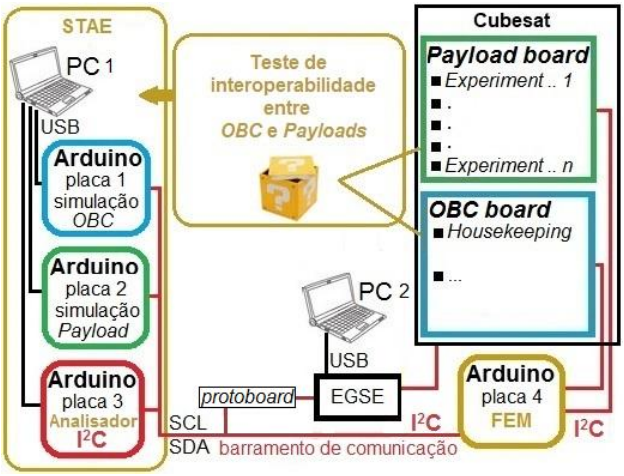
A arquitetura para executar a evolução dos testes inicialmente utiliza as placas computacionais Arduino conectadas, conforme o protocolo de comunicação do satélite em teste, por meio da *protoboard*. As placas Arduino são conectadas ao PC 1 contendo os ambientes para realização e evolução dos testes. Nessas placas são embarcados códigos computacionais simulando o comportamento dos subsistemas reais em interação.

Após a validação do software embarcado nos subsistemas simulados, os subsistemas reais são conectados com os subsistemas simulados por meio da *protoboard*. O objetivo é verificar e validar o comportamento dos subsistemas reais em interação antes de serem integrados ao satélite.

Havendo a validação dos subsistemas reais conforme a evolução dos testes, os subsistemas reais são integrados ao satélite. Após a integração, o EGSE é conectado ao barramento de comunicação do satélite. O EGSE permite executar alguns testes ao qual foi configurado, como inicializar os componentes do satélite e realizar testes de inicialização do computador de bordo e suas cargas úteis. O monitoramento das operações do EGSE é realizado pelo PC 2 ao qual está conectado.

A Figura 5.1 apresenta a arquitetura do STAE. O barramento de comunicação utilizado na abordagem apresentada é o I2C.

Figura 5.1 – Arquitetura do STAE.



Fonte: Produção do autor.

O STAE comporta testar os requisitos lógicos de interoperabilidade entre os subsistemas computador de bordo e cargas úteis do Cubesat. A técnica MDE permite gerar códigos computacionais de forma automatizada a partir da modelagem do comportamento desses subsistemas possibilitando a implementação dos códigos nas placas Arduino, simulando o comportamento dos subsistemas em interação. A técnica Teste Baseado em Modelo (*Model-Based Testing* - MBT) permite gerar os casos de teste baseados nos modelos que são gerados a partir dos requisitos dos software comunicantes dos subsistemas em questão.

Durante a evolução dos testes os diagramas e representações gráficas, como por exemplo, o diagrama de máquina de estado que representa o comportamento esperado dos subsistemas comunicantes, são enriquecidos com requisitos que enriquecem o modelo original de interoperabilidade com funcionalidades dos subsistemas mais próximas da realidade, facilitando o entendimento dos engenheiros envolvidos no projeto. A partir da modelagem da máquina de estado pode ser analisada a simulação do comportamento dos subsistemas do Cubesat de forma isolada e em interoperabilidade. A simulação na fase do projeto preliminar é utilizada para verificar se os requisitos de interoperabilidade foram modelados conforme especificados, possibilitando detectar e remover falhas nos modelos. As alterações identificadas nessa fase

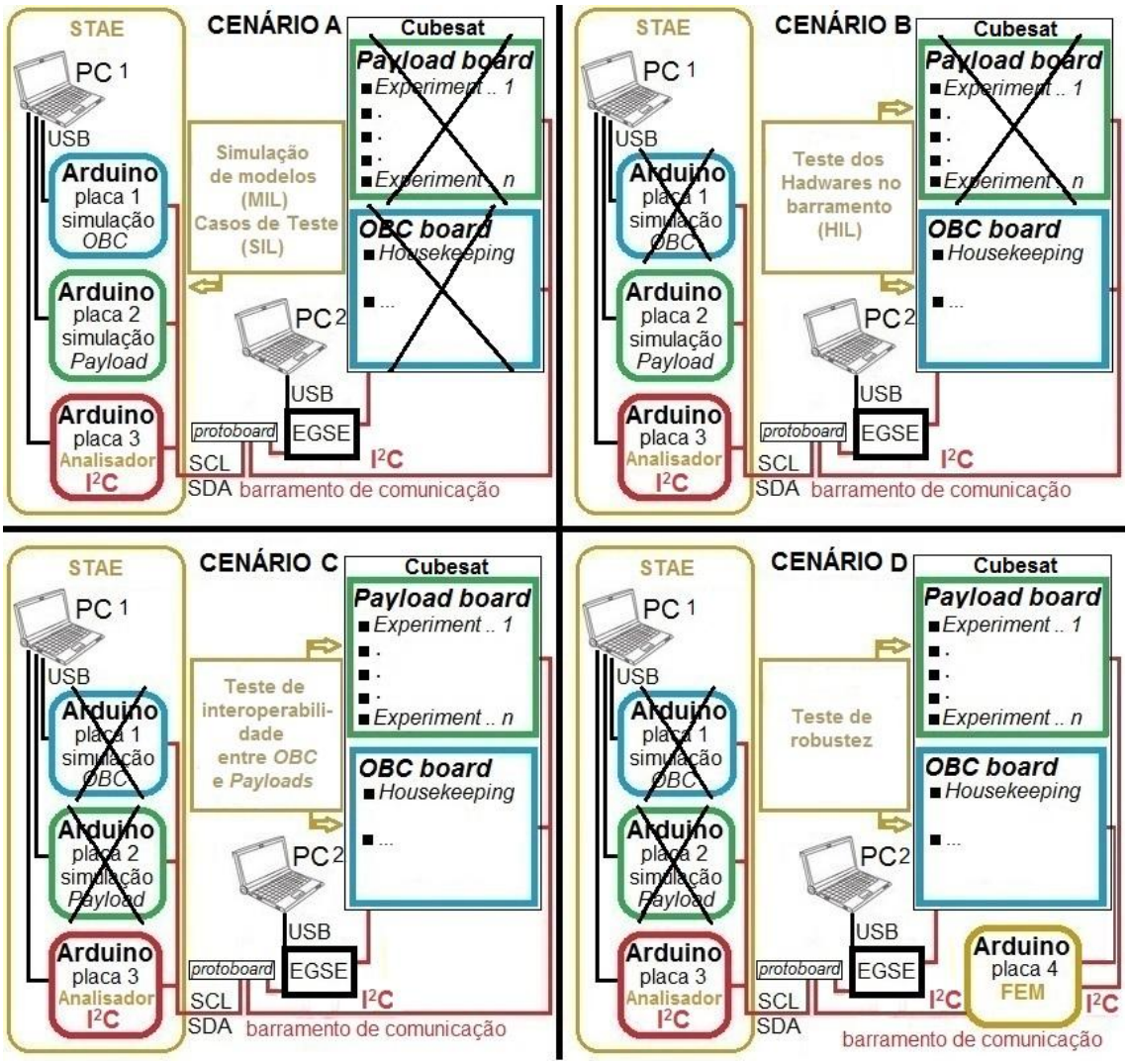
demandam menor tempo para serem corrigidas e menos gastos já que o software ainda não foi desenvolvido (JACKLIN, 2015).

A arquitetura STAE ainda possui a vantagem de acoplar um mecanismo de injeção de falhas (*Fault Emulator Mechanism* - FEM) no barramento de comunicação utilizado pelos subsistemas comunicantes sob teste, conforme proposto pela abordagem InRob (MATTIELLO-FRANCISCO, 2009). O propósito do FEM é injetar falhas de comunicação no barramento durante os testes de interoperabilidade visando a verificação dos requisitos de robustez dos subsistemas comunicantes (BATISTA et al. 2018).

5.1.1. **Cenários do STAE**

A reutilização do STAE ocorre em quatro cenários de uma forma incremental, conforme o diagrama apresentado na Figura 5.2, descritos a seguir.

Figura 5.2 - Cenários de teste baseados no barramento de comunicação I2C.



Fonte: Produção do autor.

- **Cenário A**: O objetivo é simular o comportamento dos subsistemas OBC e carga úteis por um modelo de interoperabilidade que foca a interação entre os subsistemas (*Model-in-the-loop* - MIL). Testes de interoperabilidade permitem validar o modelo. A partir deste modelo são derivados modelos comportamentais dos subsistemas comunicantes (por exemplo do subsistemas OBC e uma carga útil, *Payload* escolhida). Códigos computacionais de cada subsistema são gerados de forma automatizada por ferramentas (MDE). Esses códigos são embarcados e executados nas placas Arduino 1 e 2,

simulando o comportamento dos subsistemas no barramento de comunicação (*Software-in-the-loop* - SIL). Usando a técnica MBT, casos de teste abstratos são automaticamente derivados dos modelos comportamentais. Os casos de testes gerados são transformados em procedimentos de testes executados no Cenário 1. A placa 3 permite analisar o tráfego no barramento de comunicação durante a execução dos casos de testes, possibilitando comparar as informações geradas na interação dos subsistemas com os requisitos especificados. Observando a interoperabilidade, se um subsistema não interagir conforme esperado, evidenciando inconsistência no registro do teste, haverá necessidade de melhorias na especificação dos requisitos e/ou modelagem.

- **Cenário B**: Apenas uma placa Arduino é configurada para simular por software um subsistema, como no Cenário A. O outro Arduino é substituído pelo subsistema real (*Hardware-in-loop* - HIL) que realiza interação com o subsistema simulado correspondente. A interação ocorre via barramento I2C do satélite. Os casos de testes e respectivos procedimentos executados no Cenário A são reutilizados neste Cenário B com o propósito de verificar se subsistema real foi implementado corretamente, conforme modelo comportamental simulado por software no Cenário A. Cada subsistema real é testado no barramento separadamente. Um subsistema real com um simulado compõem o Cenário B e o outro subsistema real com o outro simulado compõem o Cenário B´. Uma questão importante nesse cenário é a reutilização dos conjuntos de testes já especificados no Cenário A, reduzindo o custo e o esforço na fase de aceitação.
- **Cenário C**: Ambos os subsistemas OBC e carga útil são reais no barramento I2C do satélite. A placa Arduino 3 monitora o tráfego por meio do barramento analisando os pacotes de dados e o tempo de transmissão entre esses subsistemas. O objetivo desse cenário é verificar se esses subsistemas interagem conforme os modelos comportamentais desenvolvidos e verificados no Cenário A, sob

condição nominal. Em relação ao conjunto de teste, os casos de teste de interoperabilidade já especificados no Cenário A são reutilizados e testados, reduzindo o custo e esforço na fase de integração.

- **Cenário D**: São analisados os subsistemas OBC e uma carga útil reais usando o FEM (placa 4), o qual é conectado em série no I2C. Após os dois subsistemas terem sido testados sob condição nominal na fase de integração, o objetivo desse cenário é apoiar o teste de robustez. O FEM tem um papel importante nesse cenário porque as injeções de falhas não são intrusivas nos subsistemas sob teste. Esse mecanismo permite verificar a interação dos subsistemas sob condições de exceção especificadas. Em relação aos conjuntos de testes especificados para os testes de robustez, a metodologia InRob é utilizada para auxiliar na geração de casos de teste automático.

5.2. **STAE na evolução da sistemática de testes**

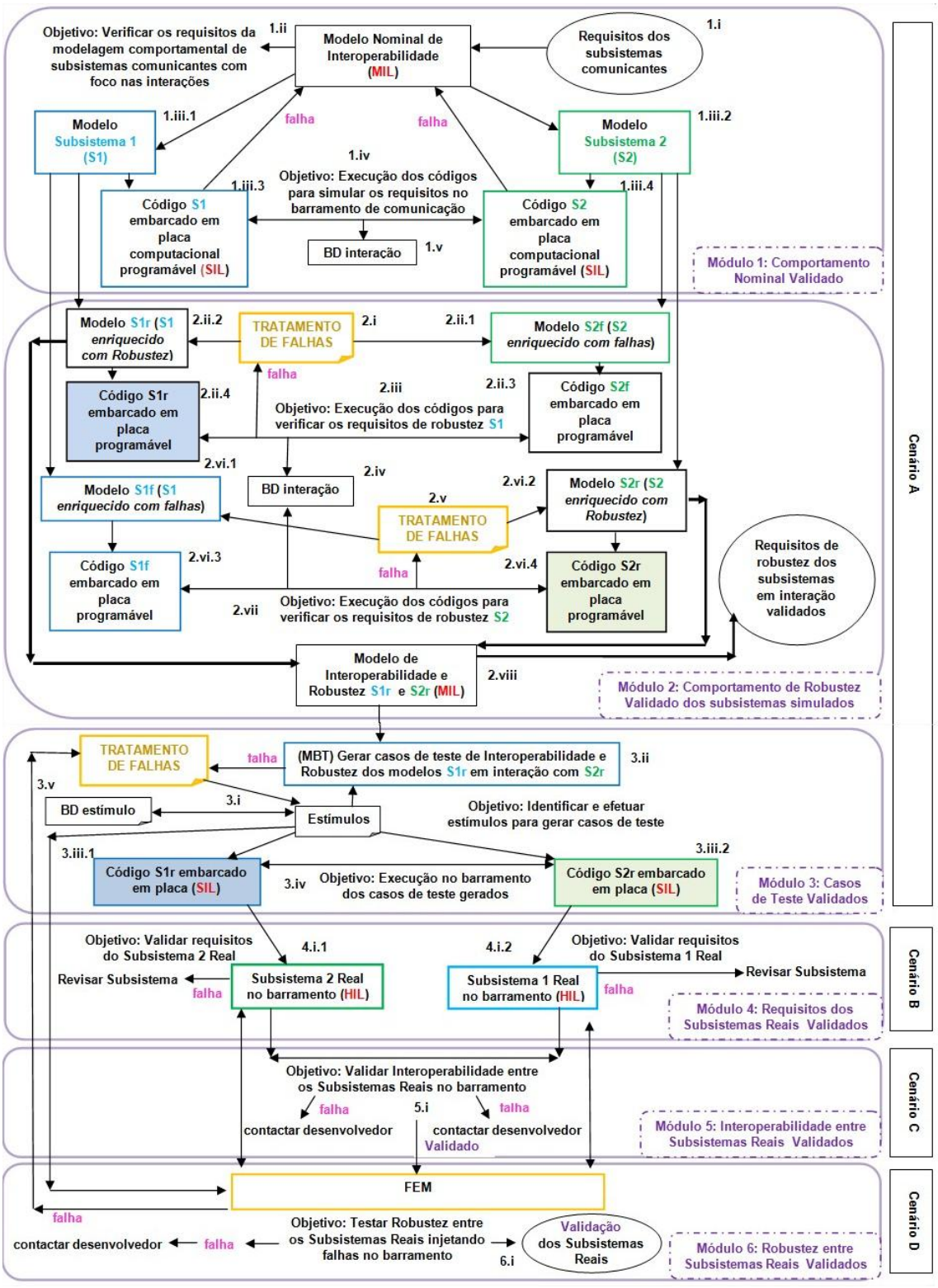
A implementação do STAE abrange as evoluções das modelagens baseadas nos requisitos de comportamentos dos subsistemas intensivos em software do computador de bordo (OBC) e uma carga útil (PL) de um nanosatélite. Essas evoluções envolvem seis módulos partindo do levantamento das especificações dos requisitos de interação entre o OBC e uma PL até a validação desses subsistemas reais no barramento de comunicação. Dentro de cada módulo são realizados procedimentos para a validação do objetivo de cada teste. Conforme a evolução dos testes os procedimentos são subdivididos realizando procedimentos individuais para cada subsistema a ser testado.

5.2.1. **Fluxo detalhado da evolução da sistemática de testes**

O fluxo detalhado dos procedimentos para a realização da sistemática de testes visando tratamento de falhas de interoperabilidade é ilustrado na Figura 5.3. Os procedimentos estão associados aos cenários de uso do STAE e aos seis módulos que compõem a abordagem descritos no capítulo 4.

A identificação da sequencia das evoluções das modelagens dos subsistemas a serem testados está representada da seguinte forma: i) o primeiro número em algarismo numérico identifica o módulo de teste, e ii) o segundo número em algarismo romano identifica o procedimento do módulo, e iii) o terceiro número em algarismo numérico representa a subdivisão do procedimento. Durante os testes os resultados de interação entre os subsistemas são registrados em uma base de dados denominada BD interação e os estímulos de falhas são registrados em outra base de dados denominada BD estímulo. A argumentação para cada procedimento está descrito após a figura apresentada a seguir.

Figura 5.3 - Fluxo detalhado da evolução da sistemática de testes.



Fonte: Produção do autor.

- **Módulo 1**:

  1.i: Analisar as especificações dos requisitos do comportamento nominal dos software dos subsistemas comunicantes a serem testados com foco nas interações. O objetivo é realizar a modelagem da interação desses subsistemas.

  1.ii: Modelar o comportamento nominal dos subsistemas comunicantes, baseado nos requisitos de cada subsistema. O objetivo é validar conformidade da modelagem com os requisitos, de interoperabilidade, entre os subsistemas comunicantes por meio de simulação MIL (*Model-in-the-loop*) utilizando MBT.

  1.iii: Separar a modelagem de cada um dos subsistemas em modelos simulados distintos. O objetivo é gerar os códigos computacionais executáveis SIL (*Software-in-the-loop*) de forma automatizada dos modelos dos subsistemas comunicantes, por meio de ferramentas computacionais MDE. Esses códigos serão embarcados em placas computacionais programáveis distintas existentes na arquitetura STAE com o propósito de verificar aspectos de interoperabilidade dos subsistemas, com o uso do canal de comunicação real (I2C no caso do Cenário A). Neste procedimento permite implementar manualmente os códigos contendo as bibliotecas do protocolo de comunicação entre os subsistemas. Outras bibliotecas necessárias, como acesso a base de dados, também podem ser incluídas.

    1.iii.1: Implementar o modelo de um dos subsistemas verificados em uma ferramenta computacional que utiliza técnica MDE, a qual possibilitará gerar códigos computacionais de forma automatizada desse modelo [Subsistema 1 (S1)].

    1.iii.2: Implementar também o outro modelo na ferramenta MDE [Subsistema 2 (S2)].

    1.iii.3: Gerar os códigos de um dos modelos (S1) e embarcar em uma placa.

1.iii.4: Gerar os códigos do outro modelo (S2) e embarcar em outra placa.

1.iv: O objetivo é executar os códigos que simulam os requisitos dos subsistemas comunicantes para validar o comportamento nominal em interoperabilidade no barramento de comunicação. As placas contendo os códigos são conectadas e testadas por meio do barramento de comunicação (*Software-in-the-loop* - SIL). Ocorrendo falhas, deve ser realizada nova verificação dos requisitos e nova modelagem.

1.v: Registrar em uma Base de Dados, denominada BD interação, as informações geradas e trocadas durante a interação entre os subsistemas modelados. O objetivo é permitir análises dessas informações durante a evolução dos testes.

- **Módulo 2**:

2.i: Analisar os requisitos dos subsistemas comunicantes considerados nos modelos construídos no Módulo 1 utilizando a taxonomia de dependabilidade. O objetivo é identificar possíveis falhas que possam ocorrer durante a interação entre os subsistemas e relacionar mitigações para evitar essas falhas. As falhas identificadas são registradas em uma planilha de dependabilidade. Essa planilha foi desenvolvida para a abordagem sistemática de teste desta tese baseada no conceito de dependabilidade. Os procedimentos para tratamento das possíveis falhas são.

2.i.1: Analisar os requisitos da interação entre os subsistemas comunicantes e identificar possíveis falhas que possam ocorrer. Registrar essas falhas na planilha de dependabilidade com suas respectivas mitigações.

2.i.2: Escolher na planilha possíveis falhas que possam enriquecer o modelo de um dos subsistemas (referido por modelo com falhas) para testar a robustez do outro

subsistema cujo modelo deverá ser enriquecido com as mitigações (modelo robusto).

2.i.3: Identificar na árvore de dependabilidade, por rastreabilidade, se as possíveis falhas relacionadas na planilha de dependabilidade poderão ocasionar outros tipos de falhas. A planilha de dependabilidade possibilita identificar o relacionamento das classes de falhas baseada no conceito de dependabilidade.

2.i.4: Customizar uma tabela de Análise dos Modos de Falha e seus Efeitos (*Failure Mode and Effect Analysis* - FMEA). Nesta tabela as falhas identificadas serão relacionadas com os atributos da planilha de dependabilidade e com os mecanismos e as mitigações para tratamento das possíveis falhas.

2.ii: Enriquecer os modelos nominais dos subsistemas comunicantes. O modelo nominal de um subsistema (S2) será enriquecido com falhas gerando um modelo mutante enriquecido com falhas (S2f) e o outro modelo nominal (S1) será enriquecido com robustez gerando um modelo com robustez (S1r). O objetivo é verificar a robustez do modelo com inclusão de robustez. Posteriormente, gerar códigos computacionais executáveis de forma automatizada dos modelos alterados, por meio de ferramentas MDE. Os códigos de cada modelo são implementados em placas computacionais programáveis distintas. O objetivo é executar os códigos embarcados exercitando a comunicação entre eles via barramento real.

2.ii.1: Enriquecer o modelo S2 com falhas gerando um modelo S2f.

2.ii.2: Enriquecer o modelo S1 com robustez gerando um modelo S1r.

2.ii.3: Gerar os códigos do modelo S1r e embarcar em uma placa computacional.

2.ii.4: Gerar os códigos do modelo S2f e embarcar em outra placa computacional.

2.iii: O objetivo é executar os códigos para verificar e validar os requisitos de robustez do modelo com inclusão de robustez (S1r). Os códigos enriquecidos no modelo S2f permitem estimular situações de falha no modelo S1r, as quais quando não previstas podem levar S1r a um estado errôneo. Neste caso, deve ser verificado se esta falha estava relacionada na planilha e se a respectiva mitigação foi realizada de forma correta. No caso dessa falha ainda não estar relacionada, deve ser registrada na planilha de dependabilidade e na tabela FMEA, juntamente com a respectiva (s) mitigação (ções) enriquecendo o modelo S1r com a robustez necessária.

2.iv: Registrar em uma Base de Dados (BD Interação) as informações geradas e trocadas durante a interação entre os subsistemas enriquecidos. O objetivo é permitir verificar se as informações de interação geradas durante a evolução dos testes estão em conformidade com os requisitos.

2.v: Realizar os procedimentos descritos em 2.i para tratamento de falhas. Porém, neste procedimento o modelo nominal do subsistema (S1) será enriquecido com falhas gerando um modelo mutante enriquecido com falhas (S1f) e no outro modelo nominal (S2) será incluído robustez gerando um modelo com robustez (S2r). O objetivo agora é verificar a robustez do modelo com inclusão de robustez (S2r). Posteriormente, gerar códigos computacionais executáveis de forma automatizada dos modelos enriquecidos, por meio de ferramentas MDE. Os códigos de cada modelo são implementados em placas computacionais programáveis distintas. O objetivo é executar os códigos

embarcados exercitando a comunicação entre eles via barramento.

2.vi: Enriquecer os modelos nominais dos subsistemas comunicantes para testar o modelo com robustez S2r.

2.vi.1: Enriquecer o modelo S1 com falhas gerando um modelo S1f.

2.vi.2: Enriquecer o modelo S2 com robustez gerando um modelo S2r.

2.vi.3: Gerar os códigos do modelo S1f e embarcar em uma placa computacional.

2.vi.4: Gerar os códigos do modelo S2r e embarcar em outra placa computacional.

2.vii: O objetivo é análogo ao procedimento descrito em 2.iii porém, agora o objetivo é executar os códigos para verificar e validar os requisitos de robustez do modelo com robustez (S2r). As informações geradas e trocadas durante a interação entre os subsistemas S1f e S2r são registradas em uma Base de Dados (BD interação), cujo objetivo é permitir análises dessas informações durante a evolução dos testes.

2.viii: Testar a interoperabilidade entre os modelos com robustez dos subsistemas simulados (S1r e S2r). O objetivo é verificar e validar os requisitos de robustez desses modelos em interação (MIL).

- **Módulo 3**:

3.i: Realizar os procedimentos descritos em 2.i para tratamento de falhas. O objetivo é identificar possíveis falhas que possam ocorrer na interação entre os modelos com robustez S1r e S2r.

3.i.1: Identificar na planilha de dependabilidade possíveis falhas que possam estimular o comportamento de robustez nos

modelos com robustez. O objetivo é gerar casos de tese a partir desses estímulos. Esses estímulos permitem gerar casos de teste de situações não previstas nas definições dos requisitos.

3.i.2: Armazenar em uma Base de Dados, denominada BD estímulo, os estímulos identificados. O objetivo é utilizar esses estímulos para testar a robustez dos subsistemas comunicantes.

3.ii: Utilizar os modelos dos subsistemas validados enriquecidos com robustez (S1r e S2r) em uma ferramenta computacional, a qual utiliza a técnica MBT, para gerar casos de teste abstratos de forma automatizada a partir dos estímulos identificados. O objetivo é gerar casos de teste de interoperabilidade e robustez desses modelos em interação. No caso de algum estímulo ocasionar alguma falha não prevista, esse estímulo de falha deve ser registrado na planilha de dependabilidade e na tabela FMEA juntamente com a respectiva (s) mitigação (ções) para enriquecer o modelo com a robustez necessária.

3.iii: Implementa os estímulos nos modelos enriquecidos com robustez validados (S1r e S2r) e gerar os códigos computacionais desses modelos.

3.iii.1: Embarcar os códigos do modelo S1r com estímulos em uma placa computacional programável.

3.iii.2: Embarcar os códigos do modelo S2r com estímulos em outra placa computacional programável.

3.iv: Interconectar no barramento de comunicação as placas contendo os códigos com os estímulos para os modelos S1r e S2r. O objetivo é executar os casos de teste no barramento de comunicação (SIL) e verificar conformidade dos resultados com as especificações dos requisitos. Assim os casos de teste validam os modelos com robustez. No caso de algum estímulo ocasionar

alguma falha não prevista, esse estímulo de falha deve ser registrado na planilha de dependabilidade juntamente com a respectiva (s) mitigação (ções) para enriquecer o modelo com a robustez necessária.

Estes procedimentos compõem o Cenário de teste A, o qual realiza o MIL e SIL.

- **Módulo 4**:

  4.i: O objetivo é testar e validar os requisitos dos subsistemas reais separadamente no barramento (HIL), sendo realizado no Cenário B.

  4.i.1: Testar a interoperabilidade entre a placa contendo os códigos do modelo do subsistema com robustez simulado (S1r) validado com o Subsistema 2 Real. O objetivo é verificar se o Subsistema Real 2 não apresenta falhas de interoperabilidade no barramento (HIL) com o Subsistema S1r simulado. Não apresentando falhas, os requisitos do Subsistema 2 real são validados. Havendo falha, deve ser verificado em qual subsistema apresentou a falha. Se a falha decorreu de um problema no subsistema simulado, deve ser analisado novamente esse subsistema corrigindo a falha detectada. Se a falha foi devido ao subsistema real, deve ser contactado o desenvolvedor.

  4.i.2: Testar a interoperabilidade entre a placa contendo os códigos do modelo do subsistema com robustez simulado (S2r) validado com o Subsistema 1 Real. O objetivo é verificar se o Subsistema Real 1 não apresenta falhas de interoperabilidade no barramento (HIL) com o Subsistema S2r simulado. Não apresentando falhas, os requisitos do Subsistema 1 real são validados. Havendo falha, deve ser verificado em qual subsistema apresentou a falha. Se a falha decorreu de um problema no subsistema simulado, deve ser

analisado novamente esse subsistema corrigindo a falha detectada. Se a falha foi devido ao subsistema real, deve ser contactado o desenvolvedor.

- **Módulo 5**:

  5.i: Após a validação separadamente dos Subsistemas Reais com auxílio dos subsistemas simulados, o objetivo é testar a interoperabilidade entre os subsistemas reais no barramento (HIL), sendo realizado no Cenário C. Havendo falha, deve ser contactado o desenvolvedor do subsistema que apresentou falha. Será verificado se o comportamento dos subsistemas reais em interoperabilidade estão conforme os requisitos especificados.

- **Módulo 6**:

  6.i: O objetivo é validar os requisitos de robustez dos subsistemas reais em interoperabilidade no barramento por meio de um Mecanismo Emulador de Falhas (*Fault Emulator Mechanism* - FEM), sendo realizado no Cenário D. O FEM tem um papel importante porque as injeções dos estímulos de falhas não são intrusivas nos subsistemas sob teste. Esse mecanismo permite a verificação da interação dos subsistemas sob condições de exceção que deverão ser especificadas.

  6.i.1: Caso as falhas injetadas via FEM não levem os subsistemas reais comunicantes a situações errôneas, os subsistemas são considerados validados. Entretanto, se o comportamento de robustez não estiver em conformidade com o esperado frente às falhas injetadas, essas falhas devem ser identificadas na planilha de dependabilidade, e os desenvolvedores dos subsistemas reais devem ser contactados.

  6.i.2: Havendo falha não previstas, os seguintes procedimentos são realizados.

6.i.2.1: Contactar o desenvolvedor do subsistema que apresentou falha.

6.i.2.2: Os estímulos que ocasionaram falhas são relacionados na planilha de dependabilidade e na tabela FMEA com suas respectivas mitigações.

6.i.2.3: Falhas não previstas na planilha necessitam de nova análise dos requisitos dos subsistemas realizando novamente a sistematização de testes para tratamento de falha.

5.3. **Implementação da sistemática de testes**

A implementação da abordagem sistemática de testes é feita aplicando-se os seis módulos descritos na seção 5.2 em dois subsistemas comunicantes de um nanosatélite que utiliza o padrão Cubesat. A título de exemplo, os subsistemas escolhidos são: o computador de bordo (OBC) da plataforma do satélite, responsável pelas funções de gestão dos dados da missão e monitoramento da saúde dos demais subsistemas do nanosatélite; e uma carga útil (PL) científica da missão, ambos subsistemas intensivos em software. As subseções seguintes descrevem os procedimentos da abordagem sistemática considerando a interoperabilidade destes dois subsistemas.

5.3.1. **Módulo 1: Validação do comportamento nominal**

O objetivo inicial é levantar os requisitos de interoperabilidade especificados para a interação do OBC com a PL, e representar o comportamento nominal da interação em modelo de estado. Testes são realizados por meio da execução dos modelos desses subsistemas, adotando-se o conceito "*Model-in-the-loop*" (MIL) utilizando ferramenta computacional baseada em conceito MBT. Uma vez validada a especificação, códigos computacionais de cada subsistema comunicante são automaticamente gerados a partir dos modelos MIL

validados. Os códigos são gerados de forma automatizada utilizando ferramenta MDE e embarcados em placas computacionais programáveis distintas no Sistema de Teste com o propósito de simular o comportamento de interoperabilidade especificado na presença do barramento de comunicação real, adotando-se o conceito de "*Simulation-in-the-loop*" (SIL).

5.3.1.1. Especificações dos requisitos

As especificações dos requisitos é um fator importante para identificar os objetivos que devem ser alcançados. A Engenharia de Requisitos apresenta métodos para identificar os requisitos de um sistema. Um dos métodos é realizar um questionamento buscando identificar os seguintes elementos: i) ator, ii) ação, iii) objetivo, vi) finalidade, v) alvo, vi) lugar de ação, vii) situação, ix) restrição, x) armazenamento, xi) processamento, entre outras perguntas necessárias para identificar os requisitos. As respostas desses questionamentos permitem identificar elementos que não estão claramente definidas nos requisitos. Quanto mais definidos e claros os requisitos mais completas serão as informações que irão definir a modelagem (LOPES, 2015).

Os requisitos de um sistema geralmente sofrem alterações durante o processo de desenvolvimento e testes. Modificar a lógica e a documentação dos requisitos incorporando as alterações geralmente demanda trabalho, tempo e custos. A utilização da MDE para gerar códigos computacionais de forma automatizada facilita e agiliza a remodelagem da máquina de estado conforme as alterações dos requisitos. A MBT permite visualizar de forma rápida a simulação do novo comportamento entre os subsistemas, agilizando a geração automatizada de novos casos de teste e execução de novos testes.

Algumas informações básicas nas atividades (etapas) de interação entre os subsistemas em questão foram pré-definidas: i) OBC deve enviar comandos para a PL solicitando informações de operações e dados armazenados na PL, ii) os dados do experimento armazenados em um buffer da PL devem ser lidos pelo OBC e gravados em sua área da memória para serem transmitidos à estação solo, iii) após a transferência de dados do buffer da PL para a memória

do OBC, a PL deve realizar a aquisição de novos dados, e iv) a troca de informações entre esses dois subsistemas deve ser por meio do barramento de comunicação.

As informações básicas do OBC em interação com uma PL, podem ser preenchidas conforme a Tabela 5.2 para levantamento dos requisitos.

Tabela 5.2 - Requisitos básicos do OBC em interação com uma PL.

Questionamento	Elemento	Requisito
Quem? (*Who*)	1-Ator	OBC
O que? (*What*)	2-Ação (verbo, desejo)	enviar comandos e receber informações
Do que? (*Than*)	3-Objeto (necessidade)	coletar dados do experimento da PL
Por quê? (*Why*)	4-Finalidade	fornecer informações para estação solo
Por quem? (*Who*)	5-Alvo	PL
Aonde? (*Where*)	6-Lugar da ação	área de armazenamento
Quando? (*When*)	7-Situação	a cada 5 milissegundos
De que maneira? (*How*)	8-Restrição	protocolo de comunicação
Limitações físicas (P*hysical limitations*)	9-Armazenamento	
Limitações de desempenho (*Performance limitations*)	10-Processamento	

Fonte: Produção do autor.

As informações básicas de uma carga útil (PL) em interação com o OBC, podem ser preenchidas conforme a Tabela 5.3 para levantamento dos requisitos.

Tabela 5.3 - Requisitos básicos da PL em interação com o OBC.

Questionamento	**Elemento**	**Requisito**
Quem? (*Who*)	1-Ator	PL
O que? (*What*)	2-Ação (verbo, desejo)	receber comandos e enviar informações
Do que? (*Than*)	3-Objeto (necessidade)	coletar dados do experimento e enviar para o OBC
Por quê? (*Why*)	4-Finalidade	esvaziar buffer para aquisição de novos dados
Por quem? (*Who*)	5-Alvo	OBC
Aonde? (*Where*)	6-Lugar da ação	buffer
Quando? (*When*)	7-Situação	sob demanda
De que maneira? (*How*)	8-Restrição	protocolo de comunicação
Limitações físicas (*Physical limitations*)	9-Armazenamento	
Limitações de desempenho (*Performance limitations*)	10-Processamento	

Fonte: Produção do autor.

Após o entendimento dos requisitos básicos desses subsistemas em interação são coletadas mais informações sobre o comportamento dos subsistemas em questão. A Tabela 5.4 apresenta o detalhamento dos requisitos do OBC em interação com uma PL definindo e identificando entre colchetes os estados, os quais serão relacionados com as etapas.

Tabela 5.4 - Requisitos complementares do OBC em interação com uma PL.

	Requisitos
1.	em operação deve realizar serviços como enviar comandos e receber dados de serviço (*housekeeping*), deve avaliar as informações recebidas pela PL para definir qual o próximo comando a ser enviado a esta carga útil, entre outras operações [*IDLE*]
2.	deve enviar comandos (*Command*) para uma PL [*Send*]
3.	deve receber informações (*Information*) de uma PL [*Receive*] e deve definir a próxima ação
4.	deve solicitar o envio dos dados do experimento da PL
5.	deve gravar na memória os dados lidos do experimento [*Write*]

Fonte: Produção do autor.

A Tabela 5.5 apresenta o detalhamento dos requisitos de uma PL em interação com o OBC definindo e identificando entre colchetes os estados, os quais serão relacionados com as etapas.

Tabela 5.5 - Requisitos complementares da PL em interação com o OBC.

	Requisitos
1.	em operação [*ON*] deve aguardar início de comunicação com OBC
2.	deve receber comandos (*Command*) do OBC [*Receive*] e deve avaliar a próxima ação
3.	deve enviar informações (*Information*) solicitadas pelo OBC [*Send*]
4.	deve realizar aquisição dos dados do experimento e deve armazenar no buffer [*Acquisition_Experiment*]
5.	deve ler os dados do experimento [*Read_Experiment*] e deve enviar para o OBC

Fonte: Produção do autor.

Outras informações importantes a serem identificadas são as condições excepcionais entre o OBC a uma PL. Essas condições estão relacionadas na Tabela 5.6.

Tabela 5.6 - Condições excepcionais da interação do OBC com a PL.

Condições	
1.	No caso do OBC receber uma informação não reconhecida deve informar que houve problema (*Problem*) e deve reenviar o primeiro comando
2.	No caso da PL receber um comando não reconhecido, deve informar que houve problema (*Problem*) e deve ler novamente o primeiro comando reenviado pelo OBC
3.	No caso da PL reconhecer o comando, deve informar que está certo (*OK*)
4.	Quando a PL terminar a aquisição do experimento, deve informar que esta pronta (*Ready*) para ler novos comandos e deve enviar os dados para o OBC
5.	Quando a PL terminar de enviar os dados para o OBC, deve informar que está tudo certo (*OK*) e deve retornar a aquisição de novos dados do experimento

Fonte: Produção do autor.

Após levantados os requisitos comportamentais dos subsistemas com foco na interação, são correlacionados os comandos, as informações, as operações, as condições e as etapas da interação entre os subsistemas em teste. Ou seja, se um subsistema (origem) enviar um determinado comando (evento) para outro subsistema (destino), a respectiva operação (ação) deverá ser realizada pelo subsistema destino em determinada situação (condição) conforme a sequência das etapas (estados). A Tabela 5.7 mostra a correlação dessas informações a partir de comandos enviados pelo OBC, devido à função de "mestre" assumida pelo OBC na comunicação mestre-escravo.

Tabela 5.7 – Operações, condições e etapas do OBC interagindo com a PL.

OBC comandos	**Operação (ação)**	**Etapa (*Step*) OBC**	**Condição**	**PL informações**	**Operação (ação)**	**Etapa (*Step*) PL**
	Housekeeping, comandos, entre outras operações	1 *IDLE*			Operando	1 *ON*
Command	Enviar primeiro comando a PL	2 *Send*				2 *Receive*
	Receber informação da PL	3.1 *Receive*	PL reconheceu o comando do OBC	*OK*	Informar que o comando recebido está certo (OK)	3.1 *Send*
		Ou 3.2 *Receive*	# PL não reconheceu o comando do OBC. # PL retornará a ler o primeiro comando. # OBC retornará a enviar primeiro comando. # No caso do OBC não reconhecer o comando, retornará a enviar primeiro comando.	*Problem*	Ou informar que houve problema no comando recebido (*Problem*)	Ou 3.2 *Send*
					Aquisição dos dados do experimento. A PL poderá ter diversos experimentos a serem realizados (n)	4..n *Acquisition_ Experiment*
	Verificar próximo comando a ser enviado	4 *IDLE*				
Command...n	Enviar comandos subsequentes a PL. O OBC poderá enviar diversos comandos (n)	5..n *Send*				5 *Receive*
	Receber informação da PL	6.1..n *Receive*	PL pronta para enviar dados	*Ready*	Informar que terminou aquisição de dados (*Ready*) A PL poderá enviar diversas informações (n)	6.1..n *Send*

continua.

Tabela 5.7 – Conclusão.

		Ou 6.2..n *Receive*	# PL não reconheceu o comando do OBC. # PL retornará a ler o primeiro comando. # OBC retornará a enviar primeiro comando. # No caso do OBC não reconhecer o comando, retornará a enviar primeiro comando.	*Problem*	Ou informar que houve problema no comando recebido (*Problem*)	Ou 6.2..n *Send*
					Leitura dos dados do experimento. Próximo *Step* (n+1)	n+1 *Read_ Experiment*
	Receber dados do experimento. Próximo *Step* (n+1)	n+1 Receive		*Experiment Data*	Enviar os dados do experimento	n+2 *Send*
	Gravar dados do experimento	n+2 *Write*				
	Verificar se PL terminou de enviar dados para solicitar novos dados do experimento	3.1	PL informará que está tudo certo para aquisição de novos dados	*OK*	Término do envio dos dados (*OK*)	3.1

Fonte: Produção do autor.

Realizado os levantamentos das operações e das etapas, elabora-se um diagrama que permite a visualização gráfica das atividades (etapas) dos subsistemas OBC e PL em interação.

5.3.1.2. Diagrama de atividades

O diagrama de atividades (PIMENTEL, 2015) é utilizado para a representação das sequências das etapas (*step*) dos subsistemas em interação de forma gráfica, o qual possibilita uma visualização e identificação das etapas realizadas de cada subsistema destacando a interação entre eles.

Nos diagramas desenvolvidos neste trabalho as etapas correspondem às atividades as quais são representadas por retângulos ou quadrados e as

transições entre as atividades são representadas por setas. As atividades estão associadas aos estados dos modelos a serem realizados.

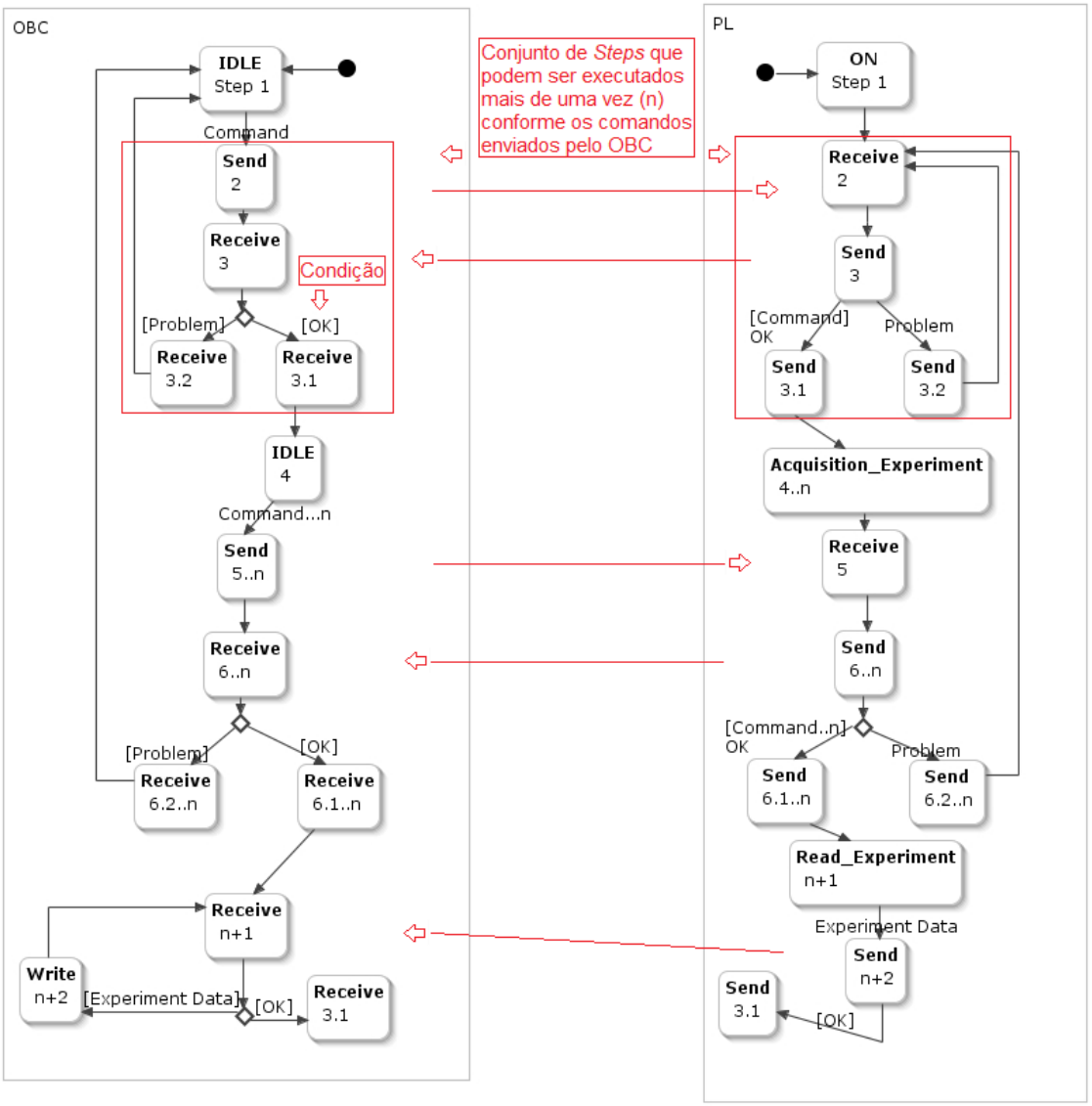
No diagrama de atividades do OBC, antes de cada atividade *Send* é definido na atividade *IDLE* qual comando será enviado para a PL. Após cada atividade *Receive* é verificada qual condição foi satisfeita (entre colchetes [ ]) definindo qual será a próxima etapa (*Step*) a ser realizada. Assim a condição caracteriza o evento de transição entre estados.

Algumas etapas (*Steps*) podem ser executadas mais de uma vez dependendo dos comandos que o OBC irá enviar para uma determinada carga útil, como demonstrado no diagrama da Figura 5.4 pelo retângulo em vermelho.

No diagrama de atividades da PL, antes da atividade *Send* é verificada qual condição foi satisfeita para enviar a próxima informação para o OBC.

Os diagramas desenvolvidos neste trabalho visam a interoperabilidade entre os subsistemas em questão. A Figura 5.4 apresenta os diagramas de atividades do OBC e da PL em interação. As setas longas (vermelhas) entre os atores OBC e PL indicam o sincronismo da interação entre eles, isto é em qual *Step* um subsistema envia ou recebe (*Send ou Receive*) informação para o outro nos momentos de interação.

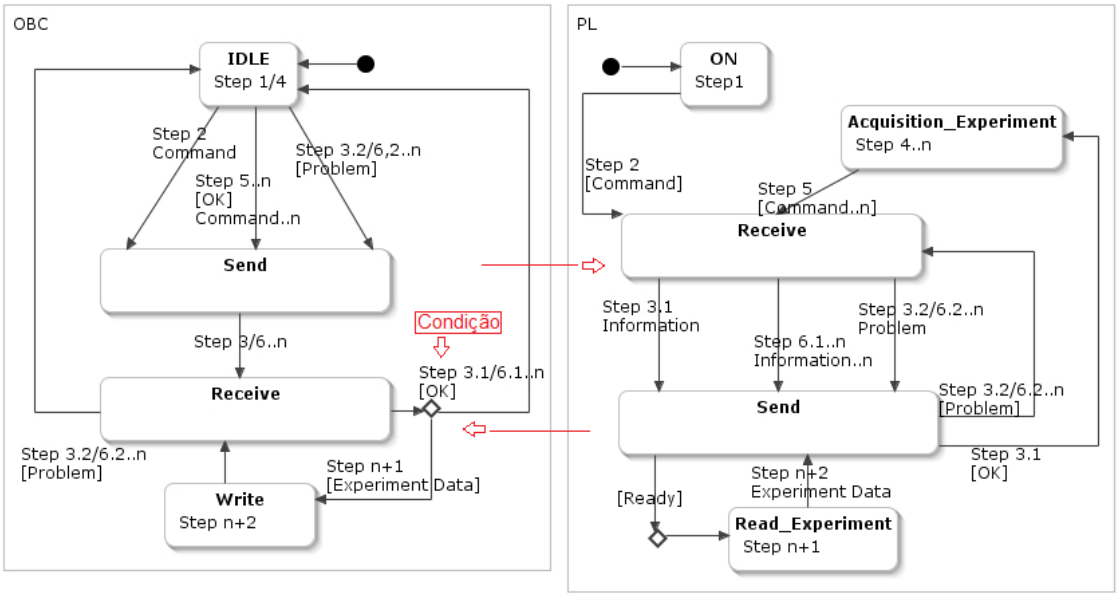
Figura 5.4 – Diagramas de atividades do OBC e de uma PL interagindo.



Fonte: Produção do autor.

Nos diagramas dos subsistemas OBC e PL as atividades *Send* são as etapas em que esses subsistemas enviam informações. As atividades *Receive* são as etapas em que esses subsistemas recebem informações. Tendo como foco a interação entre o OBC e uma PL, este diagrama pode ser reestruturado sendo representado apenas uma atividade de *Send* e uma de *Receive* em cada diagrama identificando as situações de entradas e saídas como representado na Figura 5.5.

Figura 5.5 – Diagramas de atividades reestruturados.



Fonte: Produção do autor.

Os diagramas reestruturados são utilizados como base para a realização das modelagens dos subsistemas em interação.

5.3.1.3. Modelagem

O objetivo inicial da modelagem é verificar por simulação os requisitos da modelagem comportamental de duas Máquinas de Estado comunicantes com foco nas interações (MIL). As máquinas modeladas são dois subsistemas, sendo um o computador de bordo denominado (OBC – *On-board Computer*) e o outro uma carga útil denominada (PL - *Payload*) de um Cubesat.

Observando os diagramas de atividades realizados anteriormente é realizada a modelagem dessas Máquinas de Estado. As atividades nos diagramas de atividades são os estados na modelagem. Entretanto, as definições até o momento definem os eventos e as condições nas transições das máquinas de estado. Porém, precisam ser acrescentadas nas transições algumas variáveis, tais como contadores, área de armazenamento, entre outras informações necessárias para executar as ações que irão definir a sequência das etapas

(*Steps*) (estados) a serem alcançados. Essas informações complementares possibilitam a modelagem dos subsistemas em questão baseados nos diagrama de atividades desenvolvidos até o momento.

A Tabela 5.8 apresenta eventos, ações, condições e variáveis (em negrito as variáveis de controle) definidas para modelagem do OBC em interação com a PL simulando o comportamento conforme a sequência de etapas a serem alcançadas. O OBC em interação com a PL irá enviar os dados por meio de uma variável denominada *SendOBC* e a PL por meio da *SendPL*.

Tabela 5.8 - Estados, eventos, ações e condições do OBC.

Estados **OBC**	Eventos	Ações	Descrição	Condições	Descrição
IDLE	*Step* 1/4	*Definir comando Housekeeping* ou outros	Ações do OBC		
	OK Step 3.1/6.2..n			SendPL = OK ou **WriteOBC**=1	PL recebeu comando de forma certa ou OBC gravou os dados do experimento (WriteOBC=1)
	Problem Step 3.2/6.2..n			SendPL não= 1 e não= 2 e **WriteOBC** não =1	OBC recebeu informação que a PL teve problema ao receber comando do OBC e OBC não reconheceu informação da PL e OBC não gravou dados do experimento
Send	SendPL *Step 2*	SendOBC= *Command*	OBC enviará primeiro comando.	SendPL=0	PL em operação para receber comando
	Command..n Step 5..n	*SendOBC= Command..n* **WriteOBC**=0	Enviar comando (s). Inicia variável indicadora que OBC gravou os dados	SendPL= *OK*	PL recebeu comando de forma certa
	Problem Step 3.2/6.2..n	SendOBC=0	OBC envia informação de problema. Não reconheceu informação recebida	SendPL não= 0 e não=*OK* e não= *Ready*	PL não recebeu comando de forma certa (*OK*) e não está pronta para enviar dados (*Ready*)]
Receive	SendPL *Step* 3/6..n		OBC recebe informação da PL		
	Step n+2	**WriteOBC**=1	Informa que OBC gravou os dados do experimento		
Write	SendPL *Step* n+1	**OBCData**= SendPL	Área de armazenamento do OBC recebe e grava os dados do experimento	SendPL =*Ready*	PL pronta para enviar os dados do experimento

Fonte: Produção do autor.

A Tabela 5.9 apresenta eventos, ações, condições e variáveis (em negrito as variáveis de controle) que irão compor a modelagem da PL interagindo com o OBC simulando o comportamento conforme as etapas dos requisitos a serem alcançadas.

Tabela 5.9 - Estados, eventos, ações e condições da PL.

Estados **PL**	Eventos	Ações	Descrição	Condições	Descrição
ON	*Step* 1	SendPL=0	PL em operação para receber comando		
Receive	*Command Step* 2		PL recebe primeiro comando do OBC		
	Command_n Step 5		PL recebe comandos subsequentes do OBC [OBC poderá enviar diversos comandos (n)]		
	Problem Step 3.2/6.2..n			SendPL=0	PL identifica problema no recebimento do comando
Send	*Command Step* 3.1	SendPL=*OK* (*Information*)	PL informa recebimento do comando de forma certa (*OK*)	SendOBC= *Command*	OBC enviou primeiro comando
	Command..n Step 6.1..n	SendPL=*Ready* (*Information..n*) **SendDataOBC** =0	PL está pronta para enviar dados do experimento (*Ready*). Inicializa variável indicadora que dados foram enviados para o OBC	SendOBC= *Command..n*	OBC enviou comando subsequente
	Problem Step 3.2/6.2..n	SendPL= *Problem*	PL informar problema no recebimento do comando	SendOBC não= *Command* e não = *Command_n*	PL não reconheceu comando recebido
	Experiment Data Step n+2	SendPL= **PLBuffer** **SendDataOBC** =1	PL envia para o OBC dados do experimento armazenados no seu Buffer (PLBuffer). Inicializa variável indicando que os dados foram enviados para o OBC		
Acquisition _Experiment	*Information Step* 3.1	SendPL=*OK*	PL informa recebimento de comando de forma certa (*OK*)	SendPL=*OK* ou **SendDataOBC**=1	PL recebeu comando de forma certa ou PL enviou dados para o OBC
Read_Experiment	*Command_n Step* n+1		OBC enviou comando solicitando envio dos dados do experimento	SendPL=*Ready* e SendOBC não=0	PL pronta para ler dados do experimento e enviar para o OBC (*Ready*) e OBC enviou comando

Fonte: Produção do autor.

Para realizar a modelagem e execução dos modelos comportamentais dos subsistemas em interação é utilizada uma ferramenta computacional de modelagem MBT.

5.3.1.4. Ferramenta computacional de modelagem

Diversas ferramentas de modelagem permitem simular o comportamento na interação entre os subsistemas para fins de verificação dos requisitos no início do ciclo de desenvolvimento do sistema. A ferramenta de modelagem selecionada para a implementação da técnica MBT na sistematização dos testes foi a Uppaal versão 4.1.19 - rev. 5648 (UPPAAL, 2009).

A sincronização entre as máquinas de estado na Uppaal é realizada utilizando os seguintes indicadores: i) identificador de canal seguido do símbolo exclamação (!) é definido como *output* para escrever informação no canal, ii) identificador de canal seguido do símbolo interrogação (?) é definido como *input* para ler informação no canal. Ações de escrita e leitura no mesmo canal geram a sincronização.

Na modelagem dos subsistemas OBC em interação com a PL o sincronismo é realizado por meio dos identificadores SendfromOBC! para o OBC enviar comandos à PL e SendfromPL? para receber informações. Assim como, SendfromPL! para a PL enviar informações e SendfromOBC? para receber.

Os estados das máquinas do OBC são denotados por OBC acrescido dos estados, por exemplo, *OBC_Send*, para diferenciar dos estados da máquina da PL, cujos estados são denotados por PL acrescidos do estado propriamente dito.

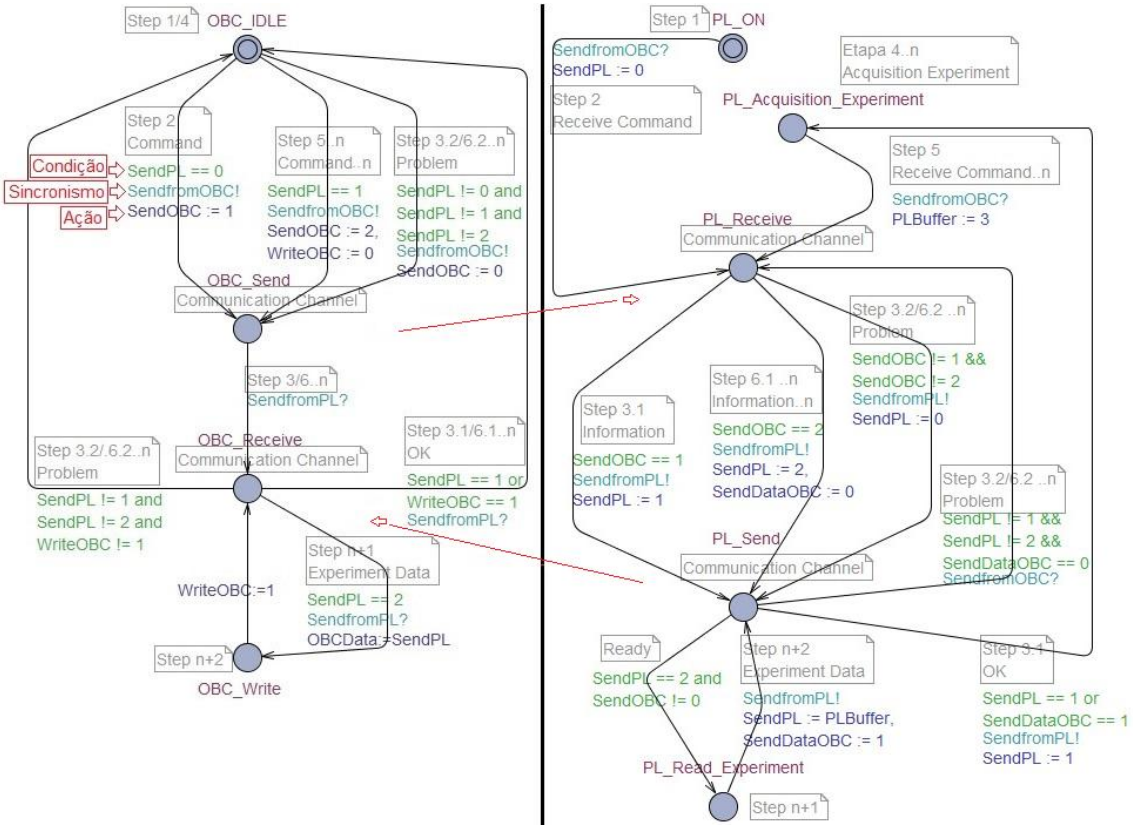
Utilizando o diagrama de atividades, a tabela de estados e eventos, as informações complementares para as transições e indicadores de sincronismo são realizadas as modelagens do OBC e da PL na ferramenta Uppaal. O objetivo é verificar os requisitos da modelagem comportamental dos subsistemas OBC e PL com foco nas interações executada por simulação (MIL).

Nas modelagens os comandos e informações são alterados para representações numéricas. Essas alterações são apenas para haver comparações numéricas e não alfanuméricas. Sendo assim, no modelo OBC: i) *Command*=1, ii) *Command..n*=2, e iii) *Problem*=0. Na modelagem da PL: i)

Information=OK=1, ii) *Information..n=Ready*=2, iii) *Problem*=0, e iv) *Experiment Data*=3.

Na ferramenta Uppaal as condições são identificadas na cor verde clara, o sincronismo na cor azul claro e as ações na cor azul escuro, conforme identificado com setas na Figura 5.6.

Figura 5.6 – Modelagem das máquinas de estado do OBC (esquerda) e de uma PL (direita).

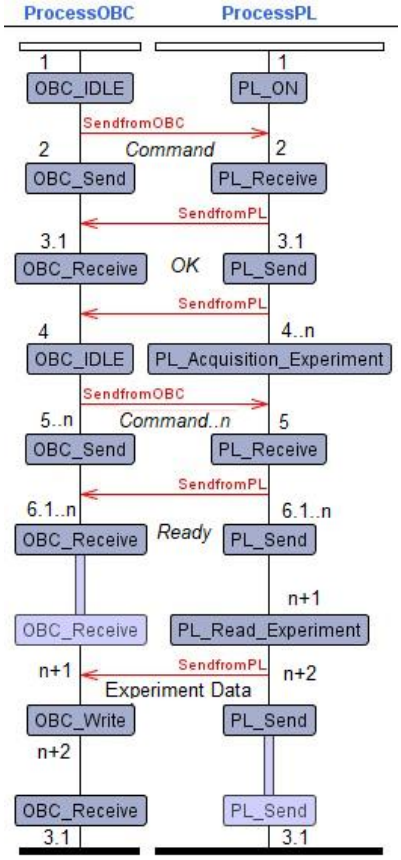


Fonte: Produção do autor.

A ferramenta Uppaal permite simular o comportamento das máquinas de estado dos subsistemas em interação a partir da modelagem. A representação gráfica e a simulação possibilitam verificar nesta fase inicial de desenvolvimento do projeto se os fluxos das informações e os requisitos estão sendo alcançados conforme as especificações.

Essa ferramenta também permite gerar um diagrama de sequência a partir da simulação (*Simulator*) do comportamento das máquinas de estado apresentando a mudança de um estado para outro assim como a interoperabilidade entre os subsistemas comunicantes. A Figura 5.7 contém o diagrama de sequência apresentando as interações entre o OBC e a PL representadas por setas. As informações trocadas entre esses subsistemas e o número das etapas (*Steps*) foram acrescentadas manualmente para uma melhor visualização e entendimento do que está ocorrendo em cada troca de informação.

Figura 5.7 – Diagrama de sequência do OBC em interoperabilidade com uma PL.



Fonte: Produção do autor.

Comparando este diagrama de sequencia com o diagrama de atividades da Figura 5.4 desenvolvido com base nos requisitos, pode ser observado que a

sequencia das etapas que devem ser alcançadas durante a interação do OBC com a PL estão sendo realizadas na mesma sequência nos dois diagramas. Desta forma foi verificado que os subsistemas OBC e PL modelados em interação estão atendendo aos requisitos comportamental nominal desses subsistemas comunicantes conforme definidos, validando assim a modelagem (MIL).

A simulação também possibilita estimular eventos de forma manual alterando e testando o fluxo das informações. Esses estímulos permitem verificar se há necessidade de alterações nas definições dos requisitos e/ou modelos dos subsistemas comunicantes.

5.3.1.5. Transformação de modelos

A utilização de recursos da Engenharia Dirigida por Modelos (MDE) permite a transformação de modelos de sistemas em códigos computacionais executáveis de forma automatizada possibilitando testar os códigos em ambiente de comunicação real. Um ambiente computacional que permite a geração desses códigos é o Eclipse (ECLIPSE, 2016). O Eclipse possui ferramentas computacionais que permitem gerar códigos em diversas linguagens. A ferramenta Yakindu (2017) gera códigos computacionais de forma automatizada em linguagem C e códigos para placas computacionais Arduino, as quais são utilizadas no STAE.

Na abordagem apresentada é utilizada a ferramenta Yakindu para gerar os códigos computacionais que serão embarcados em placas computacionais Arduino que ao serem executados simulam o comportamento funcional dos subsistemas comunicantes em interação no barramento de comunicação (SIL).

A versão Eclipse instalada para realizar a transformação de modelo é Eclipse IDE for C/C++ Developers, Version: Neon.1 Release (4.6.1). Dentro dessa plataforma foram instaladas as seguintes ferramentas computacionais para atenderem ao desenvolvimento do STAE: i) *Eclipse C++ IDE for Arduino 2.0*, ii) *Arduino AVR Boards 1.6.15*, iii) *CDT releases neon 9.0*, iv) *C++ Development*

Tools SDK 9.1.0, versão V2, v) *Yakindu SCT 2.9.3*, e vi) *Yakindu Statechart Tools for Arduino 0.3.0*.

Nessa transformação de modelos são utilizadas as representações dos estados (step), transições, ações e condições na Uppaal para realizar as modelagens do OBC e da PL na Yakindu.

As alterações realizadas na modelagem na Yakindu comparada a Uppaal são: i) retirada do sincronismo definido na Uppaal, ii) utilização do recurso escolha (*choice*), iii) a condição está representada entre colchetes [ ], e iv) a ação está após o comando *always*.

O recurso *choice* é representado pela figura geométrica em formato de um losango. Nessa representação antes do losango é especificada a condição e depois a ação. O *choice* requer uma alternativa de saída *default* caso a ação especificada não seja atendida. Nessa alternativa *default* já existe a possibilidade de serem incluídos alguns tratamentos de falhas nas ações.

Outro diferencial da modelagem na Yakindu comparada a Uppaal apresentado neste trabalho é referente ao sincronismo. As modelagens realizadas na abordagem desta tese utilizando a ferramenta Yakindu não apresentam indicadores de sincronismo. Uma máquina de estado realiza seus eventos e ações sem a necessidade de aguardar informação recebida da outra máquina de estado, aproximando-se assim do comportamento real dos subsistemas comunicantes.

Na modelagem na Yakindu as variáveis denominadas *SendOBC* e *SendPL* foram definidas para realizar a troca de informações na interoperabilidade entre os subsistemas OBC e PL.

5.3.1.6. Geração de modelos

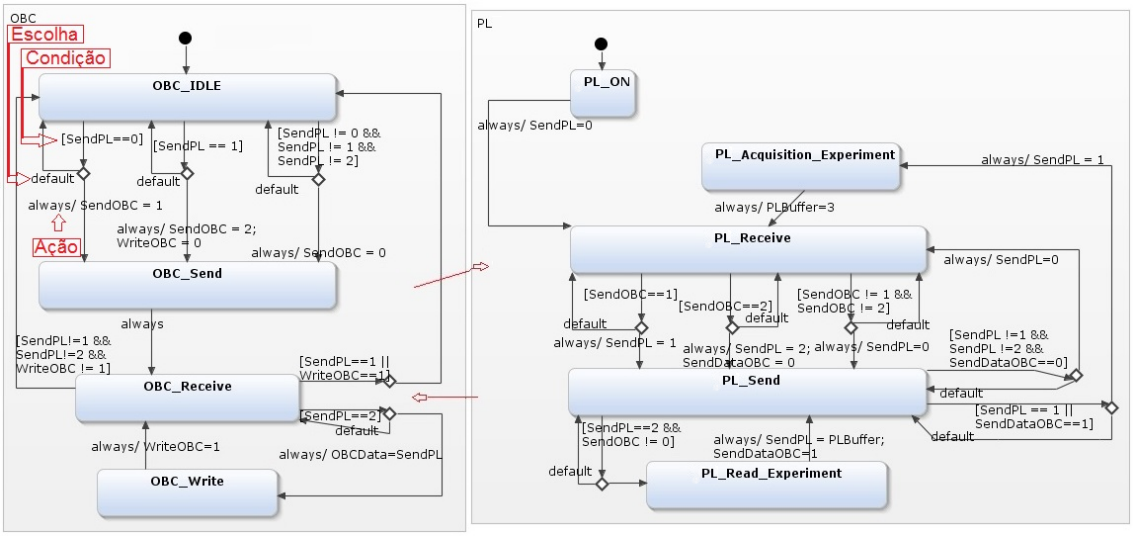
Para modelar os subsistemas no ambiente Eclipse/Yakindu são realizados os seguintes procedimentos: i) conectar a placa Arduino a porta USB do computador contendo o ambiente de desenvolvimento Yakindu, ii) no *menu* superior desse ambiente selecionar *New Launch Target*, iii) definir nome para a

placa Arduino, iv) selecionar porta de conexão com a placa, v) selecionar o modelo da placa, vi) selecionar *AVR ISP*, vii) selecionar no *menu* superior do ambiente (Novo/Projeto/*Yakindu Statechart Tools for Arduino Arduino*/*Project with Yakindu Statechart*), viii) definir o nome do projeto (nome do modelo do subsistema), ix) definir o ciclo de período (neste trabalho é definido 10 ms para utilizar placas Arduino), x) selecionar a arquitetura (modelo da placa Arduino), e xi) selecionar o contador (neste trabalho é utilizado o contador 1 de 16 bits). Neste momento é gerada uma pasta dentro do projeto identificado com o nome *model* e dentro dessa pasta são criados dois arquivos. Esses arquivos são: i) um arquivo com extensão .sct, o qual contém modelo evolutivo a partir da modelagem na Uppaal, e ii) um arquivo com extensão .sgen, o qual permite gerar os códigos do modelo de forma automatizada selecionando a opção *Generate Code Artifacts*.

Inicialmente para testar a modelagem dos subsistemas em interação é realizada a modelagem do OBC e da PL em um único projeto. Posteriormente esses subsistemas são modelados em projetos diferentes, cuja finalidade é gerar códigos para cada modelo de cada subsistema permitindo embarcar esses códigos em placas Arduino distintas. O objetivo é cada placa simular o comportamento de um subsistema real.

A Figura 5.8 apresenta a máquina de estado do OBC à esquerda. Os eventos recebidos pelos estados são por meio da variável *SendOBC* (comando a ser enviado pelo OBC) ou *SendPL* (informação recebida pela PL). A máquina de estado da PL está à direita. Os eventos recebidos pelos estados são por meio da variável *SendPL* (informação a ser enviada pela PL) ou *SendOBC* (informação recebida pelo OBC).

Figura 5.8 – Máquinas de estado do OBC (esquerda) e da PL (direita).



Fonte: Produção do autor.

A ferramenta Yakindu nesta etapa também permite simular o comportamento das máquinas de estado dos subsistemas comunicantes possibilitando verificar se os fluxos das informações e os requisitos estão sendo atendidos conforme as especificações. Para realizar a simulação clicar com o botão do lado direito do *mouse* sobre o arquivo .sct, selecionar *Runs As* e optar por *Statechart Simulation*. Os valores das variáveis aparecem na janela de simulação.

5.3.1.7. Geração dos códigos computacionais dos modelos

Utilizando a ferramenta Yakindu, a qual utiliza conceito MDE, são gerados os códigos computacionais executáveis de cada modelo de forma automatizada. O objetivo é embarcar os códigos em placas computacionais distintas para cada modelo e testar esses códigos em ambiente de comunicação real.

Para a geração dos códigos separadamente dos subsistemas modelados no ambiente de desenvolvimento Eclipse/Yakindu são realizados os seguintes procedimentos: i) conectar as placas Arduino ao computador, contendo o ambiente de desenvolvimento Eclipse/Yakindu, por meio dos cabos específicos dessas placas utilizando as portas de comunicação apropriadas, ii) abrir dois ambientes de desenvolvimento Yakindu, iii) utilizar um ambiente para

configurar uma placa conectada e o outro ambiente para a outra placa, vi) configurar um ambiente com uma modelagem de um subsistema a ser testado e o outro ambiente com a outra modelagem, v) gerar os códigos computacionais de forma automatizadas de cada modelagem em seu respectivo ambiente selecionando a opção *Generate Code Artifacts* no arquivo com extensão .sgen criado a partir da definição do projeto do modelo do subsistema a ser testado, vi) implementar manualmente nos códigos gerados de cada modelagem as bibliotecas necessárias para haver interoperabilidade entre os subsistemas comunicantes (protocolo de comunicação), e vii) implementar manualmente outras bibliotecas necessárias (banco de dados, entre outras).

A Yakindu gera no projeto da modelagem em desenvolvimento uma pasta denominada src e outra pasta src-gen. Na pasta src é gerado um arquivo denominado *NomeProjetoConnector.cpp*. Nesse arquivo comandos podem ser incluídos para definição da porta serial do Arduino e também controles de entrada e saída. Na pasta src-gen é gerado um arquivo denominado *NomeProjeto.cpp*. Nesse arquivo as bibliotecas e alguns códigos do protocolo de comunicação podem ser implementados de forma manual. Outras bibliotecas como acesso a bando de dados, envio de data e hora também podem ser implementadas. A Tabela 5.10 apresenta algumas linhas de comandos referentes ao protocolo I2C implementadas manualmente nos arquivos OBC.cpp e PL.cpp. Cada linha possui um comentário após duas barras (//) descrevendo o que será realizado.

Tabela 5.10 - Arquivo OBC.cpp (esquerda) Arquivo PL.cpp (direita) contendo comandos do protocolo I2C.

Arquivo OBC.cpp	Arquivo PL.cpp

#include "Wire.h" // inclui biblioteca I2C
char c; // define variável c tipo caractere
char read_I2C; // define variável real_I2C tipo caractere
void OBC::init() // inicia função
{
Wire.begin(0x3C); // inicia comunicação com endereço do escravo (PL)
}
void Artigo_OBC::runCycle() // executa função
{
Wire.beginTransmission(0x3C); // inicia transmissão
Wire.write(ifaceOBC.TC);// envia informação
Wire.endTransmission();// termina transmissão
Wire.begin(0x3C); // inicia transmissão
Wire.requestFrom(0x3C,5); // solicita leitura de informação
while (Wire.available()) // avalia comunicação
{
char c = Wire.read(); // lê informação
read_I2C=c; // armazena informação lida na variável c
}
Wire.endTransmission(); // termino da transmissão
}
void OBC::enact_OBC_I2C_Connection_r1_Read_from_Slave( ) // executa função
{
ifaceOBC.Read = read_I2C; // transfere a informação lida para variável de leitura do OBC
}
	#include "Wire.h" // inclui biblioteca I2C
	char c; // define variável c tipo caractere
	char x; // define variável x tipo caractere
	char read_I2C; // define variável read_I2C tipo caractere
	void receiveEvent (int howMany) // inicia função para recebimento de informação
	{
	while (Wire.available()) // avalia comunicação
	{
	char c = Wire.read (); // lê informação
	read_I2C = c; // armazena a informação lida na variável read_I2C
	}
	} // término da função de recebimento de informação
	void PL::init() // executa função
	{
	Wire.begin (0x3C); // inicia comunicação I2C
	Wire.onReceive (receiveEvent); // executa função de recebimento de informação
	x=ifacePL.Send; // transfere a informação da variável lida pela PL para a variável x
	Wire.endTransmission(); // termino da transmissão
	}
	void PL::enact_PL_I2C_Connection_r1_Read_from_Master( ) // executa função
	{
	ifacePL.Read = read_I2C; // transfere a informação lida para a variável de leitura da PL
	}

Fonte: Produção do autor.

Havendo necessidade de testar os modelos com outro protocolo de comunicação, a biblioteca I2C poderá ser substituída por outra biblioteca.

5.3.1.8. Embarcar os códigos computacionais dos modelos

Para embarcar e simular os códigos dos modelos dos subsistemas nas placas computacionais programáveis utilizando o ambiente Eclipse/Yakindu são realizados os seguintes procedimentos: i) utilizar os ambientes de desenvolvimento Yakindu, um ambiente com o modelo do OBC e o outro ambiente com o modelo da PL, para embarcar e executar os códigos gerados de cada modelo nas respectivas placas computacionais que simulam o ambiente de cada subsistema, ii) interconectar as placas por meio de um barramento de comunicação conforme o protocolo de comunicação do

Cubesat, iii) utilizar os ambientes de desenvolvimento para executarem os códigos de cada placa, iv) no *menu* superior de cada ambiente de desenvolvimento selecionar, na janela superior do lado esquerdo de *on:*, a pasta do arquivo com extensão .sct criado a partir do modelo, v) selecionar no *menu* superior de cada ambiente, na janela do lado direito de *on:,* o console da placa a qual será embarcado os códigos do modelo a ser testado, e vi) selecionar no *menu* superior de ambos ambientes *Launch in RUN* para embarcar e executar os códigos nas respectivas placas, e vii) utilizar os consoles dos ambientes para verificar a execução dos códigos.

5.3.1.9. Validação dos requisitos do comportamento nominal

Os códigos gerados e embarcados nas placas computacionais são testados no barramento de comunicação. As placas são conectadas conforme o protocolo de comunicação do Cubesat implementado na arquitetura STAE. O objetivo é executar os códigos para simular os requisitos dos subsistemas modelados embarcados em ambiente simulado (SIL), mas que já faz uso do barramento de comunicação real.

No caso da execução ocorrer conforme as especificações dos requisitos, os modelos são considerados validados. Caso contrário, há necessidade de nova análise dos requisitos e dos modelos e realizar novamente os procedimentos para nova realização da evolução da sistemática de testes.

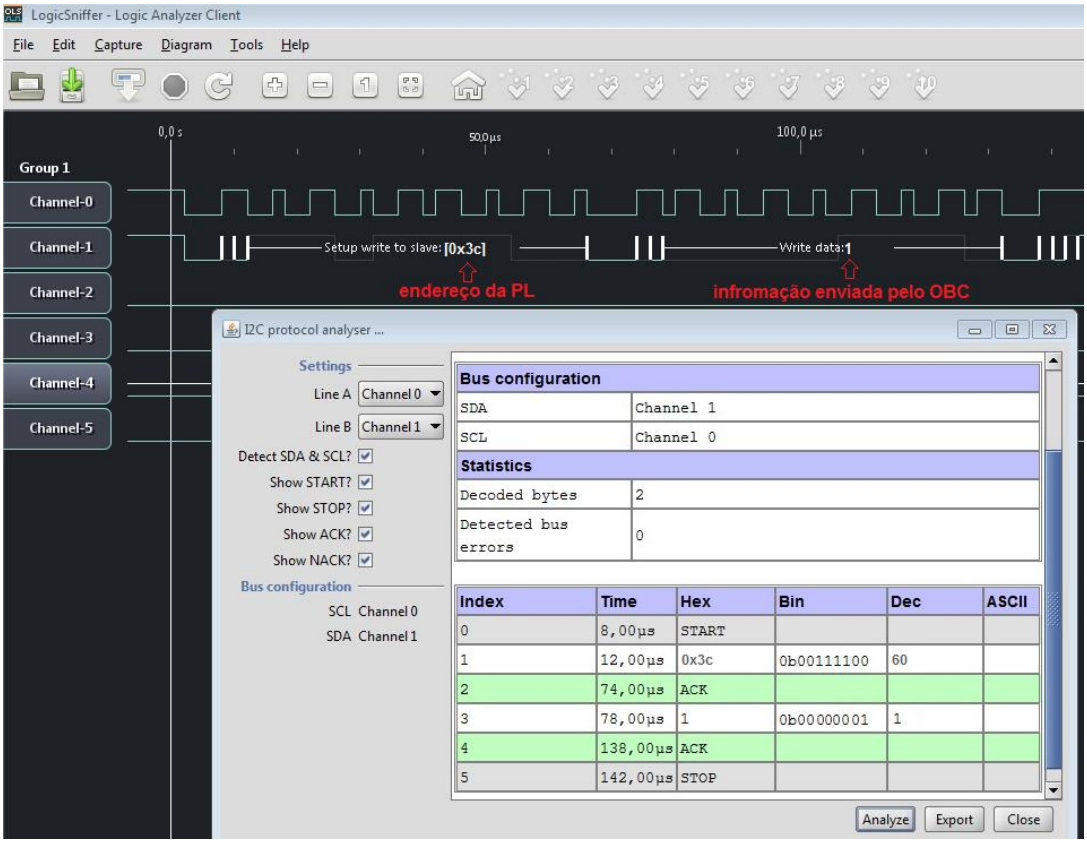
As falhas detectadas são relacionadas na planilha de dependabilidade e na tabela FMEA.

5.3.1.10. Analisador lógico

Na arquitetura do STAE é utilizado um analisador lógico para capturar e analisar as informações que trafegam no barramento de comunicação I2C. O analisador é conectado ao barramento e identifica os sinais SDA (*data*) e SCL (*clock*) do protocolo de comunicação. Conforme o padrão I2C é identificado o

endereçamento da mensagem, os sinais de leitura ou gravação, o pacote de mensagem e o término de cada pacote. Em destaque na parte inferior da Figura 5.9 apresenta as seguintes informações: i) início da mensagem (*START*), ii) endereçamento do envio da informação pelo OBC (*0x3c* = endereço da PL), iii) reconhecimento da informação (*ACK*), iv) informação enviada pelo OBC (*1*), v) reconhecimento da informação (*ACK*), vi) término do envio da informação (*STOP*), e vii) outras informações como tempo de envio do pacote, representação binária e decimal dessas informações.

Figura 5.9 – Analisador lógico I2C.



Fonte: Produção do autor.

O analisador lógico utilizado foi o "*LogicSniffer - Logic Analyzer Client*" versão "logic_analyzer-agla_v0_10", o qual captura as informações por meio do aplicativo ols-0.9.7.2 (LXTREME, 2017). Este analisador está disponibilizado gratuitamente na internet.

Caso as informações capturadas no barramento apresentam coerência com as especificações dos requisitos dos subsistemas comunicantes, considera-se que os modelos do OBC e PL estão validados no barramento (SIL).

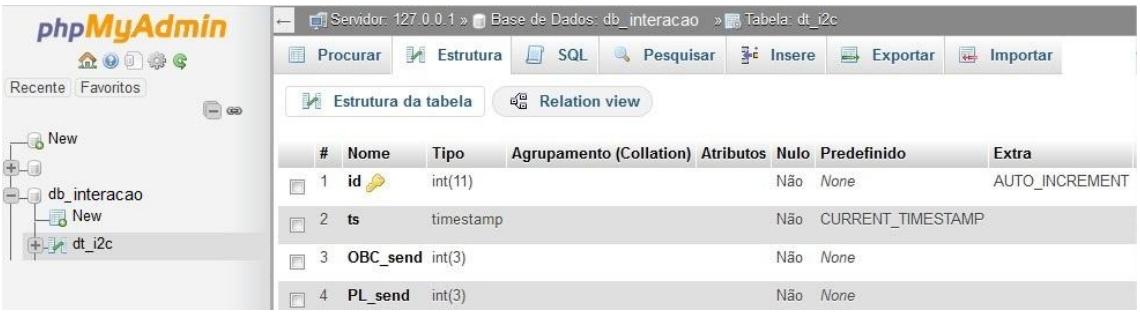
5.3.1.11. Base de dados Interação

Na evolução dos testes as informações geradas e trocadas na simulação entre os subsistemas em interação são armazenadas em uma base de dados denominada BD Interação. O objetivo é comparar as informações geradas com as informações esperadas conforme as especificações dos requisitos. Estes resultados armazenados podem ser utilizados para análise e comparação com outros resultados obtidos durante a evolução dos testes e também com outros testes similares.

A presente implementação utiliza um aplicativo computacional denominado Meu Administrador Pessoal de Páginas Caseiro (*Personal Home Page My Admin* – PhpMyAdmin), o qual possui recursos que facilitam o gerenciamento de base de dados por meio de uma linguagem denominada Minha Linguagem de Consulta (*My Structured Query Language* – MYSQL). A plataforma utilizada neste trabalho para uso desses aplicativos é *Xampp Control Painel versão 3.2.2*.

A base de dados que armazena as informações da interação entre os subsistemas comunicantes contém os seguintes dados: i) *id* (número sequencial de registro), ii) *timestamp* (data e hora de armazenamento da informação), iii) *OBC_send* (informação enviada pelo OBC, e iv) *PL_send* (informação enviada pela PL). A Figura 5.10 apresenta esta base de dados com uma tabela de dados denominada *dt_i2c*.

Figura 5.10 – Base de dados Interação.



Fonte: Produção do autor.

As bibliotecas e comandos necessários para os subsistemas comunicantes acessarem a base de dados são implementados manualmente nos códigos gerados dos modelos desses subsistemas.

A Tabela 5.11 apresenta algumas linhas de comandos implementadas manualmente referentes ao protocolo I2C, ethernet e acesso a base de dados. Cada linha possui um comentário após duas barras (//) descrevendo o que será realizado.

Tabela 5.11 - Comandos para gravar dados na base MySQL.

```
// bibliotecas ethernet e I2C
#include <SPI.h>
#include <String.h>
#include <Ethernet.h>
#include <Wire.h>
const byte MY_ADDRESS = 0x3C; // endereço da PL
// Configuração para conexão com Arduino
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x9B, 0x36 };
// Endereço do Arduino
byte ip[] = { 192, 168, 1, 100 };
// Endereço do computador - IP
byte servidor[] = { 192, 168, 1, 101 };
// Porta do Arquivo como servidor 8090
// O bando de dados MySQL está na porta 80
EthernetServer server(8090);
EthernetClient cliente;
// Base de Dados db_nanosatc_br
// Tabela dt_i2c
// Comandos para interagir com a base de dados
String readString = String(50);
// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0; // will store last time LED was updated
// constants won't change :
const long interval = 5000;// interval at which to blink (milliseconds)
  Ethernet.begin(mac, ip); // inicializa acesso via ethernet
  Serial.begin(9600); // define velocidade da porta serial
void () {
  EthernetClient client = server.available(); // inicializa ethernet com cliente
}
void () {
//MYSQL
  EthernetClient client = server.available(); // acesso da ethernet com cliente
    if (cliente.connect(servidor, 80)) { // verifica conexão com a porta 80
      Serial.println("Conected Data Base"); // aviso de conexão com base de dados
      // arquivo php com comandos para gravar dados na base de dados
      cliente.print("GET /arduino/salvardadosArtigo2.php?");
      cliente.print("OBC_send=");
      cliente.print(ifaceOBC.TC); // grava informação enviada pelo OBC
      cliente.print("&PL_send=");
      cliente.print(read_I2C); // grava informação enviada pela SLP
      cliente.stop(); // termino da conexão
    }
    else {
      Serial.println("Connection fail - Data base"); // falha na conexão
      cliente.stop(); // termino da conexão
    }
} //FIM MYSQL
```

Fonte: Produção do autor.

As linhas de comandos da Tabela 5.11 são implementadas no arquivo OBC.cpp.

A Tabela 5.12 apresenta as linhas de comandos de dois arquivos php: i) arquivo salvardados.php, o qual grava dados na base de dados, e ii) arquivo conecta.php, o qual realiza a conexão com a base de dados. Nas linhas de comandos está descrito após duas barras (//) o que será realizado.

Tabela 5.12 – Arquivo salvardados.php (esquerda) e conecta.php (direita).

salvardados.php

```
<?php
	include("conecta.php"); // acessa arquivo que realiza conexão com a base de dados
	$OBC_send = $_GET['OBC_send']; // lê conteúdo da variável OBC_send
	$PL_send = $_GET['PL_send']; // lê conteúdo da variável PL_send

	$sql_insert = "insert into dt_i2c (OBC_send,PL_send) values ('$OBC_send','$PL_send')"; // grava na tabela dt_i2c informações lidas

	mysql_query($sql_insert);
	if($sql_insert) // verifica se ocorreu erro
{
		echo "Salvo com sucesso";
	} else {
		echo "Ocorreu um erro";
	}
?>
```

conecta.php

```
<?php
	$usuario = "root"; // verifica usuário e senha
	$senha = "";
	$host = "localhost";
	$conexao = mysql_connect($host,$usuario,$senha);
	$selecionabd = mysql_select_db('db_interacao', $conexao); // seleciona a base de dados db_interacao
	if($conexao) // verifica conexão
	{
		echo "Conectou com sucesso";
	} else {
		echo "Ocorreu um erro";
	}
?>
```

Fonte: Produção do autor.

Os arquivos foram configurados em php para permitir que as bases de dados MySQL sejam acessadas pelas placas Arduino.

As execuções realizadas para acessar as bases de dados utilizando as placas Arduino são: i) conectar duas placas Arduino conforme padrão do protocolo de comunicação, ii) em uma das placas, conectar uma placa com acesso a *ethernet*, iii) conectar um equipamento que permita a conexão da placa *ethernet* do Arduino com a placa *ethernet* do computador ao qual as placas Arduino são conectadas (concentrador), e iv) identificar os endereços de Protocolo da Internet (*Internet Protocol – IP*) da placa Arduino e da placa de rede *ethernet* do computador. Esses endereços *IP*s são configurados no arquivo da Tabela 5.11 apresentado anteriormente.

Os resultados armazenados na base de dados denominada **Interação** são verificados quanto à conformidade do comportamento dos subsistemas embarcados nos Arduinos em relação ao comportamento especificado nos requisitos de interação entre os subsistemas OBC e uma PL, validando assim a interoperabilidade dos subsistemas modelados em ambiente simulado com barramento real (SIL).

5.3.2. **Módulo 2: Validação da robustez do comportamento dos subsistemas**

Na sequencia são implementados os requisitos de robustez nos modelos nominais dos subsistemas simulados. O objetivo é evitar as possíveis falhas identificadas e relacionadas na planilha de dependabilidade e na tabela FMEA, desenvolvidas para a abordagem apresentada utilizando a taxonomia de dependabilidade.

Na planilha e na tabela FMEA são relacionados procedimentos com mitigações para evitar cada falha identificada. O objetivo é implementar as falhas identificadas em um dos modelos validado gerando um modelo enriquecido com falhas. As mitigações são implementadas no outro modelo gerando um modelo enriquecido com robustez.

Destes modelos enriquecidos são gerados novos códigos computacionais e implementados em placas computacionais distintas. O objetivo é validar os requisitos do modelo com robustez provocando falhas por meio do modelo enriquecido com falha.

Posteriormente, são implementadas falhas no modelo nominal que anteriormente foi enriquecido com robustez. O outro modelo nominal que foi anteriormente enriquecido com robustez agora é enriquecido com falhas. Ou seja, houve inversão dos modelos enriquecidos, cuja finalidade agora é validar os requisitos de robustez do outro modelo. Na sequência, são testados os modelos com robustez validados em interoperabilidade cujo objetivo é validar os requisitos de robustez dos subsistemas simulados em interação (MIL).

As evoluções dos testes até o momento buscam o tratamento das possíveis falhas identificadas validando a robustez do comportamento dos subsistemas simulados.

Os procedimentos para realizar esses objetivos são: 1) identificar e evitar possíveis falhas nas interações dos subsistemas comunicantes usando a taxonomia de dependabilidade, 2) enriquecer os modelos nominais com falhas

e robustez para testar os requisitos de robustez de ambos os subsistemas, e 3) testar os requisitos de robustez dos modelos em interoperabilidade.

5.3.2.1. Tratamento de falhas

O objetivo é identificar e evitar possíveis falhas que possam ocorrer durante a interação dos subsistemas OBC e PL embarcados no nanosatélite. Os seguintes procedimentos são realizados para identificação de possíveis falhas:

- Análise da tabela de requisitos: Analisar a tabela de requisitos para identificar operações e elementos não especificados que possam causar comportamentos indesejáveis, e consequentemente falhas. Nessa tabela são realizados questionamentos buscando identificar operações e elementos ainda não relacionados aos requisitos levantados.
- Planilha de dependabilidade: Relacionar na planilha de dependabilidade, desenvolvida nesta tese, as possíveis falhas identificadas nas respectivas classes de falhas baseadas no conceito de dependabilidade. Para cada possível falha é relacionada no mínimo uma mitigação.
- Árvore de dependabilidade: Identificar por dependabilidade e rastreabilidade se uma possível falha pode ocasionar outra falha.
- Tabela FMEA: Relacionar as possíveis falhas identificadas com os mecanismos necessários para mitigação dessas falhas.

As mitigações são implementadas nas modelagens nominais realizando transformação de modelos com robustez, cujo objetivo é evitar os comportamentos indesejáveis dos subsistemas frente às falhas.

5.3.2.2. Análise da tabela de requisitos

O objetivo é identificar possíveis falhas que possam ocorrer levando em consideração limitações, falhas de software, interferência física, entre outros tipos de falhas relacionados na taxonomia de dependabilidade.

Analisando a tabela de requisitos da interação entre os subsistemas OBC e PL surgem questionamentos sobre limitações. Limitações dos subsistemas e limitações nos momentos de interação são preocupações eminentes. Como exemplo, na tabela de requisitos o elemento **10-Processamento** questiona a respeito de limitações de desempenho. Os subsistemas OBC e uma carga útil (PL) trocam informações por meio do barramento de comunicação do Cubesat. As limitações do barramento são analisadas e identificadas na planilha de dependabilidade conforme sua classificação, assim como as mitigações necessárias.

5.3.2.3. Planilha de dependabilidade

O objetivo desta planilha é relacionar cada possível falha identificada na classificação de falhas da taxonomia de dependabilidade com suas respectivas mitigações.

Para exemplificar, dois exemplos de falhas que podem ocorrer na interação entre os subsistemas comunicantes são relacionados: 1) subsistema não enviar informações pelo canal de comunicação, e 2) subsistema não receber informação especificada. O exemplo um (1) pode ser um erro de Interferência Física (grupo de Falhas de Interação na taxonomia de dependabilidade). O exemplo dois (2) pode ocorrer pelo mesmo motivo de Interferência Física. Se ocorrer erro no barramento de comunicação um subsistema pode receber informações com erros ou até não receber informações. Neste exemplo, se os erros de interferência física não forem previstos nos software dos subsistemas (uma situação não prevista pode ocasionar Falha de Software) o sistema poderá ficar parado (grupo de Falhas de Desenvolvimento na taxonomia de dependabilidade).

A Figura 5.11 cataloga na planilha de dependabilidade apenas os dois exemplos de falhas descritos anteriormente. Nesta planilha cada falha catalogada está relacionada com apenas uma mitigação que pode ser realizada para evitar a respectiva falha. Outras mitigações também podem ser realizadas.

Essa planilha apresenta as seguintes informações: (a) possíveis falhas identificadas. Essas falhas são relacionadas com os seguintes itens: b) classe de falha, c) tipo de falha, d) grupo, e) combinação a que pertence, e f) mitigação para evitar a determinada falha.

Figura 5.11 – Planilha de dependabilidade (dois exemplos de possíveis falhas).

Planilha de dependabilidade					Mitigações
Classes de Falhas (linha) Combinações de Falhas (coluna)	d →	Grupos	Falhas de Desenvolvimento	Falhas de Interação	
b ↓	e →	Combinações	1	16	
	c →	Falhas	A) Falhas de Software	F) Interferência Física	
Classes de Falhas		Subclasses	a ↓		f ↓
Fase de criação ou ocorrência	I	Desenvolvimento	Subsistema não receber informação especificada		Definir operações a serem realizadas no caso de não receber informação especificada
	II	Operacional			
Limites do Sistema	III	Interna			
	IV	Externa			
Causa fenomenológica	V	Evento Natural			
	VI	Humana			
Dimensão	VII	Hardware			
	VIII	Software			
Objetivo	IX	Não Maliciosa			
	X	Maliciosa			
Intenção	XI	Não Intencional			
	XII	Intencional			
Capacidade	XIII	Acidental			
	XIV	Incompetência			
Persistência	XV	Permanente		a ↓	
	XVI	Transitória		Subsistema não enviar informações pelo canal de comunicação	Analisar tensão nos subsistemas

Fonte: Produção do autor.

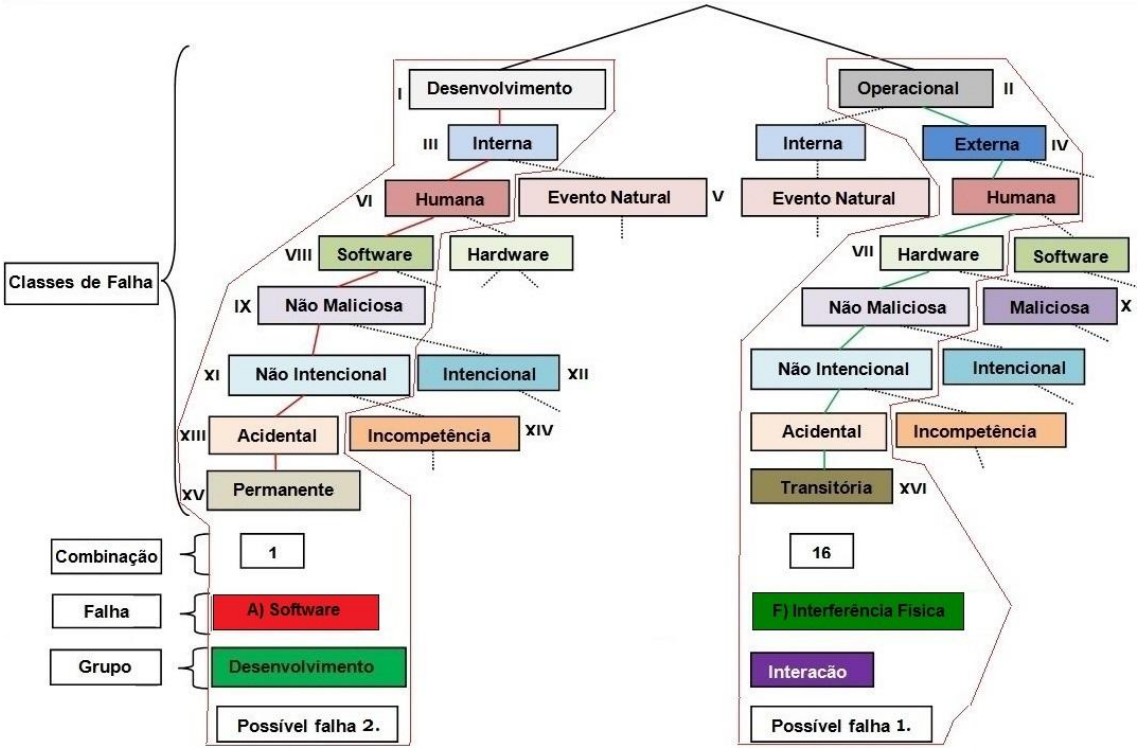
A planilha de dependabilidade é utilizada ao final de cada teste realizado entre os subsistemas no STAE. Se o levantamento dos requisitos e a identificação das possíveis falhas estão definidos de forma a atender as necessidades do sistema, os erros detectados durante os testes, por forma natural ou induzido, estão relacionados nesta planilha.

5.3.2.4. Árvore de dependabilidade

A árvore de dependabilidade é definida baseada nos conceitos da taxonomia da dependabilidade (AVIZIENIS et al., 2004). O objetivo da utilização desta árvore é permite uma visão gráfica dos relacionamentos entre as classes de falhas. Essa visão também permite analisar se uma falha em um determinado grupo pode por dependabilidade e rastreabilidade ocasionar outro tipo de falha.

A Figura 5.12 apresenta a relação dos dois exemplos de falhas relacionado anteriormente identificados nos seguimentos da árvore de dependabilidade. Esta figura apresenta de forma gráfica o relacionamento dessas falhas por dependabilidade e rastreabilidade. Os seguimentos são: 1) Falha de Interferência Física do grupo de Falhas de Interação, e 2) Falha de Software do grupo de Falhas de Desenvolvimento.

Figura 5.12 – Relacionamento entre possíveis falhas.



Fonte: Produção do autor.

As mitigações para evitar essas falhas devem estar previstas para o grupo de Interferência Física e analisado por dependabilidade e, por rastreabilidade também devem estar previstas no grupo de Desenvolvimento de Software.

5.3.2.5. Tabela FMEA

As possíveis falhas selecionadas e identificadas na planilha de dependabilidade são relacionadas na tabela FMEA customizada para a abordagem apresentada nesta tese. Esta tabela é composta pelas informações das possíveis falhas relacionadas na planilha e complementada com as seguintes informações: a) procedimento identificando possível falha, b) subclasse, combinação e identificação da falha (estas informações estão relacionadas com a mitigação na planilha), c) efeito, d) local afetado, e e) mecanismo de controle. A Tabela 5.13 relaciona essas informações.

Tabela 5.13 - FMEA com as possíveis falhas selecionadas.

a) Procedimento identificando possível falha	b) Subclasse / Combinação / Identificação da falha	c) Efeitos da falha (consequência)	d) Local afetado pela falha	e) Mecanismo de controle para detecção da falha
1.Conexão física dos subsistemas no barramento	XVI / 16 / .1	Subsistemas não trocam informações	Subsistemas	Analisador Lógico I2C
2.Envio de informações de um subsistema para o outro	I / 1 / .1	Subsistemas ficarem parados	Subsistemas	Analisador lógico I2C

Fonte: Produção do autor.

Conforme os testes são realizados, novas informações são incrementadas permitindo nova análise dos requisitos, análise de falhas, modelagem e casos de teste, cujo objetivo é aprimorar o STAE para tratamento de falhas na interação entre os subsistemas comunicantes do Cubesat.

A tabela FMEA e a planilha de dependabilidade são utilizadas no aplicativo Excel da Microsoft permitindo que haja relacionamento entre a tabela e a planilha.

5.3.2.6. Modelos enriquecidos (PL com falha e OBC com robustez)

As falhas identificadas na planilha de dependabilidade são os comportamentos indesejados a serem utilizados para enriquecer um modelo com falhas e o outro modelo é enriquecido com as respectivas mitigações para evitar as falhas identificadas gerando um modelo com robustez.

Na abordagem apresentada inicialmente o modelo nominal da PL é enriquecido com falhas e o modelo nominal do OBC é enriquecido com robustez. O objetivo é o modelo enriquecido com falhas da PL testar a robustez do modelo com robustez do OBC.

5.3.2.7. Modelo nominal da PL enriquecido com falha

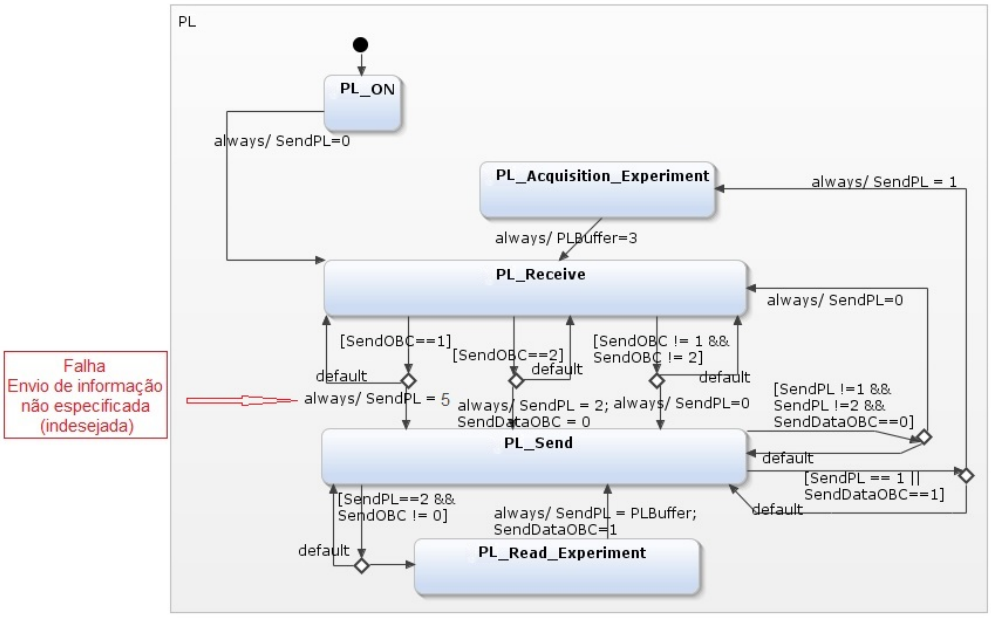
Para exemplificar a evolução dos testes, o modelo nominal da carga útil (PL) é enriquecido com a possível falha de número dois (2) identificada na planilha de dependabilidade. O modelo também é enriquecido com eventos e transições necessários para provocar a falha no outro modelo.

Analisando a possível falha identificada de número 2 (Subsistema receber informação não especificada) com as especificações dos requisitos temos:

- A PL deve retornar (*OK*) informando que recebeu de forma certa o primeiro comando enviado pelo OBC ou, no caso de ocorrer problema, retornar (*Problem*). No caso da PL sempre retornar que houve problema, o OBC poderá ficar sempre enviando o primeiro comando sem comunicar a estação solo que está havendo problema. Sendo assim, o modelo nominal da PL é enriquecido com falha enviando um comando não especificado (número cinco) para ser recebido pelo OBC, para testar o modelo do OBC.

A Figura 5.13 apresenta o modelo da PL enriquecido com falha.

Figura 5.13 – Modelo da PL enriquecido com falha.



Fonte: Produção do autor.

A abordagem apresentada permite inverter a sequência de testes enriquecendo com falhas o modelo nominal do OBC.

5.3.2.8. Modelo nominal do OBC enriquecido com robustez

O modelo nominal do OBC é enriquecido com robustez, mitigações relacionadas na planilha de dependabilidade, as quais são realizadas para evitar as falhas provocadas pelo modelo enriquecido com falhas. O modelo nominal do OBC enriquecido com robustez gera um modelo enriquecido com robustez. Analisando a falha de número 2 descrita anteriormente temos:

- No caso do modelo enriquecido com falha da PL retornar sempre informando que houve problema ao receber o primeiro comando do OBC, o modelo nominal do OBC deve ser enriquecido com a robustez necessária para identificar esse tipo de falha e realizar ações cabíveis para ser solucionada esta falha.

A Tabela 5.14 apresenta em negrito os procedimentos implementados no modelo nominal do OBC para enriquecer o modelo com robustez, cujo objetivo é identificar e realizar as ações necessárias para evitar a falha provocada pelo modelo enriquecido com falha da PL.

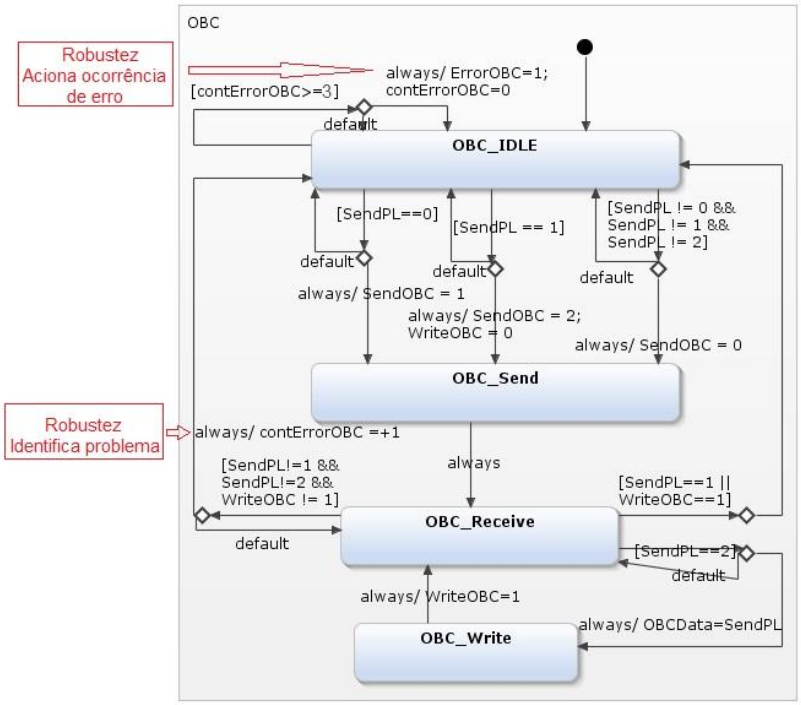
Tabela 5.14 - Estados, eventos e ações do OBC com robustez.

Estados **OBC**	Eventos	Ações	Descrição	Condições	Descrição
IDLE	*Problem Step* 3.2	**contErrorOBC=+1**	**Incrementar 1 na variável contadora de erro (contErrorOBC)**	SendPL não=1 e não=2 e WriteOBC não=1	OBC recebeu informação que a PL teve problema ao receber comando do OBC e OBC não reconheceu informação da PL e OBC não gravou dados do experimento
	contErrorOBC	**ErrorOBC=1 contErrorOBC=0**	**Variável indicadora de que houve erro é acionada (ErrorOBC=1). Inicia variável contadora de erro (contErrorOBC=0)**	**contErrorOBC >=3**	**Verifica se ocorreu erro três vezes ou mais**
Receive	*Step* n+2	WriteOBC=1 **ErrorOBC=0**	Informa que OBC gravou os dados do experimento. **Inicializa variável indicadora de erro.**		

Fonte: Produção do autor.

A Figura 5.14 apresenta o modelo com robustez do OBC identificando os pontos aonde foi incluído robustez. Esse subsistema quando recebe da PL a informação que houve problema acrescenta um (1) na variável contadora de erro (*contErrorOBC*). Após recebimento de três informações de erro, o OBC identifica por meio da variável *ErrorOBC* que está ocorrendo erro e retorna a enviar comandos para a PL.

Figura 5.14 – Modelo OBC enriquecido com robustez.



Fonte: Produção do autor.

A robustez implementada permite a Estação Solo verificar por meio da variável denominada *ErrorOBC* que está ocorrendo erro na interação entre o OBC e a PL e tomar devidas providências.

5.3.2.9. Geração dos códigos computacionais dos modelos (PL com falha e OBC com robustez)

Realizados os testes da interação por simulação dos modelos (PL enriquecido com falha e OBC enriquecido com robustez) são realizados os mesmos

procedimentos descritos anteriormente, na subseção 5.3.1.3. e 5.3.1.3.1, para gerar e embarcar em placas computacionais, respectivamente, os códigos desses modelos de forma automatizada utilizando conceitos MDE.

Os códigos gerados e embarcados agora são do modelo PL enriquecido com falha e do modelo OBC enriquecido com robustez. O objetivo é executar esses códigos para verificar a robustez do modelo OBC.

5.3.2.10. Verificação e validação dos requisitos dos modelos (PL com falha e OBC com robustez)

Os códigos gerados e embarcados nas placas computacionais são testados no barramento de comunicação. As placas são conectadas conforme o protocolo de comunicação do Cubesat. O objetivo agora é verificar e validar os requisitos do modelo OBC enriquecido com robustez embarcado em ambiente simulado (SIL), mas que já faz uso do barramento de comunicação real.

No caso da execução ocorrer conforme as especificações dos requisitos, os requisitos do modelo com robustez do OBC estão validados. Neste procedimento é analisado se a possível falha identificada foi evitada, se existe falha ainda não identificada ou se há necessidade de fazer nova análise dos modelos e/ou dos requisitos.

As falhas identificadas são relacionadas na planilha de dependabilidade e na tabela FMEA.

5.3.2.11. Base de dados Interação

Os mesmos procedimentos da subseção 5.3.1.5. para armazenar os resultados na Base de Dados **Interação** são realizados. Os registros armazenados agora são referentes às informações trocadas durante os testes de interação entre o modelo PL com falha e OBC com robustez.

O objetivo é verificar se as informações geradas e registradas testando a robustez do OBC estão coerentes conforme as especificações dos requisitos.

5.3.2.12. Tratamento de falhas

Os mesmos procedimentos da subseção 5.3.2.1. para tratamento de falhas são realizados. O objetivo agora é identificar possíveis falhas na planilha de dependabilidade para enriquecer o modelo nominal do OBC com falhas e identificar as respectivas mitigações para enriquecer o modelo nominal da PL com robustez. O objetivo é testar o modelo com robustez da PL.

São realizados os mesmos procedimentos descritos nas subseções 5.3.2.1.1., 5.3.2.1.2., 5.3.2.1.3. e 5.3.2.1.4.: i) análise da tabela de requisitos, ii) planilha de dependabilidade, iii) árvore de dependabilidade, e iv) tabela FMEA, respectivamente.

5.3.2.13. Modelos enriquecidos (OBC com falha e PL com robustez)

Na evolução dos testes agora o modelo nominal do OBC é enriquecido com as falhas identificadas utilizando a taxonomia de dependabilidade e o modelo nominal da PL é enriquecido com as respectivas mitigações relacionadas para evitar as falhas identificadas. O modelo nominal do OBC enriquecido com falhas, comportamento indesejados, gera um modelo com falhas e o modelo nominal da PL enriquecido com as mitigações gera um modelo com robustez. O objetivo é o modelo OBC enriquecido com falhas testar a robustez do modelo PL.

5.3.2.14. Modelo nominal do OBC enriquecido com falha

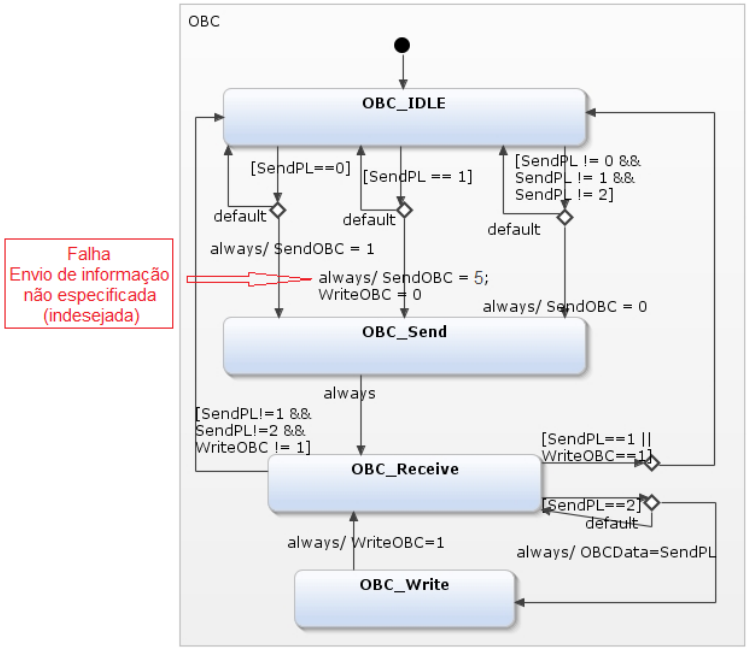
O modelo nominal do OBC é enriquecido com possíveis falhas identificadas na planilha de dependabilidade. O modelo também é enriquecido com eventos e transições necessários para provocar falha no modelo da PL com robustez. Uma falha parecida aplicada no modelo nominal da PL também foi implementada no modelo nominal do OBC. Ou seja, o modelo com falhas do OBC também envia uma informação não especificada nos requisitos para o

modelo com robustez da PL. O objetivo é verificar os requisitos de robustez do modelo da PL. A falha implementa no modelo do OBC é descrita a seguir:

- O OBC deve solicitar leitura dos dados do experimento da PL. No caso do OBC não enviar o comando solicitando de leitura dos dados, a PL não enviará esses dados. Desta forma, a falha implementada foi, o modelo nominal do OBC é enriquecido com uma falha, a qual envia um comando não especificado (número cinco) ao invés de solicitar dados do experimento. Sendo assim, a PL não recebe o comando solicitando os dados do experimento e o teste verifica a robustez do comportamento do modelo da PL para esse tipo de falha.

A Figura 5.15 apresenta o modelo nominal do OBC enriquecido com a falha, o qual envia uma informação não especificada nos requisitos (indesejada) para a PL.

Figura 5.15 – Modelo OBC enriquecido com falha.



Fonte: Produção do autor.

O comando de número cinco (5) sendo enviado pelo OBC não está especificado nos requisitos.

5.3.2.15. Modelo nominal da PL enriquecido com robustez

O modelo nominal da PL neste procedimento na evolução dos testes é enriquecido com robustez, mitigações relacionadas na planilha de dependabilidade, as quais são realizadas para evitar as falhas provocadas pelo modelo enriquecido com falhas do OBC. Analisado a falha enriquecida no modelo do OBC temos:

- No caso do modelo enriquecido com falha do OBC não solicitar para a PL o envio dos dados do experimento e enviar uma informação não especificada nos requisitos, o modelo nominal da PL não envia os dados do experimento e sempre retorna a informação comunicando problema (*Problem*). Para esse tipo de falha o modelo nominal da PL é enriquecido com robustez gerando um modelo com robustez buscando identificar e evitar esse tipo de falha.

A Tabela 5.15 apresenta em negrito os procedimentos implementados no modelo nominal da PL para enriquecer o modelo com robustez, cujo objetivo é identificar e realizar as ações necessárias para evitar a falha provocada pelo modelo enriquecido com falha do OBC.

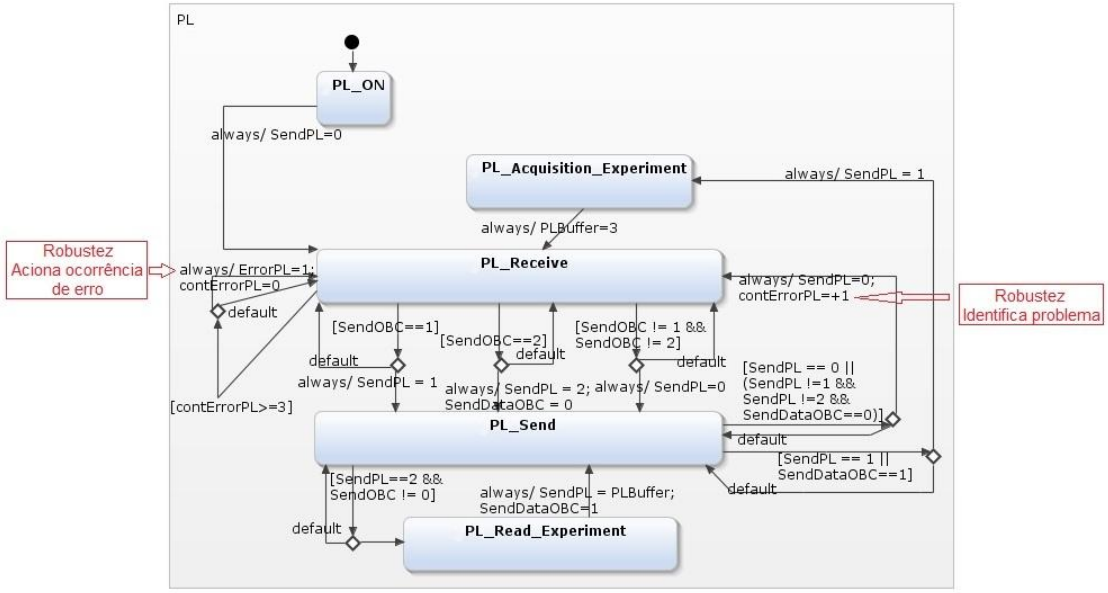
Tabela 5.15 - Estados, eventos e ações da PL com robustez.

Estados **PL**	Eventos	Ações	Descrição	Condições	Descrição
Receive	*Problem Step* 3.2/6.2..n	SendPL=0, **contErrorPL=+1**	PL em operação para receber comando. **Incrementar 1 na variável contadora de erro (contErrorPL)**	SendPL =0 **ou (SendPL não=1 e não=2 e SendDataOBC=0)**	PL identifica problema no recebimento do comando do OBC e **dados também não foram enviado (SendDataOBC=0)**
	contErrorPL	**ErrorPL=1, contErrorPL=0**	**Variável indicadora de que houve erro é acionada (ErrorPL=1) Inicia variável contadora de erro (contErrorPL=0)**	**contErrorPl>=3**	**Verifica se ocorreu três vezes ou mais erros**
Send	*Experiment Data Step* n+2	SendPL=PLBuffer, SendDataOBC=1, **ErrorPL=0**	PL envia para o OBC dados do experimento armazenados no seu Buffer (PLBuffer). Inicializa variável indicando que dados foram enviados para o OBC. **Inicializa variável indicadora de erro.**		

Fonte: Produção do autor.

A Figura 5.16 apresenta o modelo PL identificando os pontos aonde foi incluído robustez.

Figura 5.16 – Modelo PL enriquecido com robustez.



Fonte: Produção do autor.

A PL ao receber um comando não especificado do OBC identifica o problema e incrementa um (1) na variável contadora de erro (contErrorPL). No caso do problema persistir por cinco vezes ou mais, é acionado uma variável identificadora de erro (*ErrorPL*). Esta variável permite ser identificada e verificada pela Estação Solo.

5.3.2.16. Geração dos códigos computacionais dos modelos (OBC com falha e PL com robustez)

Realizado os testes da interação por simulação dos modelos (OBC enriquecido com falha e PL enriquecido com robustez) são realizados os mesmos procedimentos descritos anteriormente, na subseção 5.3.1.3. e 5.3.1.3.1. para gerar e embarcar em placas computacionais, respectivamente, os códigos desses modelos de forma automatizada utilizando a ferramenta MDE.

Os códigos gerados agora são do modelo OBC enriquecido com falha e do modelo PL enriquecido com robustez. O objetivo é executar esses códigos para verificar a robustez do modelo PL.

5.3.2.17. Verificação e validação dos requisitos nos modelos (OBC com falha e PL com robustez)

Os códigos gerados e embarcados nas placas computacionais são testados no barramento de comunicação. As placas são conectadas conforme o protocolo de comunicação do Cubesat. O objetivo agora é verificar e validar os requisitos do modelo PL enriquecido com robustez embarcado em ambiente simulado (SIL), mas que já faz uso do barramento de comunicação real.

No caso da execução demonstrar conformidade com as especificações dos requisitos, os requisitos do modelo com robustez da PL estão validados. Neste procedimento é analisado se a possível falha identificada foi evitada, se existe falha ainda não identificada ou se há necessidade de fazer nova análise dos modelos e/ou dos requisitos.

As falhas detectadas são relacionadas na planilha de dependabilidade e na tabela FMEA.

5.3.2.18. Verificação e validação dos requisitos de interoperabilidade e robustez dos modelos com robustez

Os modelos com robustez dos subsistemas simulados OBC e PL validados separadamente agora são testados por simulação em interoperabilidade utilizando conceitos MBT (MIL). O objetivo é verificar e validar os requisitos de robustez dos subsistemas comunicantes e posteriormente validar casos de teste nestes modelos.

No caso da execução demonstrar conformidade com as especificações dos requisitos, os modelos com robustez estão validados. Caso contrário, há

necessidade de nova análise dos requisitos e realização novamente dos procedimentos para nova evolução dos modelos.

As falhas identificadas são relacionadas na planilha de dependabilidade e na tabela FMEA.

5.3.3. **Módulo 3: Validação dos Casos de Teste**

As falhas identificadas e relacionadas na planilha de dependabilidade nesta evolução são utilizadas para gerar estímulos nos modelos OBC e PL cuja robustez tenha sido validada. O objetivo é gerar casos de teste abstratos. Os casos de teste são gerados automaticamente utilizando a ferramenta Uppaal baseada em conceito MBT. Os casos de teste são gerados selecionando o *menu Yggdrasil* na Uppaal. O objetivo é gerar, executar e validar, por simulação (MIL), casos de teste dos modelos OBC e PL com robustez em interação.

Após os testes atenderem as especificações dos requisitos, são gerados de forma automatizada, por meio de ferramenta Yakindu baseada em conceito MDE, os códigos dos modelos testados nos casos de teste. Esses códigos são embarcados em placas computacionais distintas. O objetivo é executar e validar os casos de teste gerados no barramento de comunicação (SIL).

5.3.3.1. Tratamento de falhas

Os mesmos procedimentos da subseção 5.3.2.1. para tratamento de falhas são realizados. Porém agora, o objetivo é utilizar as falhas identificadas na planilha de dependabilidade para efetuar estímulos durante a interoperabilidade entre os modelos OBC e PL com robustez validados.

São realizados os mesmos procedimentos descritos nas subseções 5.3.2.1.1., 5.3.2.1.2., 5.3.2.1.3. e 5.3.2.1.4.: i) análise da tabela de requisitos, ii) planilha de dependabilidade, iii) árvore de dependabilidade, e iv) tabela FMEA, respectivamente.

5.3.3.2. Estímulos

O objetivo é efetuar estímulos de comportamento nos subsistemas sob teste para gerar casos de teste de forma automatizada, utilizando conceitos MBT, e evitar as falhas identificadas durante esses testes.

A utilização da técnica MBT possibilita enriquecer os modelos com valores, eventos e transições para efetuar estímulos de comportamento entre os subsistemas em interação. Os estímulos permitem testar a interoperabilidade e robustez entre os subsistemas e gerar casos de teste de situações que possam não estar previstas nas definições dos requisitos. Analisando as possíveis falhas classificadas na planilha de dependabilidade e na tabela FMEA são identificados estímulos que geram casos de teste para verificar a robustez dos subsistemas comunicantes.

5.3.3.3. Base de dados Estímulo

Os estímulos identificados são armazenados em uma base de dados denominada BD Estímulo. A tabela contendo esses dados é denominada *dt_est* sendo estruturada da seguinte forma: i) *id* (número sequencial de registro), ii) *timestamp* (data e hora da interação do OBC com a PL), iii) *nsub* (número do subsistema que envia o estímulo de falha [1 a 100 = OBC, 201 a 200 = uma determinada PL, 201 a 300 = outra determinada PL ...]), e iv) *est* (informação contendo o estímulo de falha). A Figura 5.17 apresenta essa base de dados.

Figura 5.17 – Base de dados Estímulo.

Fonte: Produção do autor.

O objetivo é utilizar as informações desta base de dados para efetuar estímulos nos subsistemas sob teste e gerar casos de teste para observar se os resultados estão conforme as especificações dos requisitos de interação entre os subsistemas comunicantes.

5.3.3.4. Geracão dos Casos de Teste

Os modelos com robustez do OBC e da PL validados na ferramenta MDE denominada Yakindu são nesta evolução modelados na ferramenta Uppaal para gerar casos de teste de forma automatizada. Para gerar os casos de teste é selecionado o *submenu Yggdrasil* na Uppaal.

Repete-se a falha identificada anteriormente, a qual foi provocada pelo subsistema simulado da PL por meio de um evento não especificado nos requisitos (envia o número cinco). Efetuando-se um estímulo no modelo OBC com robustez um caso de teste é gerado. O objetivo é gerar casos de teste de forma automatizada e verificar os resultados da interação entre os modelos com robustez OBC e PL. Esse caso de teste é apresentado na Tabela 5.16.

Tabela 5.16 - Caso de teste.

Transições	Ações
OBC[OBC_IDLE->OBC_Send]	OBC envia primeiro comando (1) para PL
PL[PL_ON->PL_Receive]	PL recebe comando do OBC
PL[PL_Receive->PL_Send]	PL envia primeira informação (5) para OBC (informação não especificada)
OBC[OBC_Send->OBC_Receive]	OBC recebe informação da PL
OBC[OBC_Receive->OBC_IDLE]	OBC identifica problema (1ª vez)
OBC[OBC_IDLE->OBC_Send]	OBC verifica informação e envia novamente comando
PL[PL_Send->PL_Receive]	PL recebe novo comando do OBC
PL[PL_Receive->PL_Send]	PL envia informação não especificada para o OBC
OBC[OBC_Send->OBC_Receive]	OBC recebe informação da PL
OBC[OBC_Receive->OBC_IDLE]	OBC identifica problema (2ª vez)
OBC[OBC_IDLE->OBC_Send]	OBC verifica informação e envia novamente comando
PL[PL_Send->PL_Receive]	PL recebe comando do OBC
PL[PL_Receive->PL_Send]	PL envia informação não especificada para o OBC
OBC[OBC_Send->OBC_Receive]	OBC recebe informação da PL
OBC[OBC_Receive->OBC_IDLE]	OBC identifica problema (3ª vez)
OBC[OBC_IDLE->OBC_IDLE]	OBC aciona indicação de erro
OBC[OBC_IDLE->OBC_Send]	OBC verifica informação e envia comando
PL[PL_Send->PL_Receive]	PL recebe comando do OBC

Fonte: Produção do autor.

As setas (->) indicam a transição de estados da esquerda para a direita.

Este caso de teste apresenta que o OBC verificou que houve erro em três tentativas de reconhecimento da informação não especificada enviada pela PL. Na terceira tentativa o OBC acionou uma indicação de erro que permite ser verificada pela Estação Solo e tomar as devidas providências. Este caso apresenta um exemplo que a modelagem do OBC contém robustez para detectar este tipo de erro.

Como pode ser observada, esta possível falha havia sido detectada e apresentada anteriormente na subseção 5.3.2.2.2. nos procedimentos para evitar falhas e foi implementado no modelo do OBC a devida robustez.

Os estímulos também são efetuados no modelo da PL para verificar a robustez desse subsistema.

5.3.3.5. Geração dos códigos computacionais dos modelos OBC e PL com robustez

Agora os estímulos utilizados anteriormente para gerar os casos de teste são implementados nos modelos OBC e PL com robustez. Nesta evolução são realizados os mesmos procedimentos descritos anteriormente, na subseção 5.3.1.3. e 5.3.1.3.1, quando cada subsistema com robustez foi testado separadamente. Desta vez ambos os códigos dos modelos com robustez validados, do OBC e da PL, são gerados, embarcados e testados conjuntamente.

O objetivo é testar a robustez desses subsistemas em interação utilizando o barramento de comunicação real e validar os estímulos adotados nos testes gerando assim casos de teste.

5.3.3.6. Verificação dos Casos de Teste

Os códigos gerados e embarcados nas placas computacionais são testados no barramento de comunicação. As placas são conectadas conforme o protocolo de comunicação do Cubesat. O objetivo é executar os casos de testes nos

subsistemas OBC e PL com robustez e verificar sua efetividade em validar a interoperabilidade e robustez dos subsistemas OBC e PL no barramento de comunicação (SIL).

5.3.3.7. Avaliação dos resultados dos Casos de Teste

O objetivo é avaliar se os resultados dos casos de teste estão conforme as informações geradas durante a execução dos casos de testes de cada subsistema com robustez testado isoladamente, armazenadas na base de dados anteriormente.

Não ocorrendo falhas ou ocorrendo falhas provocadas pelos estímulos de falhas e essas falhas estarem identificadas na planilha de dependabilidade, o próximo procedimento é testar e validar os subsistemas reais.

No caso da execução demonstrar conformidade com as especificações dos requisitos, os modelos são considerados validados. Caso contrário, há necessidade de nova análise dos requisitos e realização novamente dos procedimentos para nova evolução dos testes.

As falhas detectadas são relacionadas na planilha de dependabilidade e na tabela FMEA.

As evoluções até o momento compõem o Cenário A, validando a modelagem dos requisitos de comportamento dos subsistemas (MIL) e validando os casos de testes simulando o comportamento dos subsistemas no barramento de comunicação (SIL). Uma vantagem importante neste cenário é que grande parte do conjunto de teste são reutilizados na verificação e validação dos diferentes modelos no STAE.

5.3.4. **Módulo 4: Validação dos requisitos dos subsistemas reais**

Neste módulo o objetivo é verificar e validar os requisitos de cada um dos subsistemas reais separadamente (HIL). Um subsistema real é conectado ao barramento de comunicação com o outro subsistema, o qual faz interação,

porém esse outro subsistema é um subsistema simulado validado. Uma questão importante neste cenário é a reutilização dos conjuntos de testes já especificados e validados nos módulos 1, 2 e 3, reduzindo o custo e o esforço na fase de aceitação.

5.3.4.1. Validação do subsistema real PL

Testar a interoperabilidade entre o subsistema OBC com robustez simulado (S1r) validado e um Subsistema PL Real. O objetivo é verificar se o Subsistema PL Real não apresenta falhas de interoperabilidade no barramento (HIL) com o subsistema OBC simulado.

Havendo falha, analisa-se qual subsistema apresentou a falha. Se a falha é devido a problema no subsistema simulado, é analisado novamente o subsistema em teste corrigindo a falha detectada. Se a falha é devido ao subsistema real, o desenvolvedor é contactado.

5.3.4.2. Validação do subsistema real OBC

Testar a interoperabilidade entre o subsistema PL com robustez simulado (S2r) validado e um Subsistema Real OBC. O objetivo é verificar se o Subsistema Real OBC não apresenta falhas de interoperabilidade no barramento (HIL) com o subsistema PL simulada.

Havendo falha, analisa-se qual subsistema apresentou a falha. Se a falha é devido a problema no subsistema simulado, é analisado novamente o subsistema em teste corrigindo a falha detectada. Se a falha é devido ao subsistema real, o desenvolvedor é contactado.

5.3.5. **Módulo 5: Validação de interoperabilidade entre os subsistemas reais**

Após a validação separada dos subsistemas reais com auxílio dos subsistemas simulados, neste módulo o objetivo é verificar se os subsistemas reais implementaram os requisitos de interoperabilidade e robustez conforme especificado por meio de testes de interação no barramento de comunicação (HIL).

Neste procedimento que compõe o módulo 5, é verificado se os subsistemas reais interagem conforme os modelos comportamentais desenvolvidos e verificados nos módulos 1, 2 e 3, sob condição nominal, interconectados conforme o protocolo de comunicação do Cubesat.

Em relação ao conjunto de teste, os casos de teste de interoperabilidade já especificados nos módulos 1, 2 e 3 são reutilizados e testados, reduzindo o custo e esforço na fase de integração.

Havendo falha, é contactado o desenvolvedor do subsistema que apresentou falha.

5.3.6. **Módulo 6: Validação de robustez entre os subsistemas reais**

Neste módulo 6, o objetivo é testar e validar os requisitos de robustez dos subsistemas reais injetando estímulos de falhas em ambos os subsistemas por meio de um Mecanismo Emulador de Falhas (FEM). Os subsistemas reais geralmente são desenvolvidos separadamente pelos seus fabricantes e precisam ser testados em interoperabilidade nas situações nominais e de exceção.

Nos módulos anteriores as falhas são injetadas por meio dos modelos simulados. Neste módulo, para testar a robustez dos subsistemas reais é utilizado o FEM conectado entre os subsistemas reais no barramento de comunicação. Desta forma as informações trocadas entre os subsistemas OBC

e PL passam pelo FEM, permitindo-o assim injetar as falhas que anteriormente faziam parte dos modelos simulados.

Esse emulador de falhas tem um papel importante nessa evolução porque as injeções de falhas não são intrusivas nos subsistemas comunicantes, testando a capacidade de cada um dos subsistemas de resistir a falhas externas.

O FEM atua da seguinte forma: i) recebe informação de um determinado subsistema, o qual está enviando uma informação para o outro subsistema em interação, ii) injeta uma determinada falha nessa informação, iii) retransmite essa informação alterada para o subsistema destino.

Esse mecanismo permite a verificação da interação dos subsistemas sob condições de exceção que são especificadas. As condições de exceção são implementadas manualmente ou por meio de uma base de dados denominada **BD Estímulo** contento uma sequência de casos de teste especificados.

Desta forma quando o FEM recebe uma informação do OBC, é pesquisada sequencialmente na base de dados **Estímulo** a primeira informação contendo o estímulo de falha a ser enviada do OBC para uma determinada PL. Posteriormente na sequência são enviadas as outras informações contendo outros estímulos de falhas dessa base de dados conforme os procedimentos de envio das informações do OBC. Estes procedimentos são recíprocos para enviar informações de uma PL para o OBC.

O FEM também é configurado permitindo que receba informações do OBC sem necessidade de receber da PL uma informação de retorno, assim como o FEM pode receber informações de uma PL sem necessidade que o OBC envie informações de retorno.

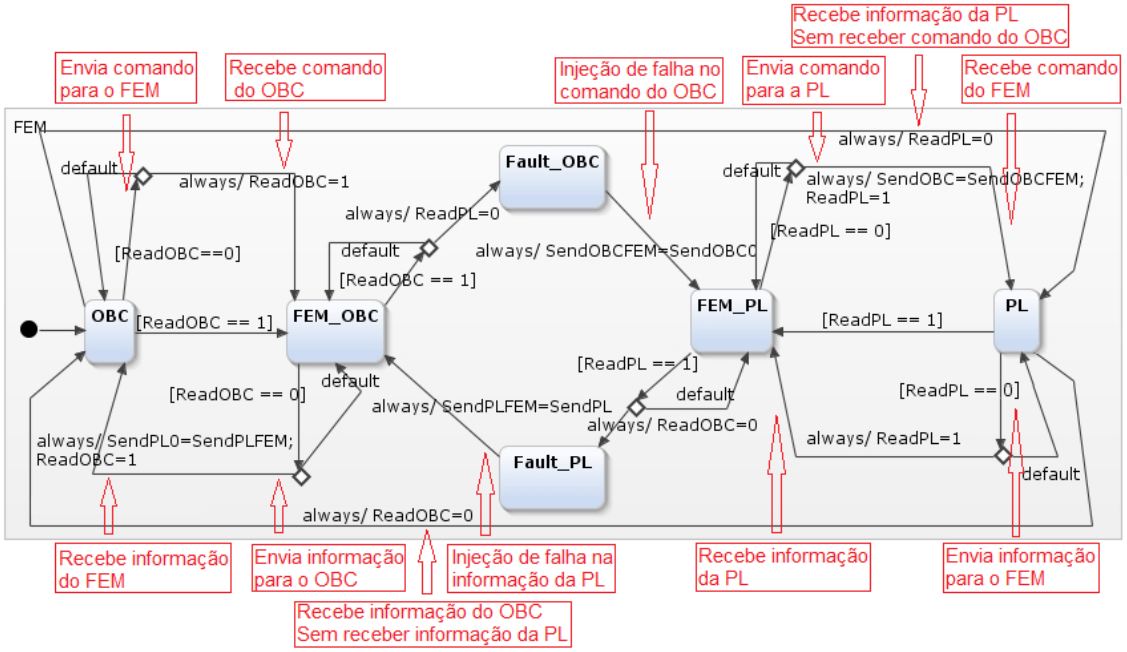
O emulador de falhas é uma placa Arduino com o protocolo de comunicação do Cubesat configurada manualmente da seguinte forma: i) duas portas (SDA e SCL) da placa são configuradas para estarem conectadas a um subsistema, e ii) outras duas portas (outras SDA e SCL) estão configuradas para estarem conectadas ao outro subsistema. Desta forma o FEM é o interconector entre os subsistemas.

Na modelagem do FEM os comandos recebidos pelo subsistema OBC são por meio da variável denominada *SendOBC0,* a qual está configurada para receber as informações pelo protocolo de comunicação. No conteúdo dessa variável é injetado falha e é transferido para a variável denominada *SendOBCFEM*. O conteúdo dessa última variável é transferido para a variável denominada *SendOBC*, o qual é transmitido para uma PL. As informações recebidas do subsistema PL são por meio da variável denominada *SendPL*, a qual está configurada para receber as informações pelo protocolo de comunicação. No conteúdo dessa variável é injetado falha e é transferido para a variável denominada *SendPLFEM*. O conteúdo dessa última variável é transferido para a variável denominada *SendPL0*, o qual é transmitido para o subsistema OBC. A variável denominada *ReadOBC* determina se o FEM irá ler um comando do OBC (=1) ou se o comando já foi lido e não há necessidade de nova leitura (=0). Da mesma forma a variável denominada *ReadPL* determina se o FEM irá ler uma informação de uma PL (=1) ou se a informação já foi lida e não há necessidade de nova leitura (=0).

Nos estados *Fault_OBC* e *Fault_PL* ocorrem às execuções de injeção de falhas. As injeções de falhas são realizadas uma a uma manualmente ou por meio da leitura dos estímulos de falhas registras na Base de Dados Estímulo.

Na modelagem do FEM os seguintes estados são definidos: *OBC* (porta conectada ao subsistema OBC, cuja porta recebe os comandos enviados por esse subsistema ou envia informações do FEM para o OBC (SendPL0]), ii) *FEM_OBC* (recebe o comando do OBC [SendOBC0] e verifica qual injeção de falha será ocasionada, ou envia a informação recebida de uma PL com injeção de falha [SendPLFEM]) para o OBC, iii) *Fault_OBC* (injeta falha no comando do OBC [SendOBCFEM]), vi) *FEM_PL* (direciona a falha para uma PL [SendOBCFEM para SendOBC], ou recebe a informação de uma PL [SendPL] e verifica qual injeção de falha será ocasionada), v) *PL* (porta conectada ao subsistema de uma PL, cuja porta transmite a falha para esse subsistema ou envia informação da PL para o FEM [SendPL], e vi) *Fault_PL* (injeta falha na informação de uma PL [SendPLFEM]). A Figura 5.18 apresenta a modelagem do FEM.

Figura 5.18 – Modelo FEM.



Fonte: Produção do autor.

Este modelo do FEM também permite ser utilizado com os modelos dos subsistemas validados por simulação para testar a robustez desses subsistemas em interação.

5.3.6.1. Avaliação da robustez entre os subsistemas reais

Não ocorrendo falhas ou ocorrendo falhas, provocadas pelos estímulos de injeção de falhas, e essas falhas forem identificadas na planilha de dependabilidade, os subsistemas reais testados são considerados validados.

No caso da execução ocorrer conforme as especificações dos requisitos, os modelos são considerados validados. Caso contrário, há necessidade de nova análise dos requisitos e realização novamente dos procedimentos para nova evolução dos testes. O desenvolvedor do subsistema que apresentou falha é contactado.

As falhas identificadas são relacionadas na planilha de dependabilidade e na tabela FMEA.

6 APLICAÇÃO DA SISTEMÁTICA DE TESTES NO ESTUDO DE CASO NANOSATC-BR2

Neste Capítulo apresenta-se a aplicação da abordagem sistemática de teste, descrita no capítulo 4, uma missão de nanosatélite que utiliza o padrão Cubesat, denominada NanosatC-BR2, em desenvolvimento no Instituto Nacional de Pesquisas Espaciais (INPE, 2014). Os testes realizados são desenvolvidos utilizando os módulos 1, 2, 3, 4, 5 e 6 da abordagem apresentada. Esses módulos permitem apresentar, verificar e validar benefícios desta abordagem em um ambiente real. Nos módulos 4 e 6, um subsistema real é testado no loop (*Hardware-in-the-loop* - HIL). O módulo 5 não é apresentado porque as cargas úteis estão em desenvolvimento. A experimentação é realizada no subsistema Computador de Bordo em interoperabilidade com uma carga útil simulada, pelo fato da carga útil real estar em desenvolvimento.

O subsistema real testado é o Computador de Bordo (OBC), sendo que o software é desenvolvido pela empresa EMSISTI e o hardware pela empresa ISIS. A sistemática de teses adotada e executada na arquitetura STAE seguem a sequência evolutiva de testes conforme apresentado anteriormente na Figura 5.3.

6.1. **Análise do ambiente de testes dos subsistemas do sistema espacial**

Inicialmente foram analisadas as normas de requisitos de segurança para realizar operações em solo e voo (ABNT NBR ISO 14620-1, 2 e 3, 2009), conceitos de dependabilidade e segurança no desenvolvimento de software de sistemas (ECSS-Q-HB-80-03, 2012), abordagens de interoperabilidade e robustez (MATTIELLO-FRANCISCO, 2009), bem como a documentação do nanosatélite NanosatC-BR2 (INPE, 2011), o qual é o Cubesat em desenvolvimento no INPE.

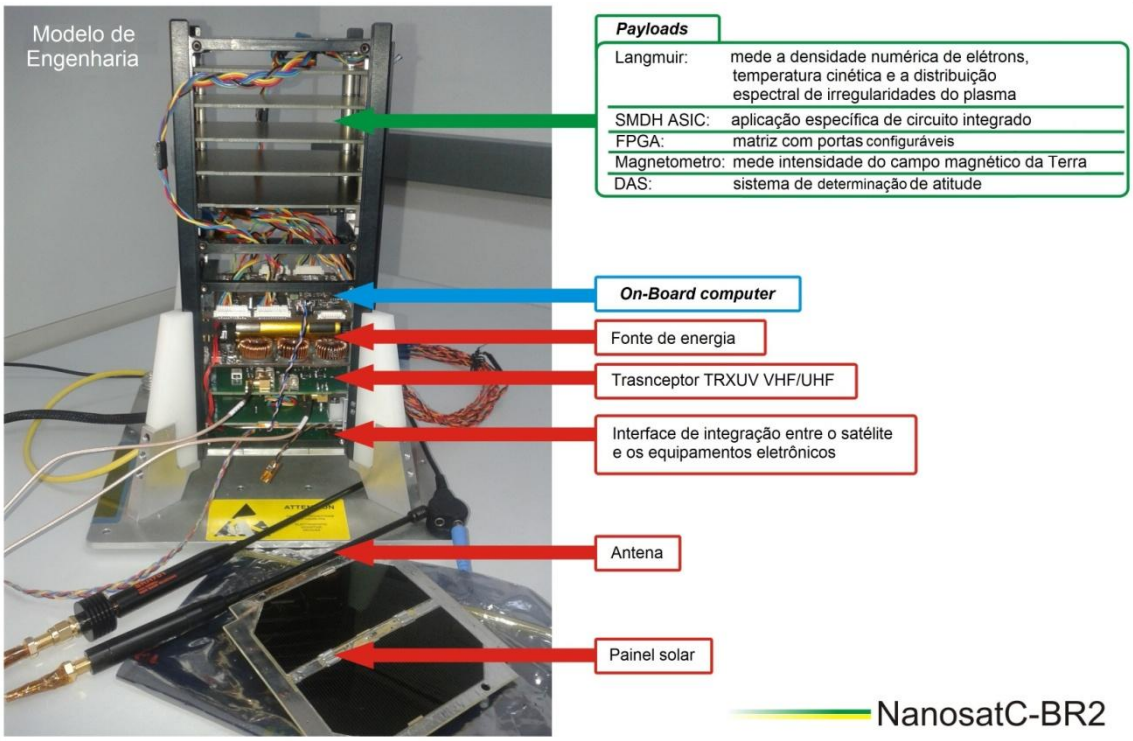
6.2. **Estudo de caso NANOSATC-BR2**

O Instituto Nacional de Pesquisas Espaciais (INPE) vem desenvolvendo pesquisas em plataformas de nanosatélites buscando adquirir conhecimento no padrão Cubesat e desenvolver novas tecnologias espaciais que possam ser qualificadas em órbita a baixo custo para atender necessidades de âmbito nacional. Atualmente o INPE, em parceria com universidades federais Brasileiras, desenvolve um Cubesat 2U, denominado NanosatC-BR2 (SCHUCH et al., 2017), cuja missão é coletar dados de campo magnético da Terra e testar em voo a resistência de circuitos à radiação entre outras novas tecnologias projetadas no Brasil.

6.2.1. **Arquitetura do NANOSATC-BR2**

A arquitetura do NanosatC-BR2 é composta pelos seguintes subsistemas: i) Computador de bordo (*On-Board Computer* - OBC), ii) Fonte de energia, iii) Transceptor TRXUV VHF/UHF, iv) Interface de integração entre o satélite e os equipamentos eletrônicos, v) Sistema de antenas, vi) Painel solar, entre outros (SISTEMAS ESPACIAIS E TECNOLOGIA, 2015). Suas cargas úteis (*Payloads*) são: i) Sonda de Langmuir (mede a densidade numérica de elétrons, temperatura cinética e a distribuição espectral de irregularidades do plasma), ii) SMDH ASIC (aplicação específica de circuito integrado) , iii) FPGA (matriz com portas configuráveis), iv) Magnetometro (mede intensidade do campo magnético da Terra), e v) DAS (Sistema de determinação de atitude) (ISIS, 2011). A Figura 6.1 apresenta a arquitetura do NanosatC-BR2, sendo que suas cargas úteis não estão integradas no satélite.

Figura 6.1 – Arquitetura do NanosatC-BR2.



Fonte: Produção do autor.

A interoperabilidade entre os subsistemas do satélite incluindo as cargas úteis é feito por meio do barramento de comunicação I2C.

6.2.2. **Arquitetura de testes STAE**

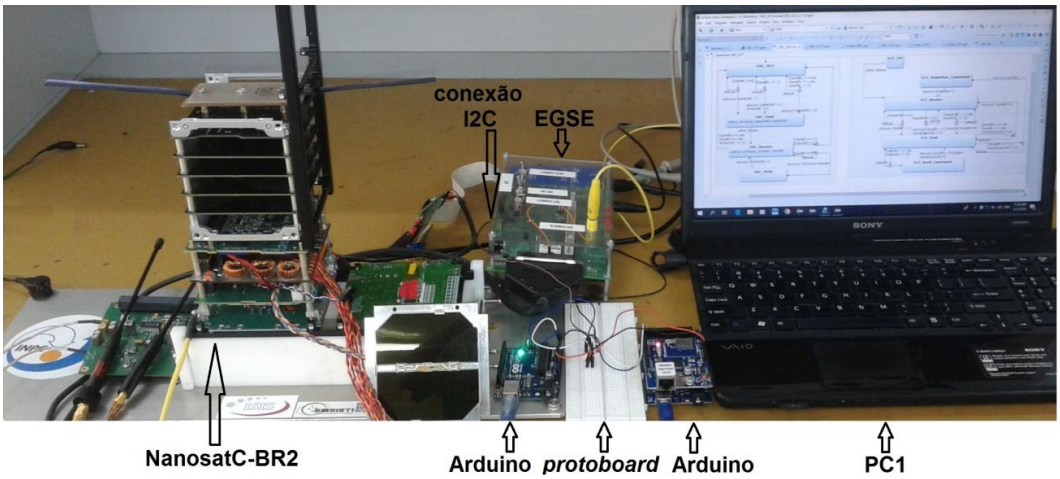
A arquitetura de testes STAE é configurada da seguinte forma:

- Inicialmente as placas computacionais Arduino, contendo os códigos computacionais que simulam o comportamento do software embarcado nos subsistemas reais em interação do satélite, são conectadas por meio de uma placa denominada *protoboard*.
- Posteriormente as placas dos subsistemas reais são conectadas às placas Arduino por meio da *protoboard* antes de haver a integração no satélite.

- Havendo a validação dos subsistemas reais, esses são integrados ao satélite.
- O EGSE é conectado ao barramento de comunicação do satélite.
- As placas Arduino são conectadas ao barramento do satélite por meio do EGSE.

A Figura 6.2 apresenta a arquitetura de teste contendo as placas Arduino conectadas ao satélite por meio do EGSE. O satélite está parcialmente integrado.

Figura 6.2 – Arquitetura de teste.



Fonte: Produção do autor.

Esta arquitetura permite executar a evolução dos testes para a abordagem apresentada nesta tese.

6.2.3. **Evolução dos testes**

A aplicação da abordagem apresentada abrange a evolução dos testes baseados nos requisitos de comportamento nominal funcional dos subsistemas intensivos em software do Computado de Bordo (OBC) e da carga útil (PL), denominada *Sonda de Lagmuir* (SLP) em interação do NanosatC-BR2, nos testes de simulação e integração. O objetivo é identificar falhas no início do

processo de V&V desses subsistemas evitando retrabalho em fases adiantadas do desenvolvimento da missão. A aplicação da abordagem sistemática de testes segue a implementação dos 6 módulos descritos no capítulo 4 como segue:

6.2.3.1. Módulo 1

Alguns requisitos básicos nas atividades de interação entre os subsistemas OBC e PL (SLP) analisados de [ALMEIDA, 2016] são relacionados a seguir: i) o OBC deve enviar comandos para a SLP solicitando informações de operações e dados adquiridos pelo experimento SLP conforme programada, ii) os dados do experimento armazenados em um buffer da SLP devem ser lidos pelo OBC e gravados em sua área da memória para serem transmitidos à estação solo, iii) após a transferência de dados do buffer da SLP para a memória do OBC, a SLP deve realizar a aquisição de novos dados, e iv) a troca de informações entre esses dois subsistemas deve ser por meio do barramento de comunicação I2C.

6.2.3.2. Especificação dos requisitos

Os requisitos dos subsistemas OBC e SLP para construção dos modelos do comportamento nominal da comunicação OBC e SLP do NanosatC-BR2 foram levantados, considerando situações de falhas relacionadas a interoperabilidade em missões espaciais.

Os requisitos especificados para a modelagem do OBC foram: o subsistema OBC assume a função de mestre e seu comportamento nominal na interação com a carga útil SLP deve ser representado na seguinte sequência de atividades (etapas):

1. O OBC deve ser ligado remotamente via estação solo. No caso desse experimento, no Cenário A o OBC simulado é acionado por meio de uma plataforma de desenvolvimento do próprio Arduino, e

nos Cenários B e D o OBC real é ligado por uma plataforma de desenvolvimento elaborada pelo desenvolvedor do OBC.

2. As operações de serviço como *housekeeping* e envio de comandos caracterizam eventos, que determinam o início da etapa *IDLE*, sendo necessário definir qual a sequência dos comandos a serem enviados para a SLP.
3. OBC envia o comando (*SlaveReady*) (etapa *Send*) representando a pergunta se a SLP está pronta para troca de informações.
4. OBC envia o comando (*SetDataTime*) (etapa *Send*) representando que irá enviar data e hora do momento da interação do OBC com a SLP para adquirir novos dados do experimento da SLP.Todas as vezes que o OBC iniciar um ciclo para leitura dos dados do experimento será enviada uma nova data e hora.
5. OBC envia Dia, Mês, Ano, hora, minuto e segundo da interação (etapa *Send*).
6. OBC aguarda cinco minutos. Tempo referente à SLP realizar a aquisição dos dados do experimento. Os dados são armazenados no buffer da SLP em 300 pacotes de 100 Bytes cada.
7. Após os cinco minutos, o OBC envia novamente o comando (*SlaveReady*) (etapa *Send*).
8. OBC envia o comando (*SetBloTx*) (etapa *Send*) representando que será informado o número de blocos de dados (pacotes) do experimento da SLP que deverá ser lido. Para a apresentação desta tese não será representado a leitura dos 300 pacotes de dados.
9. OBC envia o número de bloco de dados (etapa *Send*).
10. OBC envia o comando (*SetBloIndex*) (etapa *Send*) representando que será informado o número do índice do bloco de dados do experimento da SLP que deverá ser lido.
11. OBC envia o número do índice do bloco de dados (etapa *Send*).

12. OBC envia o comando (*StartTx*) (etapa *Send*) representando que será solicitada a leitura dos dados do experimento da SLP.
13. OBC recebe (etapa *Receive*) as informações enviadas pela SLP e os dados do experimento. Os dados enviados são a partir do índice e número de blocos solicitados.
14. OBC grava (etapa *Write*) os dados recebidos.
15. OBC retorna ao estado (etapa *IDLE*), o qual irá informar nova data e hora para a SLP.
16. OBC constantemente envia comando (etapa *Send*) verificando se a SLP está pronta (*SlaveReady*) para receber comandos ou ocupada (*Busy*). OBC também verifica se as informações recebidas pela SLP foram reconhecidas (*CmdOK*) ou não (*CmdUnk*).

Os requisitos especificados para a modelagem da SLP foram: O subsistema carga útil SLP assume a função de escravo e seu comportamento nominal na interação com o subsistema OBC deve ser representado na seguinte sequência de atividades (etapas):

1. A SLP deve ser ligada (etapa *ON*). No caso desse experimento a SLP simulada, uma plataforma de desenvolvimento do próprio Arduino é inicializada.
2. SLP aguarda receber o envio de data e hora fornecida pelo OBC para iniciar aquisição dos dados do experimento (etapa *Receive*).
3. SLP fica em estado ocupada (*Busy*) durante a aquisição dos dados (etapa *Acquisition Experiment*). O tempo estipulado é cinco minutos.
4. SLP envia o comando (*OK*) comunicando que está disponível para enviar os dados somente quando acabar a aquisição (etapa *Send*).
5. SLP aguarda a solicitação do OBC para leitura dos dados.
6. Após a solicitação do OBC para leitura dos dados, a SLP lê os dados do experimento (etapa *Read Experiment*), conforme

solicitados pelo OBC (quantidade de bloco de dados a serem lidos e a partir de que índice do buffer da SLP.

7. Após a SLP realizar a leitura dos dados solicitados pelo OBC, a SLP envia esses dados para o OBC (etapa *Send*).
8. Após envio dos dados, a SLP volta ao estado aguardando data e hora (etapa *Receive*), fornecida pelo OBC, antes de dar início à aquisição de novos dados do experimento.
9. SLP sempre que solicitada informa (etapa *Send*) se os comandos recebidos pelo OBC foram reconhecidos (*CmdOK*) ou não (*CmdUnk*).

Na interação entre o OBC e a SLP algumas outras condições excepcionais ocorrem definindo a sequência das atividades. Algumas dessas condições estão relacionadas na Tabela 6.1.

Tabela 6.1 - Condições para a realização da interação entre OBC e SLP.

Condições
• O OBC deve perguntar se a SLP está pronta para receber comandos (*SlaveReady*) até receber da SLP a informação que está *OK* (*SlaveOK*);
• As etapas devem ser realizadas na ordem sequencial, porém se algum comando enviado pelo OBC houver problema no recebimento pela SLP o OBC deve retornar para reenviar o primeiro comando e a PL deve enviar a informação que não reconheceu o comando (*CmdUnknown*) e deve retornar a ler primeiro comando do OBC;
• Os seguintes comandos enviados pelo OBC não devem receber confirmação de recebimento pela SLP: DD, MM, AA, hh, mm, número de blocos e índice do número de blocos;
• Quando o OBC enviar o sexto byte referente ao segundo (ss) do tempo a SLP deve informar que recebeu Data e Hora (*DTOK*), na sequência o OBC e a SLP devem realizar as próximas etapas;
• A SLP deve enviar os dados do experimento para o OBC conforme o índice e número de blocos dos dados solicitados.
• O OBC deve gravar os dados do experimento em sua área de armazenamento.
• Quando a SLP terminar de enviar os dados para o OBC a SLP deve voltar ao estado *SlaveOK* retornando a ler Data e Hora e o OBC deve enviar nova Data e Hora.

Fonte: Produção do autor.

Os comandos e informações enviados e recebidos na interação entre o OBC e a SLP são em hexadecimal, porém nos experimentos realizados nesta tese utilizou-se decimal para facilitar o comparativo de grandeza. A Tabela 6.2 apresenta a relação dos comandos e informações em hexadecimal e decimal.

Tabela 6.2 - Comandos e informações em hexadecimal e decimal.

Comando ou Informação	Hexadecimal	Decimal
SlaveReady	0xf0	240
SetDataTime	0xf1	241
SetBloTx	0xf2	242
SetBloIndex	0xf3	243
StartTx	0xf4	244
OK	0xfa	250
Busy	0xfb	251
CmdOK	0xfc	252
CmdUnk	0xfd	253

Fonte: Produção do autor.

A Tabela A.1 apresentada no Apêndice A contém as informações da interação entre os subsistemas OBC e SLP. Essa tabela apresenta os comandos enviados pelo OBC, as informações enviadas pela SLP, as operações (ações) desses subsistemas, as condições e a sequência de atividades (*Steps*) realizadas para atenderem aos requisitos da modelagem comportamental dos subsistemas comunicantes em interação.

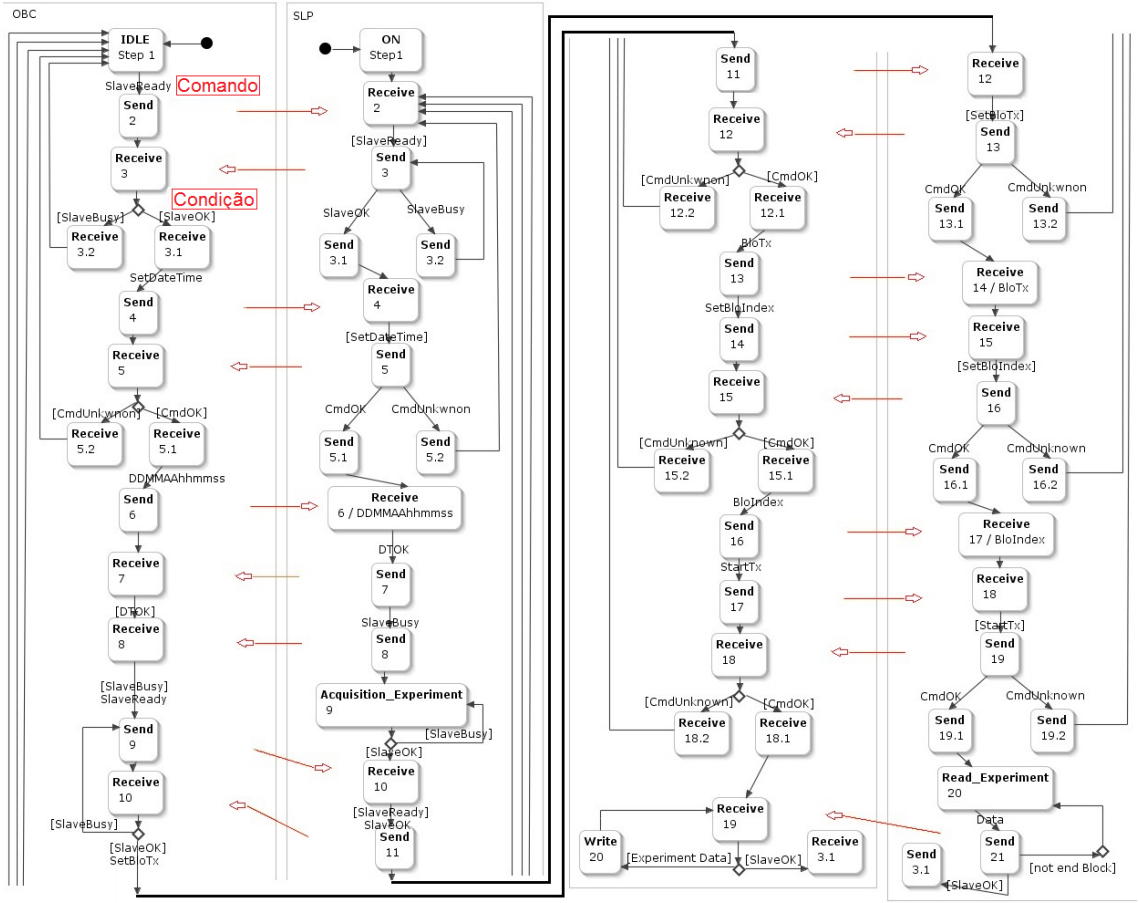
Realizado os levantamentos dos requisitos, das operações, das condições e das sequências de atividades a serem realizadas entre os subsistemas comunicantes, são realizados os diagramas de atividades. Esses diagramas permitem uma visualização gráfica das atividades dos subsistemas OBC e SLP em interação.

6.2.3.3. Diagrama de atividades

A utilização de uma representação gráfica para representação dos requisitos auxilia na visualização do fluxo das informações. Para a representação gráfica das atividades (etapas) (*Steps*) executadas entre o OBC e a SLP em comportamento nominal é utilizado o diagrama de atividades.. Essas atividades

são representadas por retângulos ou quadrados e as transições entre essas atividades são representadas por setas. As setas longas (vermelhas) indicam em qual etapa um subsistema envia informação para o outro nos momentos de interação. Os comandos estão identificados nas transições entre cada etapa. As condições estão identificadas após os símbolos no formato de losango. A Figura 6.3 apresenta os diagramas de atividades do OBC e da SLP.

Figura 6.3 – Diagrama de atividades do OBC e da SLP em interação.

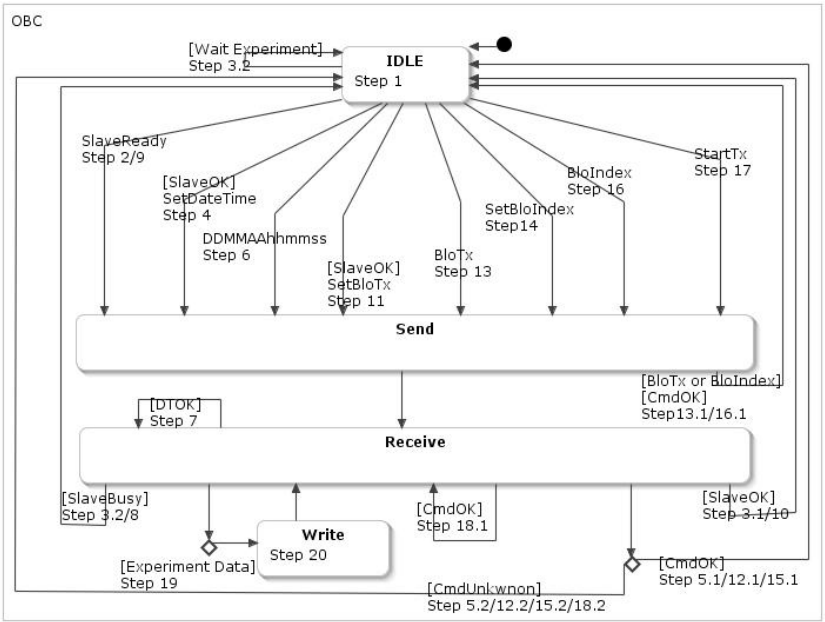


Fonte: Produção do autor.

Nos diagramas de atividades do OBC e da SLP as atividades *Send* representam que o subsistema envia comandos ou informação para o outro subsistema. As atividades *Receive* representam que o subsistema recebe informações. Tendo como foco a interação entre o OBC e a SLP, esses diagramas podem ser reestruturados sendo representado apenas uma

atividade de *Send* e uma de *Receive* identificando as atividades de entradas e saídas em cada um dos diagramas. A Figura 6.4 apresenta o diagrama de atividade do OBC reestruturado.

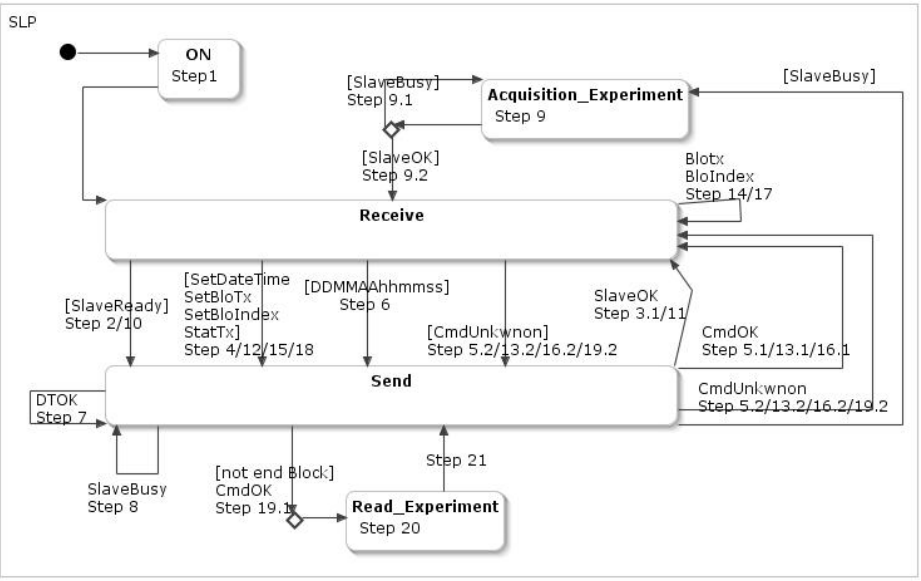
Figura 6.4 – Diagrama de atividades do OBC reestruturado.



Fonte: Produção do autor.

A Figura 6.5 apresenta o diagrama de atividade da SLP reestruturado.

Figura 6.5 – Diagrama de atividades da SLP reestruturado.



Fonte: Produção do autor.

A interação entre o OBC e a SLP é feita pelos estados *Send* e *Receive* de ambos os diagramas.

6.2.3.4. Modelagem e simulação do comportamento nominal

Utilizando os diagramas de atividades reestruturados, que foram elaborados baseados nos requisitos é realizada na ferramenta Uppaal a modelagem da Máquina de Estado do comportamento nominal dos subsistemas OBC e SLP em interação. O objetivo é verificar por simulação (MIL) se os modelos construídos em Uppaal representam os requisitos comportamentais desses subsistemas comunicantes com foco nas interações.

As etapas representadas no diagrama de atividades são definidas como estado na modelagem realizada na Uppaal.

A sincronização entre as máquinas de estado na Uppaal é realizada utilizando os seguintes indicadores: i) identificador de canal seguido do caractere exclamação (!) é definido como *output* para escrever informação no canal, ii) identificador de canal seguido do caractere interrogação (?) é definido como

input para ler informação no canal. Ações de escrita e leitura no mesmo canal geram a sincronização.

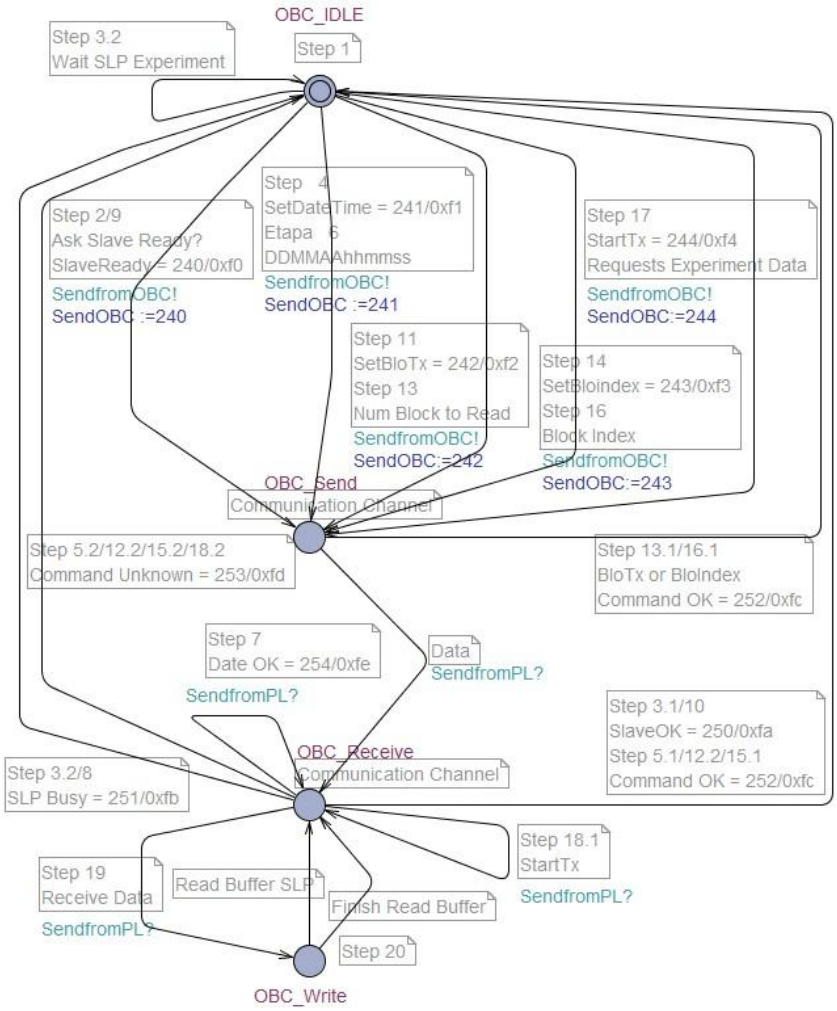
Na abordagem apresentada a modelagem dos subsistemas OBC em interação com a PL o sincronismo é realizado por meio dos identificadores SendfromOBC! para o OBC enviar comandos à SLP e SendfromPL? para receber informações. Assim como, SendfromPL! para a PL enviar informações e SendfromOBC? para receber.

Na realização da modelagem apresentada os estados das máquinas do OBC são acrescidos com OBC para diferenciar dos estados da máquina da SLP, assim como a máquina da SLP os estados também são acrescidos com SLP.

Os comandos e informações na modelagem deste trabalho são por opções alteradas para representações numéricas. Essas alterações são apenas para haver comparações numéricas e não alfanuméricas.

A Figura 6.6 apresenta o OBC modelado na ferramenta Uppaal representando as etapas para interagir com a SLP. O OBC em interação com a PL irá enviar os dados por meio de uma variável denominada *SendOBC* e a PL por meio da *SendPL*.

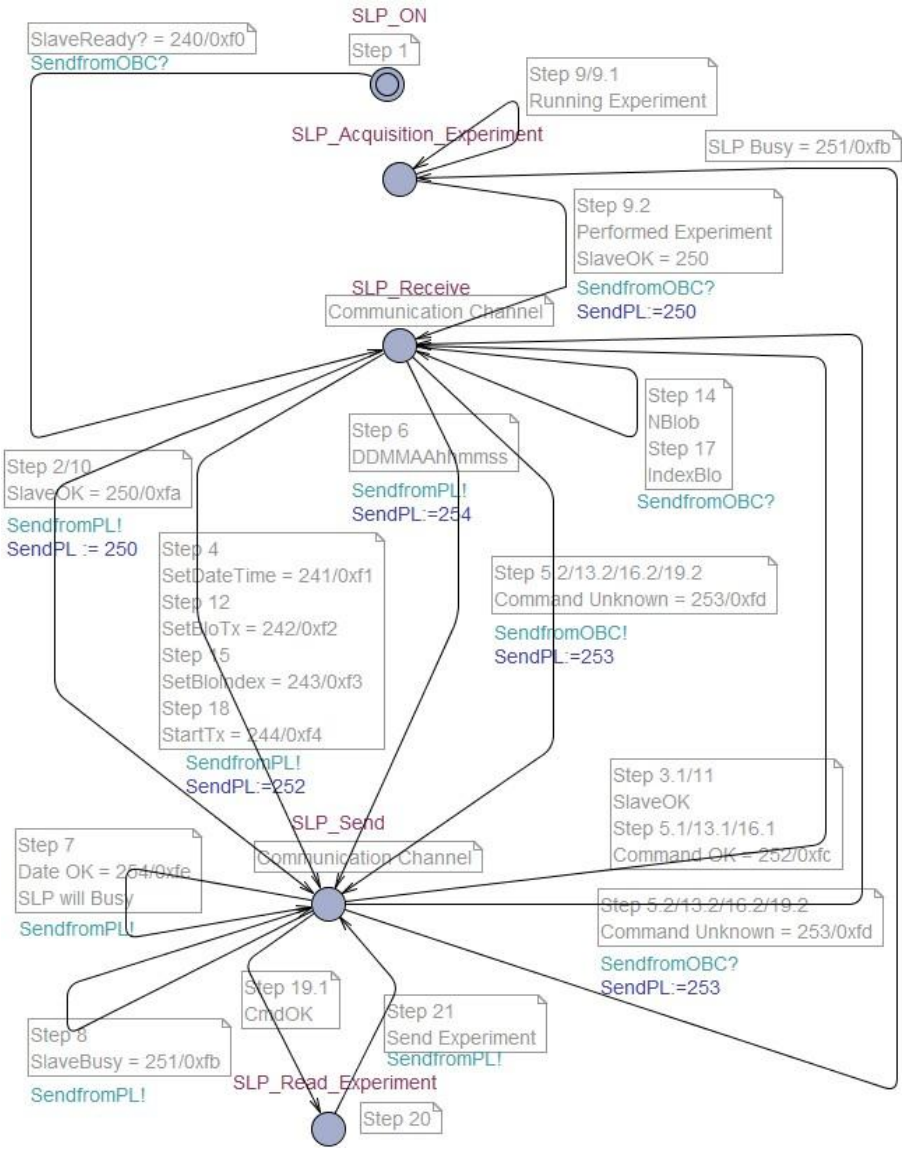
Figura 6.6 – Modelagem da máquina de estado do OBC.



Fonte: Produção do autor.

A Figura 6.7 apresenta a SLP modelada na ferramenta Uppaal representando as etapas para interagir com o OBC.

Figura 6.7 – Modelagem da máquina de estado da SLP.



Fonte: Produção do autor.

A modelagem realizada apresenta os estados e as transições das máquinas de estado, porém não apresenta as ações e as condições nas transições, as quais definem a sequência das etapas dos estados que são alcançados.

Pensando na execução das máquinas de estado em interação com a finalidade de simular o comportamento dos subsistemas, são definidas as variáveis para armazenamento de informações, transferências de dados, contadores de

indexações, entre outras informações necessárias para realizar a simulação das máquinas de estado modeladas em interação.

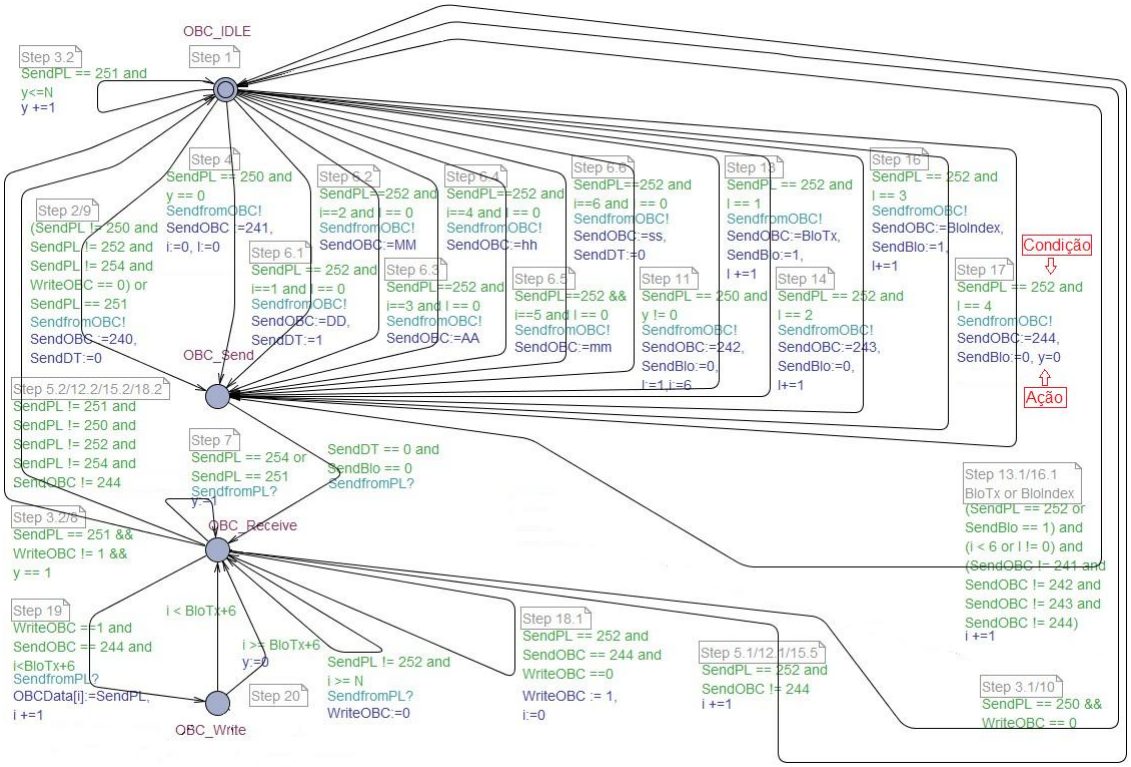
A Tabela A.2 apresentada no Apêndice A contém eventos, ações, condições e variáveis (em negrito) para realizar a modelagem do OBC interagindo com a SLP simulando o comportamento nominal desses subsistemas.

A Tabela A.3 apresentada no Apêndice A contém eventos, ações, condições e variáveis (em negrito) que compõem a modelagem da SLP interagindo com o OBC simulando o comportamento nominal desses subsistemas.

Utilizando a modelagem do OBC definida anteriormente juntamente com a Tabela 6.10, a qual relaciona os estados com os eventos, ações e condições, é possível incrementar as transições para complementar essa modelagem.

A Figura 6.8 apresenta a modelagem da máquina de estado do OBC complementada na ferramenta Uppaal. Os eventos recebidos pelos estados são por meio das variáveis denominadas *SendOBC* (comando a ser enviado pelo OBC) e *SendPL* (informação recebida pela SLP). As condições estão representadas na cor verde clara e as ações na cor azul escuro.

Figura 6.8 – Modelagem da máquina de estado do OBC para interagir com a SLP.

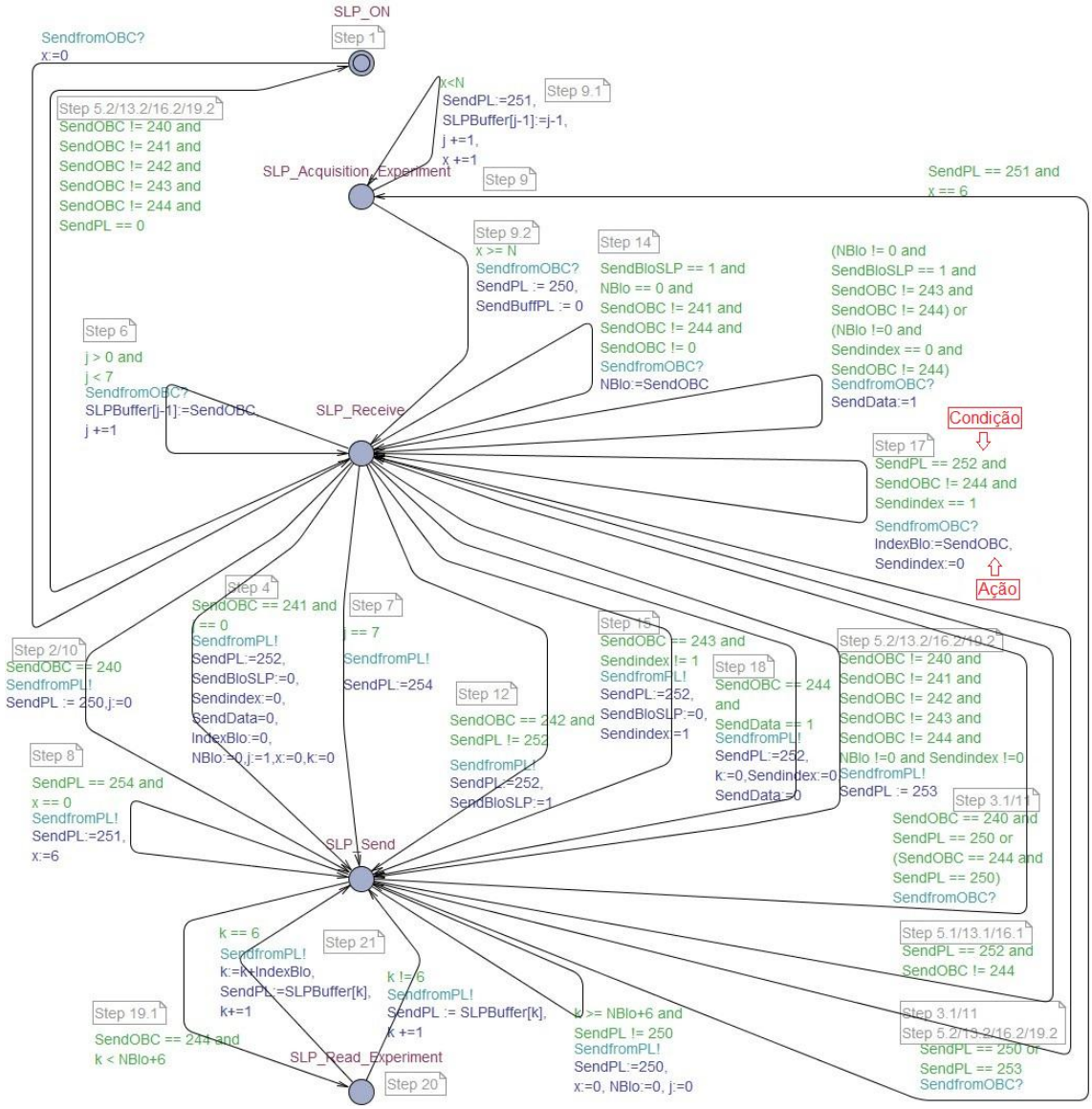


Fonte: Produção do autor.

Da mesma forma, utilizando a modelagem da SLP definida anteriormente juntamente com a Tabela 6.11, a qual relaciona os estados com os eventos, ações e condições, é possível incrementar as transições para complementar essa modelagem.

A Figura 6.9 apresenta a modelagem da máquina de estado da SLP complementada na ferramenta Uppaal. Os eventos recebidos pelos estados são por meio das variáveis denominadas *SendPL* (comando a ser enviado pela SLP) e *SendOBC* (informação recebida pelo OBC). As condições estão representadas na cor verde clara e as ações na cor azul escuro.

Figura 6.9 – Modelagem da máquina de estado da SLP para interagir com o OBC.

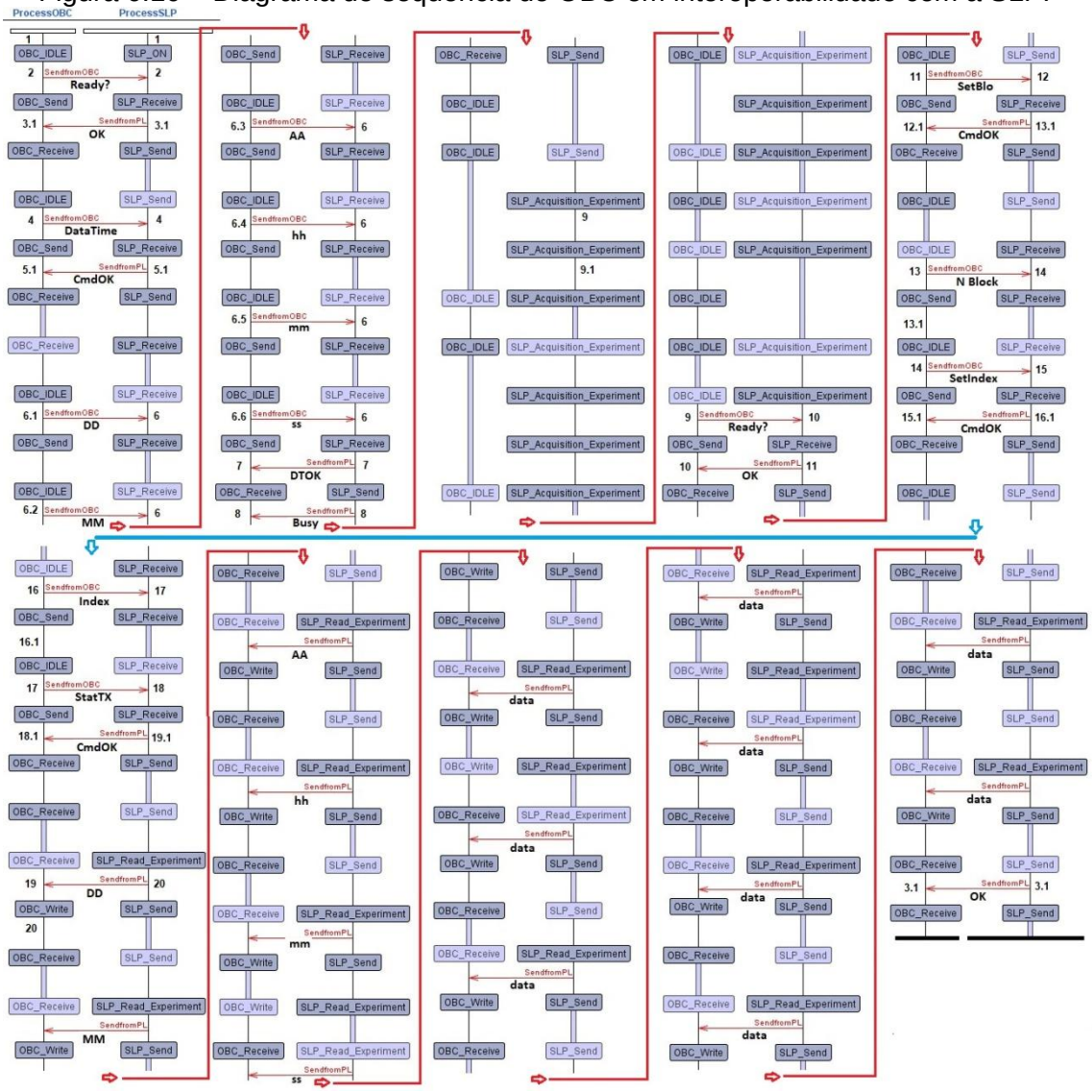


Fonte: Produção do autor.

A ferramenta Uppaal permite gerar um diagrama de sequência, a partir da simulação da modelagem do comportamento das máquinas de estado em interação, utilizando a opção *Simulator*. A Figura 6.10 apresenta o diagrama de sequencia gerado da interação entre o OBC e a SLP, destacando as informações trocadas entre esses subsistemas e o número das etapas que estão sendo realizadas. A troca de informações está representada por setas. As informações trocadas entre esses subsistemas e o número das etapas (*Steps*) foram acrescentadas manualmente para uma melhor visualização e

entendimento do que está ocorrendo em cada troca de informação. As setas longas vermelhas e azuis indicam o fluxo do diagrama.

Figura 6.10 – Diagrama de sequência do OBC em interoperabilidade com a SLP.



Fonte: Produção do autor.

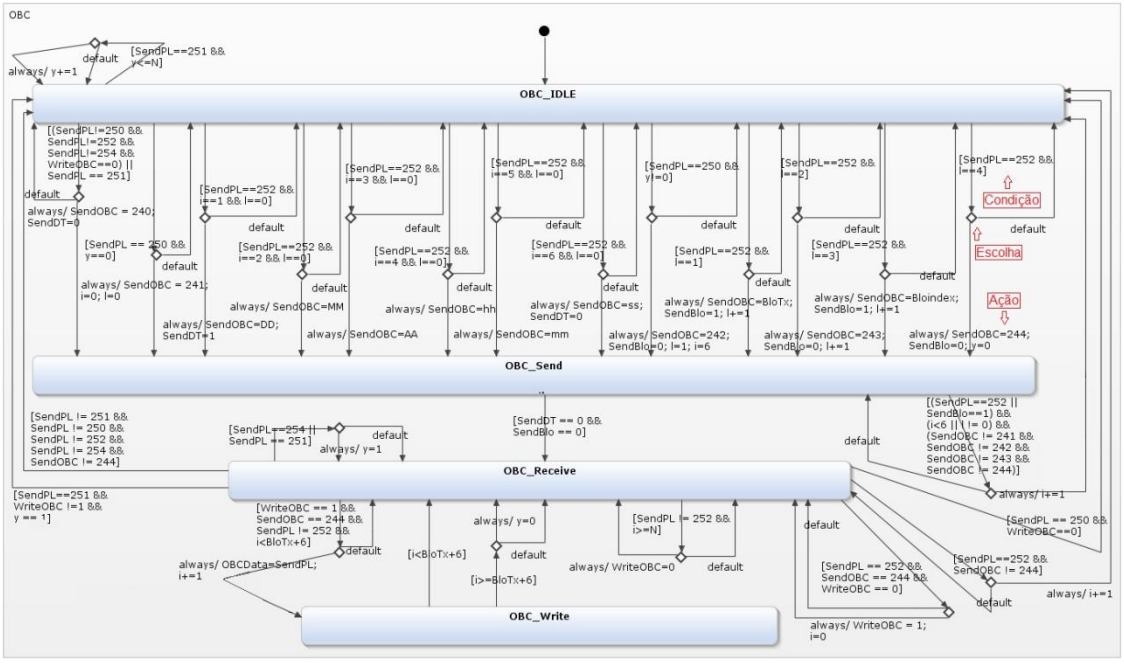
A troca das informações no diagrama de sequência está coerente com os diagramas de atividades que foram apresentados anteriormente na Figura 6.2 desenvolvidos baseados nos requisitos. Desta forma é verificado que a modelagem atende aos requisitos conforme definidos até o momento e, portanto a modelagem (MIL) foi validada.

6.2.3.5. Transformação de modelos

Os modelos validados na Uppaal são implementados na ferramenta Yakindu para gerar os códigos computacionais de forma automatizada. O objetivo é executar os códigos para simular os requisitos dos subsistemas comunicantes no barramento de comunicação

Os mesmos modelos definidos na Uppaal nas Figuras 6.7 e 6.8 são transferidos para a Yakindu. As alterações na Yakindu são referentes à: i) retirada do sincronismo definido na Uppaal, ii) utilização do recurso escolha (*choice*), iii) a condição está representada entre colchetes [ ], e iv) a ação está após o comando *always*. A Figura 6.11 apresenta a máquina de estado do OBC modelado na ferramenta Yakindu.

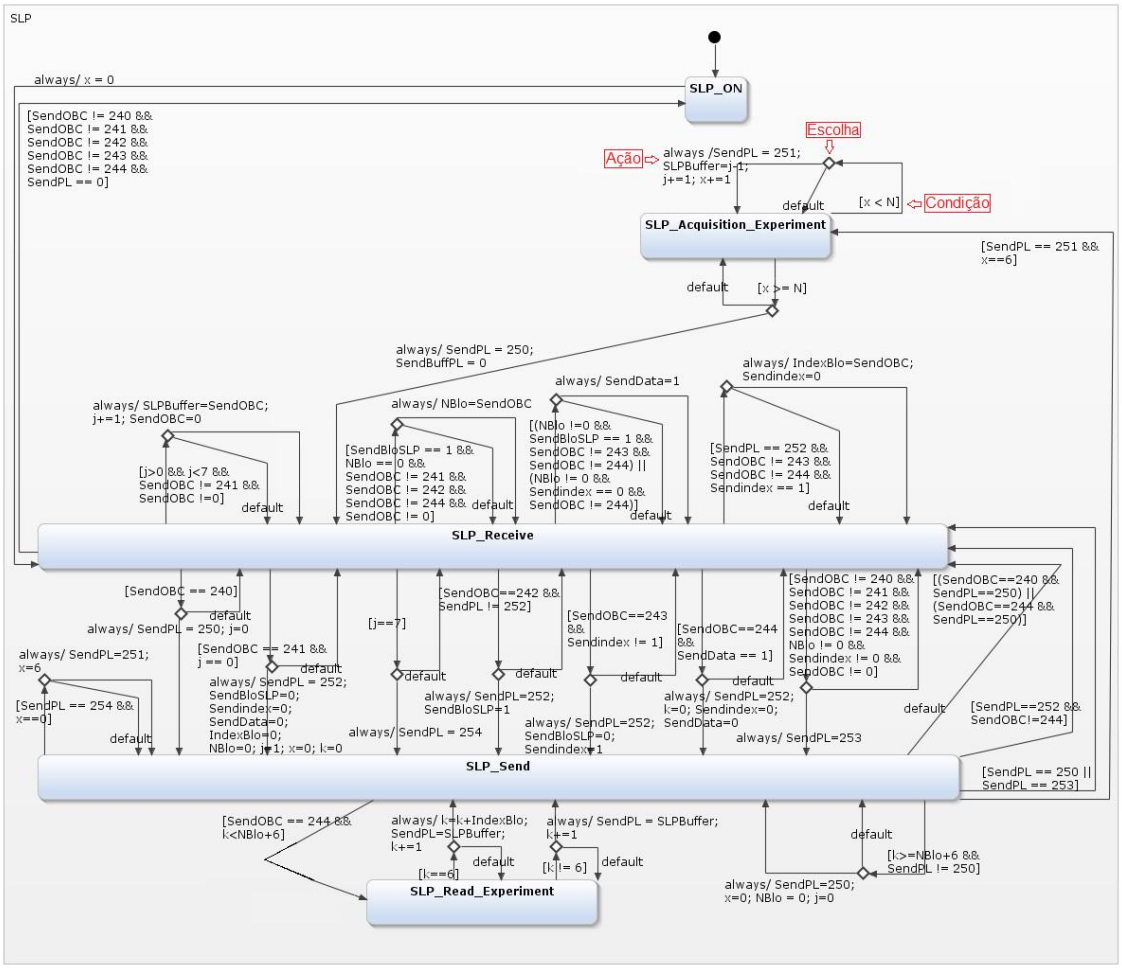
Figura 6.11 – Máquina de estado do OBC na Yakindu.



Fonte: Produção do autor.

A Figura 6.12 apresenta a máquina de estado da SLP modelado na ferramenta Yakindu.

Figura 6.12 – Máquina de estado da SLP na Yakindu.



Fonte: Produção do autor.

A Yakindu também permite simular e verificar o comportamento das máquinas de estado em interação verificando se há necessidades de alterações de requisitos nesta fase inicial do projeto.

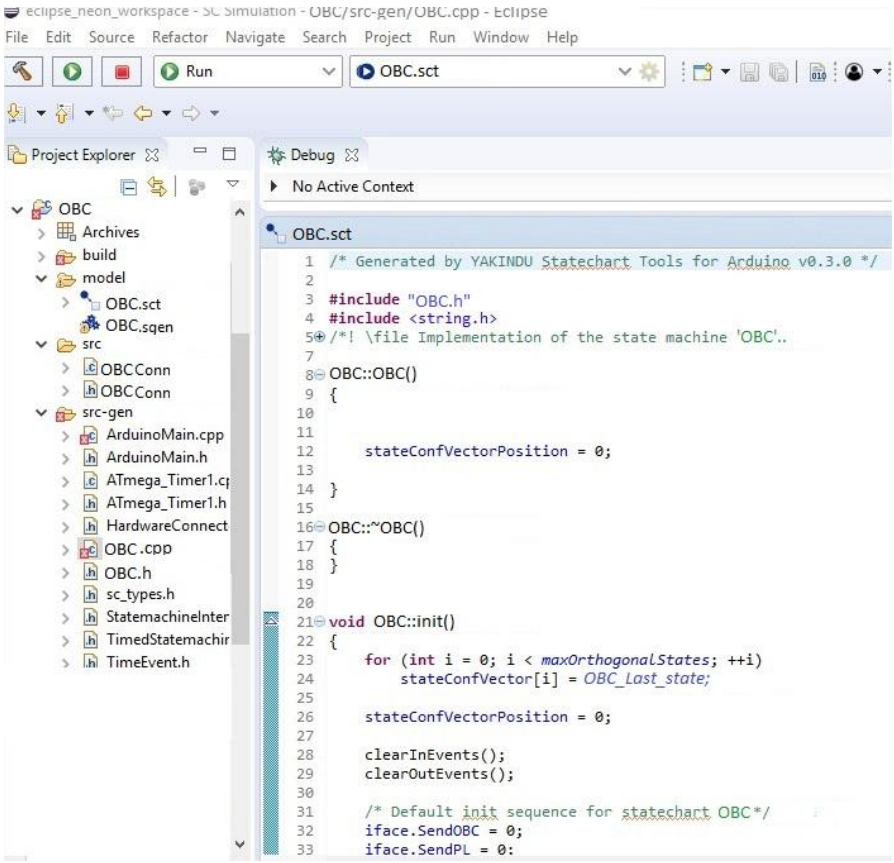
Neste momento as máquinas de estado não estão modeladas em sincronismo como foram modeladas na Uppaal. Isto permite verificar o comportamento dos

subsistemas trabalhando em paralelo independente um do outro quando não está havendo interação. Os códigos a serem gerados e embarcados em placas computacionais pela Yakindu neste trabalho trabalham em paralelo assim como os subsistemas reais quando não estão em interação.

6.2.3.6. Geração dos códigos computacionais

Utilizando a ferramenta Yakindu são gerados os códigos computacionais de forma automatizada dos modelos comportamentais dos subsistemas OBC e SLP em interação. O objetivo é embarcar os códigos em placas computacionais distintas para cada modelo e testar em ambiente de comunicação real. Os códigos são implementados nas placas Arduino, uma placa para cada subsistema. Nos arquivos gerados com extensão .cpp são implementados os comandos referentes ao protocolo de comunicação I2C. A Figura 6.13 apresenta alguns códigos gerados do arquivo OBC.cpp na Yakindu.

Figura 6.13 – Códigos gerados na Yakindu.



Fonte: Produção do autor.

As informações que trafegam no barramento I2C durante a interação do OBC com a SLP são armazenadas na Base de Dados MySQL denominada DB Interação. Essas informações são registradas em uma tabela denominada dt_i2c.

Para acesso a essa base de dados são implementados manualmente no arquivo OBC.cpp os comandos para realizarem a conexão com a base de dados MySQL e gravação dos dados gerados da interação entre os subsistemas comunicantes.

6.2.3.7. Execução dos códigos computacionais

Os códigos dos modelos comportamentais com foco nas interações entre os subsistemas OBC e SLP são testados em ambiente simulado. O objetivo é testar a interoperabilidade dos subsistemas modelados embarcados em ambiente simulado, mas que já faz uso do barramento de comunicação real (SIL).

Os resultados dos testes devem demonstrar que ambos os modelos interagirão conforme os requisitos especificados. Para averiguação desse resultado utiliza-se a base de dados Interação.

6.2.3.8. Base de dados Interação

As informações que trafegaram no barramento I2C durante a interação do OBC com a SLP modelados são armazenadas na base de dados Interação. Essas informações são registradas em uma tabela, a qual contém as seguintes informações: i) *id* (número sequencial de registros), ii) *timestamp* (data e hora de armazenamento da informação), iii) *OBC_send* (informação enviada pelo OBC, e iv) *PL_send* (informação enviada pela SLP). Os registros das informações transmitidas durante a interação entre o OBC e a SLP estão apresentados na Figura 6.14.

Figura 6.14 – Informações transmitidas no barramento I2C.

Base de Dados: db_nanosatc_br » Tabela: dt_i2c

SQL | Pesquisar | Insere

id	ts	OBC_send	PL_send
1	2018-07-24 11:20:10	240	250
2	2018-07-24 11:20:11	241	252
3	2018-07-24 11:20:12	24	0
4	2018-07-24 11:20:13	7	0
5	2018-07-24 11:20:14	18	0
6	2018-07-24 11:20:15	11	0
7	2018-07-24 11:20:16	20	0
8	2018-07-24 11:20:17	17	254
9	2018-07-24 11:20:18	0	251
10	2018-07-24 11:20:19	240	0
11	2018-07-24 11:25:20	240	250
12	2018-07-24 11:25:21	242	252
13	2018-07-24 11:25:22	1	0
14	2018-07-24 11:25:23	243	252
15	2018-07-24 11:25:24	0	0
16	2018-07-24 11:25:25	244	252
17	2018-07-24 11:25:26	0	3
18	2018-07-24 11:25:27	0	250

Fonte: Produção do autor.

O objetivo dessa base de dados é viabilizar o armazenamento das informações enviadas e recebidas pelos subsistemas em interoperabilidade dutrante os testes, para que a conformidade com o comportamento esperado possa ser verificada pós-teste.

Caso os resultados demonstrem que as informações de interação entre os subsistemas modelados OBC e SLP estão em conformidade com as especificações dos requisitos, considera-se validado o comportamento nominal desses subsistemas no barramento (SIL).

6.2.3.9. Módulo 2

A taxonomia de dependabilidade de (AVIZIENIS, 2004) foi utilizada no método desenvolvido para o desenvolvimento de uma planilha de dependabilidade e

uma tabela FMEA possibilitando classificar possíveis falhas na interação entre os subsistemas comunicantes, identificar possíveis falhas por dependabilidade, e relacionar mitigações para evitar as falhas identificadas.

6.2.3.10. Tratamento de falhas

Na abordagem apresentada nesta tese nem todos os tipos de falhas relacionados na planilha de dependabilidade são exploradas, pois o objetivo principal é apresentar a abordagem e não identificar todas as possíveis falhas entre os subsistemas que estão sendo modelados. Realizando uma nova análise dos requisitos do OBC tem-se:

- Um dos requisitos do OBC é ler os dados do experimento da SLP e armazenar esses dados em uma área de memória do OBC. Porém não está definido que ação é realizada se houver problema no armazenamento de dados no OBC. Dessa forma, na tabela de requisitos o elemento 9-Armazenamento associa a ação que deve ser realizada no caso de ocorrer problema durante o armazenamento de dados no OBC a um novo requisito. Essa ação deve ser incluída na modelagem e testada.
- O OBC realiza troca de informação com a carga útil por meio do canal de comunicação I2C. O protocolo I2C está sujeito a atraso ou até mesmo perda de informação durante a comunicação. Nas especificações dos requisitos não está definido que procedimento é realizado se o OBC não receber qualquer informação da carga útil por um dado período de tempo e durante quanto tempo deve insistir na troca de informação mesmo sem sucesso. Desta forma, na tabela de requisitos o elemento 10-Processamento associa a ação que deve ser realizada se o OBC não receber informação da carga útil e quanto tempo deve tentar a comunicação antes de seguir para outra atividade. Essa ação deve ser incluída na modelagem e testada.

A Tabela 6.3 apresenta a tabela de requisitos preenchida em negrito com esses elementos observados, os quais podem ser fonte de comportamentos indesejados.

Tabela 6.3 - Tabela de requisitos do OBC identificando elementos que podem ocasionar falhas.

Questionamento	**Elemento**	**Requisito**
Quem? (*Who*)	1-Ator	OBC
O que? (*What*)	2-Ação (verbo, desejo)	enviar (0xf0, 0xf1, 0fx2, 0xf3, 0xf4*, Date*, *Time*, *Num Block, Index*) e receber informações
Do que? (*Than*)	3-Objeto (necessidade)	coletar dados armazenados no buffer da SLP
Por quê? (*Why*)	4-Finalidade	SLP coletar novos dados
Por quem? (*Who*)	5-Alvo	SLP
Aonde? (*Where*)	6-Lugar da ação	área de armazenamento
Quando? (*When*)	7-Situação	interoperabilidade
De que maneira? (*How*)	8-Restrição	protocolo de comunicação
Limitações físicas (P*hysical limitations*)	9-Armazenamento	**área de armazenamento de dados**
Limitações de desempenho (*Performance limitations*)	10-Processamento	**canal de comunicação, perda de informação, tempo de comunicação entre mensagens**

Fonte: Produção do autor.

Realizando uma nova análise dos requisitos da SLP temos:

- Um dos requisitos da SLP é transferir os dados do experimento para o OBC. Porém não está definido que ação deve ser realizada se a SLP não conseguir ler os dados do buffer. Dessa forma, na tabela de requisitos o elemento 9-Armazenamento associa a ação que deve ser realizado se ocorrer problema durante acesso ao buffer da SLP a um novo requisito. Essa ação deve ser incluída na modelagem e testada.
- Da mesma forma que o OBC, a SLP realiza troca de informação com o OBC por meio do canal de comunicação I2C. Dessa forma, na tabela de requisitos o elemento 10-Processamento associa a ação que deve ser realizada se a SLP não receber informação do OBC a um novo requisito que especifica também quanto tempo o OBC deve tentar a comunicação antes de concluir que a SLP está defeituosa.

A Tabela 6.4 apresenta a tabela de requisitos preenchida em negrito com esses elementos observados, os quais podem ser fonte de comportamentos indesejados.

Tabela 6.4 - Tabela de requisitos da SLP identificando elementos que podem ocasionar falhas.

Questionamento	**Elemento**	**Requisito**
Quem? (*Who*)	1-Ator	SLP
O que? (*What*)	2-Ação (verbo, desejo)	enviar (0xfa, 0xfb, 0xfd, 0xfd, 0xfe, dados do experimento) e receber comandos
Do que? (*Than*)	3-Objeto (necessidade)	coletar dados do experimento e enviar estes dados para o OBC
Por quê? (*Why*)	4-Finalidade	esvaziar buffer para coletar novos dados
Por quem? (*Who*)	5-Alvo	OBC
Aonde? (*Where*)	6-Lugar da ação	buffer
Quando? (*When*)	7-Situação	interoperabilidade
De que maneira? (*How*)	8-Restrição	protocolo de comunicação
Limitações físicas (P*hysical limitations*)	9-Armazenamento	**buffer**
Limitações de desempenho (*Performance limitations*)	10-Processamento	**canal de comunicação, tempo de comunicação entre mensagens**

Fonte: Produção do autor.

Outra análise dos requisitos dos subsistemas em interação foi identificar possíveis falhas causadas por erros de software ou interferência física no barramento de comunicação.

Dentre as falhas de software, são selecionadas as seguintes para exemplificação e apresentação da abordagem deste trabalho:

1. Análise errônea dos requisitos.
2. Erros de codificação.
3. Definição insuficiente no dimensionamento das variáveis.
4. Subsistema não alcança o objetivo especificado.
5. Indefinição de comportamentos indesejados acarretando erro no fluxo do sistema.
6. Falha de comunicação.

Dentre as falhas de interferência físicas, são selecionadas as seguintes para apresentação da abordagem deste trabalho:

7. Protocolo de comunicação não ser iniciado pelos subsistemas.
8. As informações dos subsistemas não são enviadas no barramento.
9. Perda de informação no barramento.

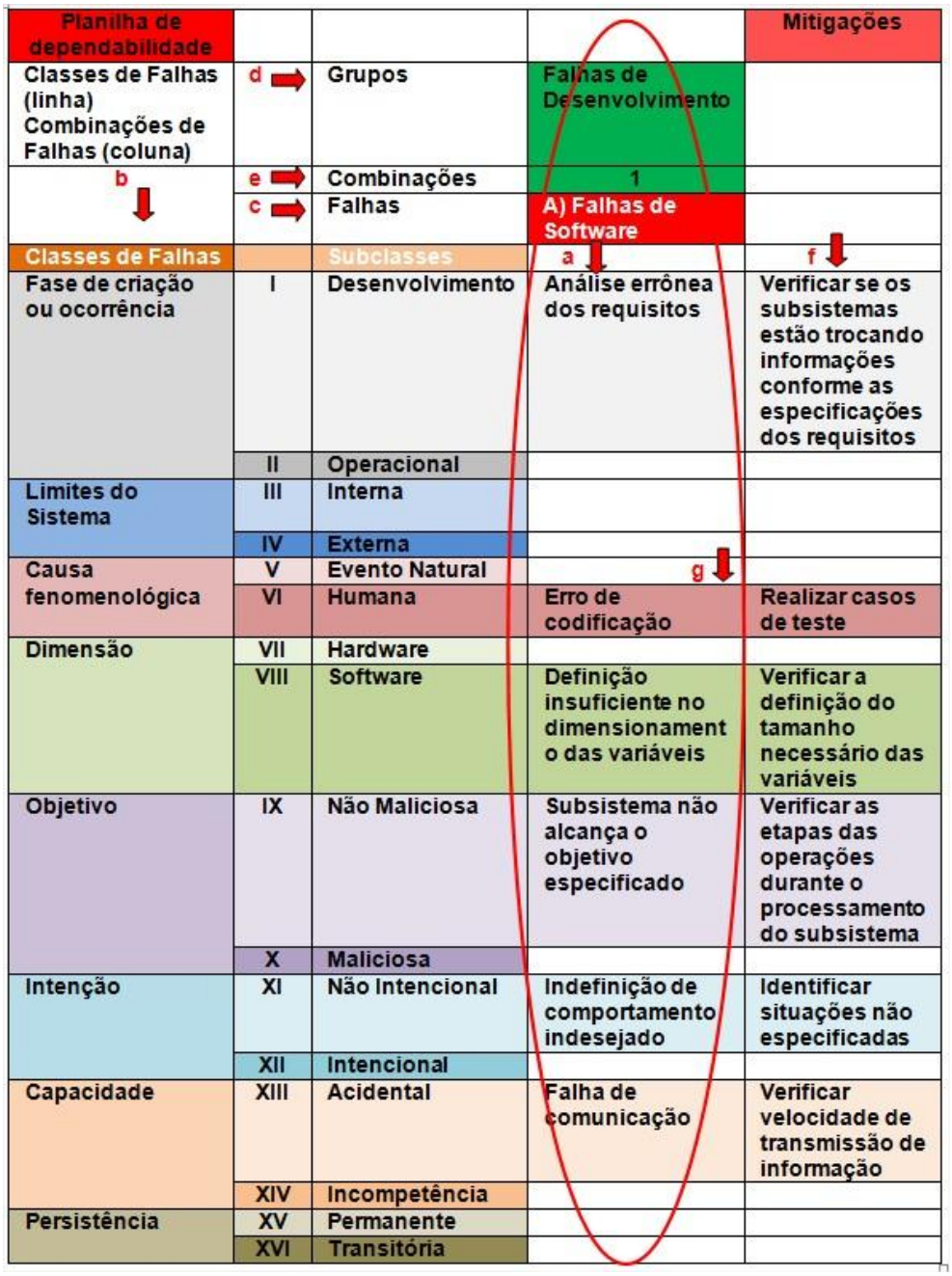
6.2.3.11. Planilha de dependabilidade

A planilha de dependabilidade é utilizada para relacionar as possíveis falhas identificadas na interação entre o OBC e a SLP com as respectivas mitigações. O objetivo é evitar as falhas identificadas na interação entre os subsistemas comunicantes utilizando as mitigações.

A Figura 6.15 cataloga apenas as primeiras colunas da planilha de dependabilidade com uma mitigação relacionada a cada possível falha analisada. Apenas a combinação de número 1 relacionada a falhas de software está representada nessa figura.

Essa planilha apresenta as seguintes informações: (a) possíveis falhas identificadas. Essas falhas são relacionadas com os seguintes itens: b) classe de falha, c) tipo de falha, d) grupo, e) combinação a que pertence, e f) mitigação para evitar a determinada falha. A coluna em destaque apresenta a possível correção da falha de uma linha (a) com a falha da linha seguinte (g) e também a associação dessa falha com o tipo de falha (c) e o grupo (d) a que pertence.

Figura 6.15 – Planilha de dependabilidade (apenas a combinação de número 1).



Planilha de dependabilidade				Mitigações
Classes de Falhas (linha) Combinações de Falhas (coluna)	d	Grupos	Falhas de Desenvolvimento	
b	e	Combinações	1	
	c	Falhas	A) Falhas de Software	
Classes de Falhas		Subclasses	a	f
Fase de criação ou ocorrência	I	Desenvolvimento	Análise errônea dos requisitos	Verificar se os subsistemas estão trocando informações conforme as especificações dos requisitos
	II	Operacional		
Limites do Sistema	III	Interna		
	IV	Externa		
Causa fenomenológica	V	Evento Natural	g	
	VI	Humana	Erro de codificação	Realizar casos de teste
Dimensão	VII	Hardware		
	VIII	Software	Definição insuficiente no dimensionamento das variáveis	Verificar a definição do tamanho necessário das variáveis
Objetivo	IX	Não Maliciosa	Subsistema não alcança o objetivo especificado	Verificar as etapas das operações durante o processamento do subsistema
	X	Maliciosa		
Intenção	XI	Não Intencional	Indefinição de comportamento indesejado	Identificar situações não especificadas
	XII	Intencional		
Capacidade	XIII	Acidental	Falha de comunicação	Verificar velocidade de transmissão de informação
	XIV	Incompetência		
Persistência	XV	Permanente		
	XVI	Transitória		

Fonte: Produção do autor.

Essa planilha de dependabilidade é utilizada no final de cada teste realizado entre os subsistemas do NanosatC-BR2. Se o levantamento dos requisitos e identificação das possíveis falhas foi definido de forma a atender as necessidades do sistema, as falhas detectadas durante os testes, por forma

natural ou induzido, estão relacionados nessa planilha. A localização de cada possível falha está relacionada com sua respectiva descrição de mitigação.

Com os registros das falhas identificadas nessa planilha pode ser observado que algumas classes de falhas não estão preenchidas. Uma análise mais criteriosa nos requisitos permite verificar se as classes não preenchidas realmente não podem ocorrer ou se até mesmo uma classe preenchida pode ocasionar outras situações de falha.

Observando a planilha pode ser analisado que a classe Limites do Sistema (subclasse III) não está preenchida. Verificando os requisitos pode ser observado que os subsistemas possuem área para armazenamento de informações. O tamanho das áreas de armazenamento deve ser definido conforme as especificações das necessidades. Se o tamanho dessas áreas não estiver definido adequadamente nem todos os dados são armazenados. Desta forma, nessa planilha na linha relacionada com classe de falha denominada Limites do Sistema e subclasse (III) Internas, é prevista e registrada uma possível falha. Sendo assim, nessa planilha as seguintes informações são incluídas conforme apresentado na Figura 6.16.

Figura 6.16 – Planilha de dependabilidade (outra possível falha detectada).

Classes de Falha	Subclasses		Combinação (1) Software	Mitigação
Limites do Sistema	III	Interna	Algumas informações executadas não são armazenadas na área reservada	Verificar capacidade de armazenamento de informação nas áreas especificadas para esta finalidade

Fonte: Produção do autor.

Este registro evidencia que informações poderão ser perdidas caso não haja espaço suficiente para armazenamento. Assim uma forma de mitigar essa falha seria verificar se há disponibilidade de área de armazenamento suficiente para as informações dos subsistemas comunicantes.

6.2.3.12. Árvore de dependabilidade

A árvore de dependabilidade permite identificar por dependabilidade e rastreabilidade possíveis falhas que ocasionam outro tipo de falha.

Para apresentação desta tese, dentre as possíveis falhas identificadas e relacionadas anteriormente, são selecionadas apenas uma falha de Software (1. Análise errônea dos requisitos) e uma falha de Interferência Física (7. Protocolo de comunicação não conseguir ser iniciado pelos subsistemas). Essas falhas são suficientes para apresentar benefícios da abordagem apresentada. Desta forma, apenas uma sequência de subclasses de falha de cada uma das duas falhas selecionadas é exemplificada, as quais foram registradas na planilha de dependabilidade.

Relacionando as possíveis falhas selecionadas com a árvore de dependabilidade temos as seguintes sequências de ocorrências que ocasionam essas falhas. O número romano apresentado entre parênteses () identifica a subclasse de falha. A letra entre parênteses identifica o tipo de falha:

- Possível Falha 1: Análise errônea dos requisitos:
    - Ocorrer uma falha de desenvolvimento (I), por um processo interno nos subsistemas que estão sendo testados (II), realizado por uma pessoa (VI), na codificação do software do subsistema que apresenta falha, por interpretação ou definição errônea dos requisitos (VIII), sendo realizado de forma não maliciosa (IX), não sendo intencional (XI), efetuado de forma acidental (XIII) e a falha é transitória (XV), ou seja, pode ocasionar uma falha no sistema se não resolvida. Esta sequência de subclasses de falhas pertence à combinação 1, sendo uma Falha de Software (A) e pertencente ao grupo Desenvolvimento.
- Possível Falha 7: Protocolo de comunicação não conseguir ser iniciado pelos subsistemas:

- Ocorrer uma falha operacional (II), por um processo externo aos subsistemas que estão sendo testados (IV), realizado por uma pessoa (VI), alteração de tensão elétrica em um ou ambos os subsistemas, ocasionado travamento no barramento de comunicação (VII), sendo realizado de forma não maliciosa (IX), não sendo intencional (XI), efetuado de forma acidental (XIII) e a falha é transitória (XVI), ou seja, pode ser resolvida sem causar danos permanentes. Esta sequência de classe de falhas pertence à combinação 16, sendo uma Falha de Interferência (F) e pertencente ao grupo Interação.

Na Figura 6.15 exibida anteriormente a planilha contém somente a primeira combinação de falha. A representação da Falha de número 7 na planilha é apresentada na Figura 6.17.

Figura 6.17 – Planilha de dependabilidade (falha número 7).

Classes de Falha	Subclasses		Combinação (16) Interferência Física	Mitigação
Persistência	XVI	Transitória	Subsistema não envia informações pelo canal de comunicação	Analisar tensão nos subsistemas

Fonte: Produção do autor.

Analisando a possível falha identificada de número 7 na árvore de dependabilidade, pode ser observado por rastreabilidade que essa falha também está relacionada com uma falha de Software. Ou seja, se houver uma interferência física no sistema em teste ocasionando uma parada na comunicação entre os subsistemas que estão sendo testados e a referida parada não estiver prevista na codificação do software, o sistema pode ficar travado. Com essa análise, devem ser previstos procedimentos para além de evitar a falha de interferência física, também ser efetuado procedimentos para evitar falhas na codificação dos componentes de software para não ocorrer falhas por motivos de interferência física. Exemplificando: Se houver alteração de tensão elétrica em algum subsistema, essa alteração pode parar o protocolo

de comunicação. Sendo assim, na definição dos componentes de software desses subsistemas deve ser previsto um número limitado de tentativas dessa comunicação e procedimentos a serem realizados evitando parar o sistema.

Por outro lado, outra análise que pode ser realizada é a possível falha identificada de número 1 também pode ocasionar uma falha de interrupção na comunicação entre os subsistemas testados devido à falha de software. Ou seja, deve ser analisada a codificação dos componentes de software dos subsistemas verificando as especificações das exceções, entradas inoportunas (caminhos furtivos), tolerâncias às falhas (AMBROSIO, 2005) entre outros casos de falhas que ocorrerem entre a comunicação dos subsistemas.

Analisando as situações descritas anteriormente e a rastreabilidade na árvore de dependabilidade, pode ser observado que na planilha de dependabilidade não estão registradas possíveis falhas associadas à classe Persistência e subclasse (XVI) Transitória. Analisando o protocolo de comunicação, se houver alteração na tensão entre os subsistemas, a comunicação entre os subsistemas comunicantes poderá ficar travada devido à falta de procedimento a ser realizado pelos componentes de software comunicantes. Desta forma, na planilha de dependabilidade na linha relacionada à classe de falha denominada Persistência e subclasse (XVI) Transitória, são registradas esta possível falha, referida por "Processamento interrompido" e a mitigação a ela associada que determina um tempo máximo para aguardar que a comunicação seja recuperada sem intervenções do mestre. Sendo assim, as informações são inseridas nessa planilha conforme Figura 6.18 sendo a falha identificada pelo número 10.

Figura 6.18 – Planilha de dependabilidade (outra possível falha identificada).

Classes de Falha	Subclasses		Combinação (1) Software	Mitigação
Persistência	XVI	Transitória	Processamento interrompido	Determinar tempo para tentativas de comunicação

Fonte: Produção do autor.

Outras situações de falhas poderão ser analisadas de forma analógica, inclusive associadas a outras classes de falhas, sendo com isto outro caminho de mitigação identificado na árvore de dependabilidade.

Quanto mais minuciosas são as análises dos requisitos e a identificação de possíveis falhas na taxonomia da dependabilidade mais casos de falhas são possíveis de serem identificados e testados.

6.2.3.13. Tabela FMEA

A Tabela 6.5 está preenchida com as informações referente às falhas de número 1, 7 e 10 identificadas durante os testes do OBC em interação com a SLP simulada, e discutidas anteriormente. Essa Tabela relaciona possíveis falhas a cada situação operacional, classifica as falhas identificadas na planilha de dependabilidade, as possíveis consequências da falha, elementos do sistema afetados pela falha e os mecanismos/equipamentos utilizados para detectar essas falhas.

Tabela 6.5 - FMEA com as possíveis falhas identificadas.

a) Procedimento identificando possível falha	b) Subclasse / Combinação / Identificação da falha	c) Efeitos da falha (consequência)	d(Local afetado pela falha	e) Mecanismo de controle para detecção da falha
1.Envio de uma sequência de comandos do OBC para a SLP	XIII / 1 / .1	Subsistemas não trocam informações	Subsistemas	Analisador Lógico I2C
7.Conexão física entre o OBC e Arduino (SLP) no barramento I2C	XVI / 16 / .1	Barramento travado	Subsistemas	Osciloscópio e regulador de tensão
10.Envio de um comando do OBC para a SLP	XVI / 1 / .1	Subsistemas não trocam informações	Subsistemas	Analisador Lógico I2C

Fonte: Produção do autor.

Apesar de o processo permitir que outras falhas sejam identificadas e relacionadas nessa Tabela, apenas essas três falhas foram identificadas e registradas, como exemplo da abordagem apresentada nesta tese.

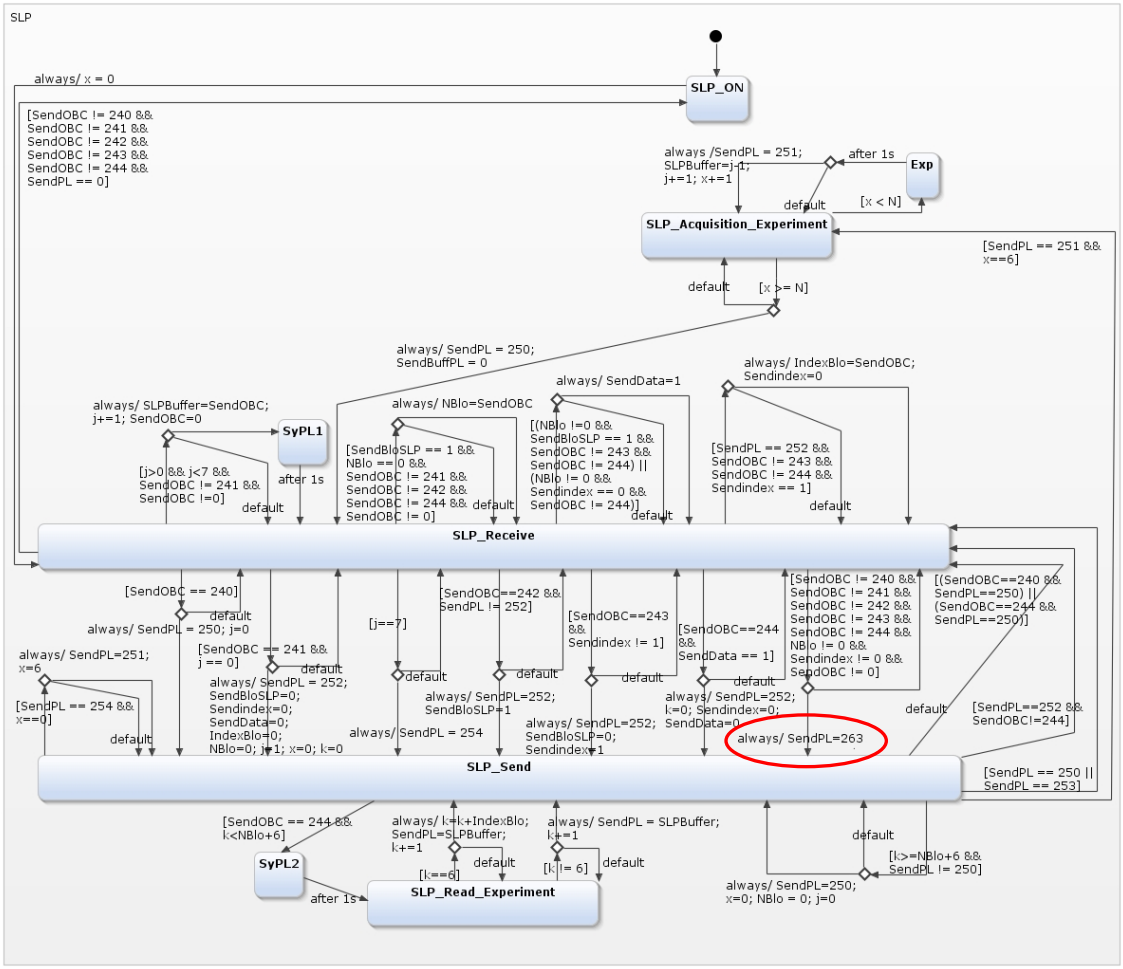
6.2.3.14. Modelos enriquecidos (PL com falha e OBC com robustez)

O modelo nominal da SLP (S2) para apresentação da abordagem é enriquecido com a seguinte falha de software identificada e relacionada anteriormente na planilha de dependabilidade:

- A SLP deve retornar o comando 0xfc (252) em resposta ao comando do OBC que solicita o envio de dados (0xf4 = 244). No caso da SLP não retornar o comando 0xfc, o OBC permanece aguardando por não haver uma definição nas especificações dos requisitos do que deve ser realizado neste caso. Sendo assim, o modelo da SLP é enriquecido com falha enviando um comando não especificado para ser recebido pelo OBC. O objetivo é testar a robustez do modelo do OBC.

As alterações na modelagem da SLP gera um modelo enriquecido com falhas denominado S2f. O modelo nominal da SLP é alterado para enviar o comando [(*SendPL*=263 (*CmdUnknown*)] não especificado nos requisitos para o OBC (circulado em vermelho). A Figura 6.19 apresenta o modelo SLP enriquecido com falha.

Figura 6.19 – Modelo da SLP enriquecido com falhas.



Fonte: Produção do autor.

Identificando as possíveis falhas que possam ocorrer, o modelo nominal do OBC (S1) é enriquecido com robustez buscando evitar possíveis falhas, gerando um modelo enriquecido com robustez (S1r).

A Tabela 6.6 apresenta em negrito a complementação dos estados, eventos, condições e ações para enriquecer com robustez o modelo nominal do OBC.

Tabela 6.6 - Estados, eventos e ações do OBC com robustez.

Estados **OBC**	Eventos	Ações	Descrição	Condições	Descrição
OBC_ IDLE	*CmdUnknown* (253) *Step* 5.2/12.2/15.2/ 18.2	**to+=1**	SLP não reconheceu comando do OBC. OBC ficará enviando comando perguntando se SlaveReady. **to= variável contadora de leitura com erro.**	SendPL not=250 e not=251 e not=252 e not= 254e **to<5**	OBC não reconheceu comando da SLP e não iniciou gravação de dados. **São realizadas cinco tentativas de leitura**
	to	**to=0 OBCError= 1**	**Comunicado alerta de erro e envio de novo comando para SLP**	**to>=5**	**Cinco leituras de comandos não reconhecidos**
OBC_ Send	*SlaveOK*(250) *Step* 4	SendOBC= *Set DateTime* (241) i=0 l=0 **OBCError= 0**	OBC informa que será enviado data e hora do momento da interação com a SLP. Inicializa variável para contar seis bytes do envio da Data e Hora (i). Inicializa variável indicadora que será enviado número de bloco de dados a ser lido (l). **OBCError = variável indicadora que houver erro na sequencia de troca de informações.**	SendPL= *SlaveOK*(2 50) e y=0	SLP está livre para troca de informação. Inicializa contador de tempo para espera de leitura de dados da SLP (y).
	CmdOK(252) *Step* 17	SendOBC= *StartTx*(244) SendBlo=0 y=0 **OBCError= 0**	OBC informa que será solicitada a leitura dos dados do experimento da SLP. Inicializa variável indicando que número de blocos e índice já foi informado (SendBlo=0). Inicializa variável contadora de tempo para espera da SLP coletar dados do experimento (y). **OBCError = variável indicadora que houver erro na sequencia de troca de informações.**	SendPL= 252 e l=4	SLP reconheceu comando (252) e os dados do experimento serão solicitados conforme índice e número de blocos de dados (l=4)
OBC_ Write	*StartTx*(244) *Step* 19	OBCData[i]= SendPL i+=1 **to=0**	Os dados lidos do buffer da SLP são transferidos e gravados na área de armazenamento do OBC (OBCData[i]. Incrementa variável contadora do índice de dados a serem lidos (i). **Inicializa variável contadora de erro (to).**	WriteOBC =1 e SendOBC = 244 e i<*BloTx*+6	Leitura e gravação dos dados do experimento da SLP (WriteOBC). Solicitação de envio de dados (244). O número de blocos solicitados será lido (*BloTX*) mais a data e hora da interação (seis bytes).
	to	**to+=1**	**Incrementa variável contadora de erro (to)**	**OBCData =0 e to<5**	**Dado lido é nulo**
	to	**OBCError= 1**	**Identifica na variável que houve erro**	**to>5**	**Cinco tentativas de leitura havendo erro**

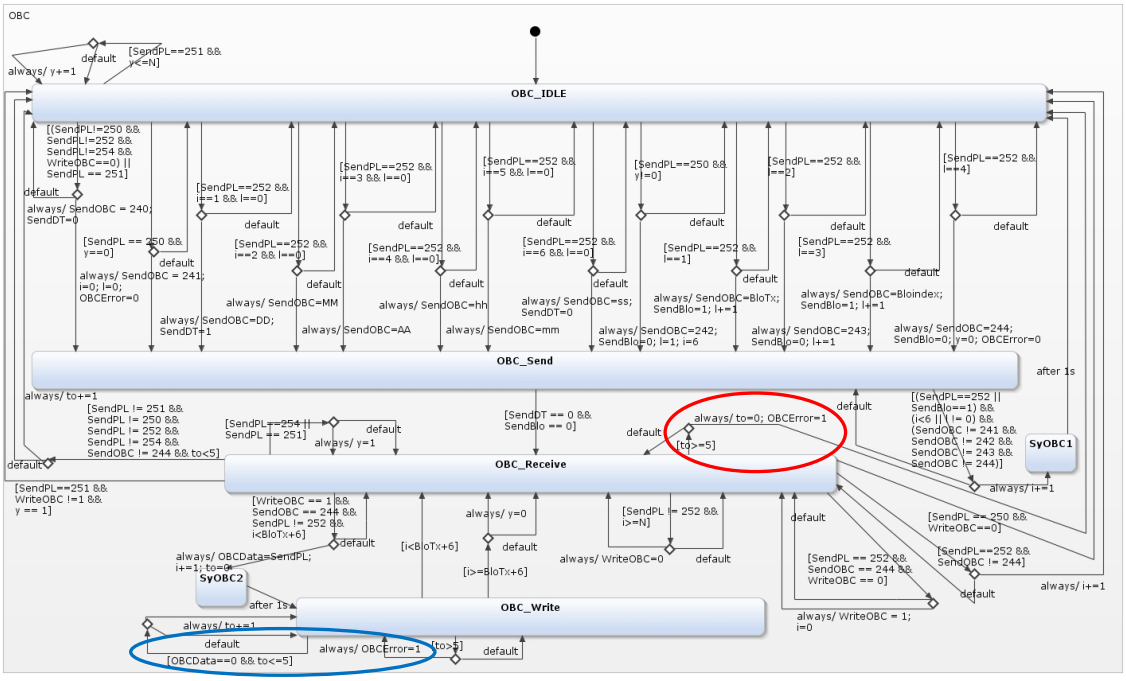
Fonte: Produção do autor.

A Figura 6.20 apresenta o resultado da transformação do modelo nominal do OBC com robustez. A robustez aplicada compreende o OBC verificar se a informação recebida da SLP é não reconhecida e, portanto, deve ser ignorada,

e realizar cinco tentativas de leitura. Após cinco tentativas um alerta de erro é acionado retornando ao estado *OBC_IDLE*, permitindo assim que o OBC saia do estado de interação com a SLP. Esta alteração é apresentada circulada em vermelho.

Outra robustez aplicada neste modelo é a verificação se o OBC grava informação na sua área de armazenamento. No caso de não haver a gravação é realizada a tentativa por cinco vezes (circulado em azul). Após estas tentativas um alerta de erro é acionado e o OBC retorna ao estado *OBC_IDLE*, como pode ser observado no modelo OBC enriquecido com robustez.

Figura 6.20 – Modelo do OBC enriquecido com robustez.



Fonte: Produção autor.

Os códigos dos modelos SLP enriquecido com falhas (S2f) e OBC enriquecido com robustez (S1r) são gerados e executados na Yakindu. Nos testes realizados entre estes modelos interagindo é observado que o modelo S1r identifica a falha e segue com suas atividades conforme as especificações. O objetivo nesta evolução foi verificar os requisitos de robustez do modelo OBC S1r.

6.2.3.15. Modelos enriquecidos (OBC com falha e PL com robustez)

Na próxima evolução, o modelo nominal do OBC (S1) é enriquecido com falha gerando um modelo enriquecido com falha S1f. O modelo S1 é alterado para enviar o comando (*SendOBC*=264) não especificado nos requisitos para o modelo da SLP.

Na sequência, o modelo nominal da SLP (S2) agora é enriquecido com robustez buscando evitar a falha que o modelo S1f ocasiona.

A Tabela 6.7 apresenta em negrito a complementação dos estados, eventos, condições e ações para enriquecer com robustez o modelo nominal da SLP.

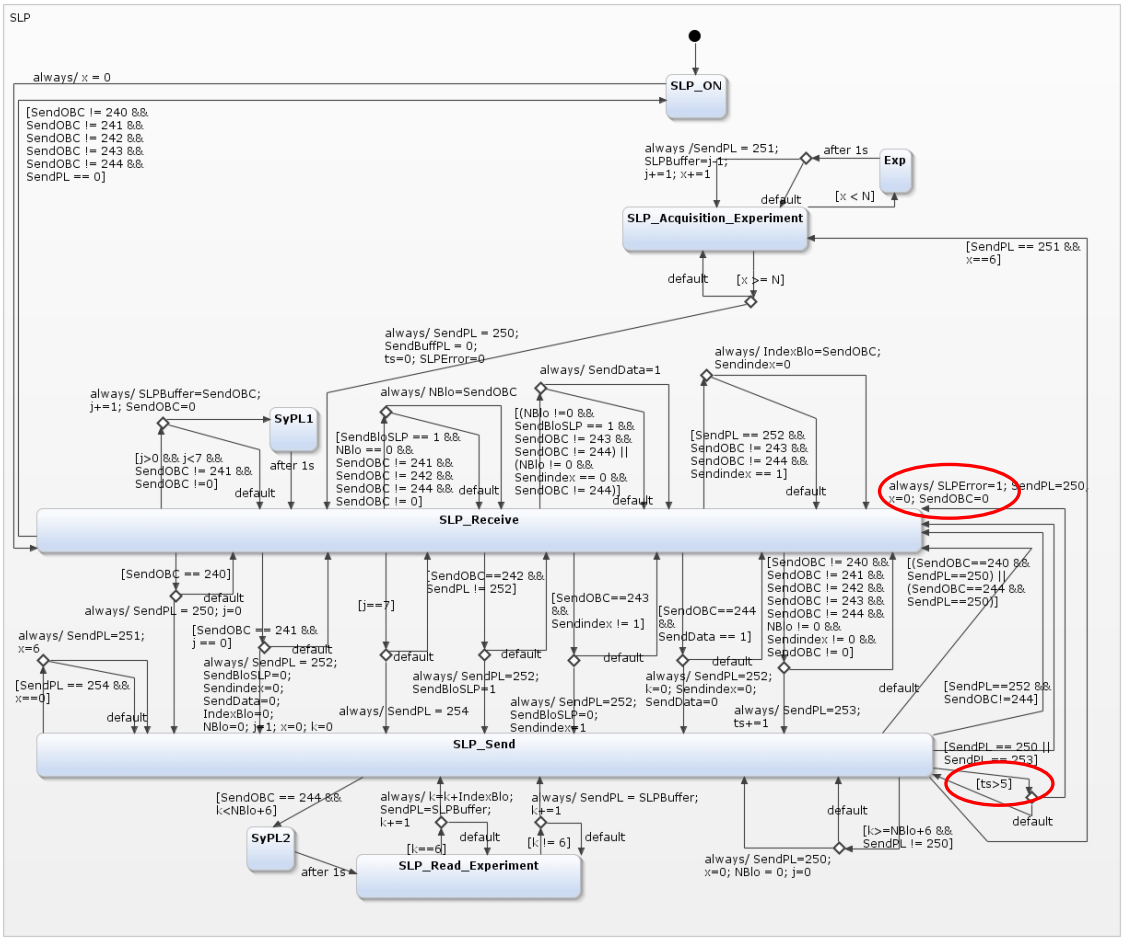
Tabela 6.7 - Estados, eventos e ações da SLP com robustez.

Estados SLP	Eventos	Ações	Descrição	Condições	Descrição
SLP_ Receive	SendPL=*SlaveOK*(250) *Step* 9.2	SendPL=*SlaveOK* (250) SendBuffPL=0 **ts=0 SLPError=0**	SendPL=*SlaveOK* (250) SendBuffPL=0. **ts =variável contadora de ocorrência de erro. SLPError = variável indicadora que houve erro.**	x>=N	Dados do experimento armazenado na capacidade do buffer da SLP
	ts	**SLPError=1, SendPL=*SlaveOK*(250) x=0 SendOBC=0**	**Identifica que houve erro (SLPError). SLP pronta para receber novamente comando. Inicializa variável apontadora do índice do buffer da SLP. Inicializa variável de leitura do OBC**	**ts=5**	**Cinco tentativas de leitura de comando não reconhecido**
SLP_ Send	*CmdUnknown Step* 5.2/13.2/16.2/ 19.2	SendPL=*CmdUnknown*(253) **ts+=1**	SLP não reconheceu comando recebido pelo OBC. **Incrementa variável de erro (ts)**	SendOBC not=240 e SendOBC not=241 e SendOBC not=242 e SendOBC not=243 e SendOBC not=244 e *NBlo* not=0 e Sendindex not=0 e **ts<5**	SLP não reconheceu comando. **Incrementa número de leitura com erro**

Fonte: Produção do autor.

A Figura 6.21 apresenta o resultado da transformação do modelo nominal da SLP para o modelo com robustez. A robustez aplicada verifica se a informação recebida do OBC não é reconhecida e realiza cinco tentativas de leitura. Após cinco tentativas um alerta de erro é acionado retornando ao estado *SLP_Receive*. Estas alterações são apresentadas em destaque circuladas em vermelho.

Figura 6.21 – Modelo da SLP enriquecido com robustez.



Fonte: Produção do autor.

Os códigos dos modelos OBC enriquecido com falhas (S1f) e SLP enriquecido com robustez (S2r) são gerados e executados na Yakindu. Nos testes realizados entre estes modelos interagindo é observado que o modelo S2r identifica a falha e segue com suas atividades conforme as especificações. O

objetivo agora nesta evolução foi verificar os requisitos de robustez do modelo SLP S2r.

6.2.3.16. Validação dos modelos com robustez

O objetivo é verificar e validar os requisitos de robustez dos modelos OBC e SLP enriquecidos com robustez (S1r e S2r respectivamente) em interação por simulação (MIL).

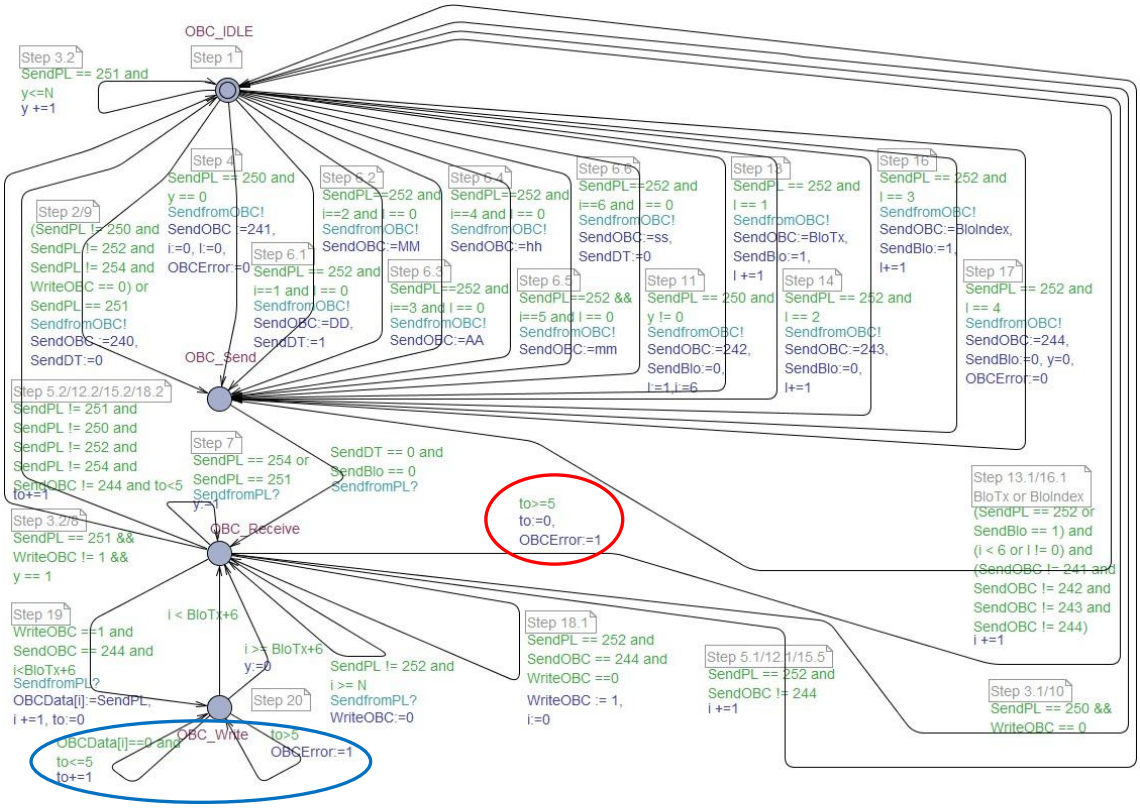
Estes modelos agora são testados conjuntamente em interoperabilidade na Yakindu. Os modelos em interoperabilidade foram assim validados, pois apresentaram a robustez de comportamento conforme as especificações.

6.2.3.17. Módulo 3

Os modelos validados com robustez (S1r e S2r) são implementados na Uppaal. O objetivo é efetuar estímulos de falhas de um modelo no outro gerando casos de teste de forma automatizada, utilizando conceito MBT. Os estímulos utilizados são as possíveis falhas identificadas e relacionadas na planilha de dependabilidade.

A Figura 6.22 apresenta o modelo enriquecido com robustez do OBC implementado na Uppaal, cujas alterações de robustez estão destacadas pelos círculos vermelho e azul.

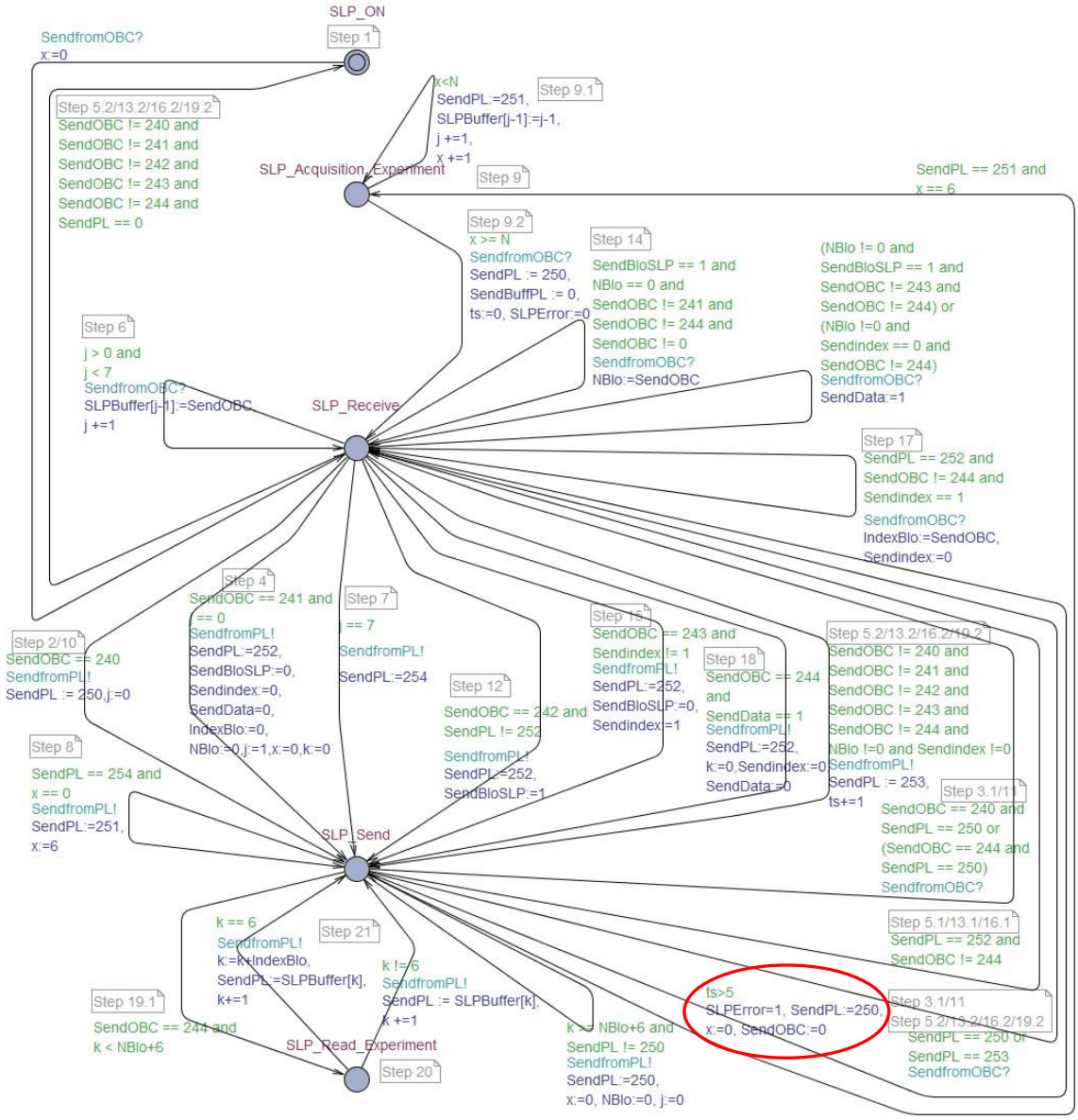
Figura 6.22 – Modelo do OBC enriquecido com robustez na Uppaal.



Fonte: Produção do autor.

A Figura 6.23 apresenta o modelo enriquecido com robustez da SLP implementado na Uppaal, cuja alteração de robustez é destacada pelo círculo vermelho.

Figura 6.23 – Modelo da SLP enriquecido com robustez na Uppaal.



Fonte: Produção do autor.

Um estímulo de falha é implementado no modelo da SLP. O estímulo é a SLP não enviar dados do experimento para o OBC. O objetivo é gerar um caso de teste verificando o modelo com robustez do OBC.

O caso de teste gerado a partir desse estímulo é apresentado na Tabela 6.8.

Tabela 6.8 - Caso de teste.

Caso de Teste	
1.OBC[OBC_IDLE->OBC_Send] 2.SLP[SLP_On->SLP_Receive] 3.SLP[SLP_Receive->SLP_Send] 4.OBC[OBC_Send->OBC_Receive] 5.OBC[OBC_Receive->OBC_IDLE] 6.OBC[OBC_IDLE->OBC_Send] 7. SLP[SLP_Send->SLP_Receive] 8.SLP[SLP_Receive->SLP_Send] 9.OBC[OBC_Send->OBC_Receive] 10. SLP[SLP_Send->SLP_Receive] 11.OBC[OBC_Receive->OBC_IDLE] 12.OBC[OBC_IDLE->OBC_Send] 13.SLP[SLP_Receive->SLP_Receive] 14.OBC[OBC_Send->OBC_IDLE] 15.OBC[OBC_IDLE->OBC_Send] 16.SLP[SLP_Receive->SLP_Receive]	17.OBC[OBC_Send->OBC_IDLE] 18.OBC[OBC_IDLE->OBC_Send] 19.SLP[SLP_Receive->SLP_Receive] 20.OBC[OBC_Send->OBC_IDLE] 21.OBC[OBC_IDLE->OBC_Send] 22.SLP[SLP_Receive->SLP_Receive] 23.OBC[OBC_Send->OBC_IDLE] 24.OBC[OBC_IDLE->OBC_Send] 25.SLP[SLP_Receive->SLP_Receive] 26.OBC[OBC_Send->OBC_IDLE] // OBC identifica falha e segue fluxo 27.OBC[OBC_IDLE->OBC_Send] 28.SLP[SLP_Receive->SLP_Receive] 29.SLP[SLP_Receive->SLP_Send] 30.OBC[OBC_Send->OBC_Receive] 31.SLP[SLP_Receive->SLP_Receive] 32.OBC[OBC_Receive->OBC_Receive]

Fonte: Produção do autor.

As setas (->) indicam a transição de um estado para outro.

Neste caso de teste o OBC contém a robustez implementada não permitindo que o sistema ficasse parado, no caso do estímulo de falha ter ocorrido. Outros estímulos, como o envio de informações não especificadas, foram provocados nos modelos OBC e SLP com robustez e ambos apresentaram o comportamento de robustez especificado. Desta forma os casos de teste foram considerados validados, completando assim o Cenário A de teste.

6.2.3.18. Módulo 4

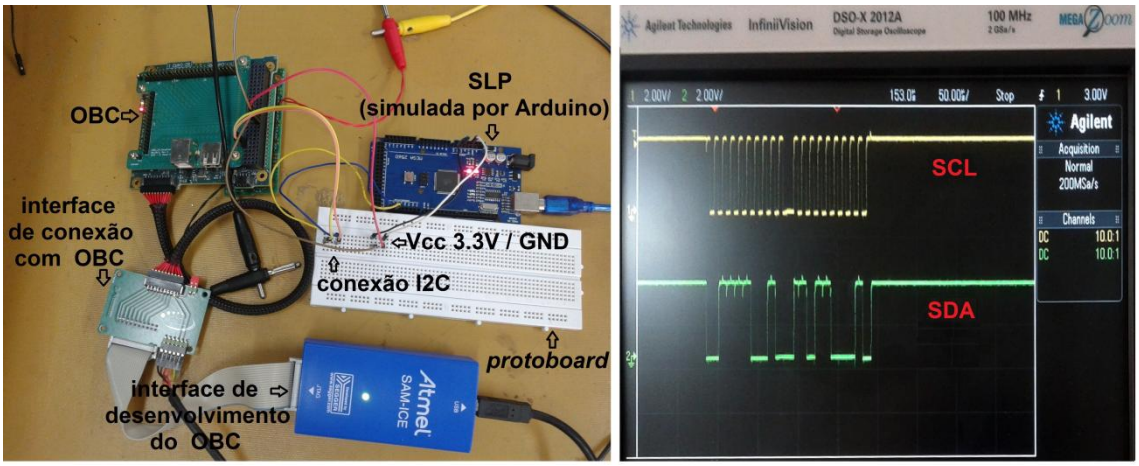
Para a realização deste módulo o subsistema OBC real foi conectado ao barramento de comunicação (HIL) com a placa contendo os códigos do subsistema simulado da PL enriquecido com robustez. O objetivo foi verificar se os requisitos do subsistema real OBC em interação com o modelo da PL validado em SIL atendiam as especificações e não apresentavam falhas de interoperabilidade no barramento (HIL).

6.2.3.19. Validação dos subsistemas reais separadamente

Neste módulo o subsistema OBC real é testado em interação com uma placa Arduino configurada com os códigos gerados pela modelagem com robustez que foi validada no Cenário A simulando o comportamento da carga útil SLP. O objetivo desses testes é validar a interoperabilidade dos subsistemas reais separadamente no barramento de comunicação (HIL).

A Figura 6.24 apresenta os subsistemas OBC real e SLP simulada conectados e sendo testados em laboratório (figura do lado esquerdo) e um osciloscópio capturando os sinais de dados (SDA) e de *clock* (SCL) no barramento de comunicação (figura do lado direito). O subsistema OBC enviou um comando para a SLP simulada. O analisador I2C *Logic Sniffer* também pode ser utilizado para analisar os sinais no barramento.

Figura 6.24 – OBC (real) e SLP (simulada) conectados via I2C.



Fonte: Produção do autor.

Para realizar este experimento as seguintes definições básicas de comunicação entre os subsistemas computador de bordo OBC e a carga útil SLP (simulada) foram consideradas:

- O software do OBC alvo de teste foi desenvolvido por empresa contratada (EMSISTI). Portanto o ambiente de teste utilizado durante o desenvolvimento foi diferente do ambiente STAE.
- A SLP utilizada é uma placa Arduino contendo um software, o qual simula o comportamento desta carga útil.
- O OBC (real) e a SLP (simulada) são conectados a um barramento conforme protocolo I2C.
- O OBC e SLP são conectados ao barramento com tensão elétrica de 3.3V cada.
- No barramento de comunicação é conectada uma fonte de alimentação controlando a corrente elétrica e a tensão elétrica máxima transmitidas nesse barramento.
- No barramento é conectado um osciloscópio detectando as informações do protocolo I2C, sendo SCL (sinal de relógio) e SDA (sinal de dados).
- O OBC é inicializado, executado e observado pela plataforma de desenvolvimento elaborada pela EMSISTI.
- A SLP é iniciada, executada e observada pela plataforma de desenvolvimento do Arduino.
- O OBC deve enviar comandos solicitando recebimento de informações das ações da SLP (*status* da SLP) e deve enviar comando solicitando recebimento dos dados coletados pela SLP.
- A SLP deve enviar informações referentes às ações ocorridas (*status*) e deve enviar dados coletados, conforme solicitados.

6.2.3.20. Veredicto

Após as conexões físicas e lógicas dos subsistemas, são realizados os testes conforme as especificações para o Cenário B. Alguns procedimentos de

verificação e detecção de falhas durante os testes realizados são relacionados a seguir:

- Procedimento 1: Conectar cabos no barramento conforme protocolo de comunicação.
    - Método 1: Observar no manual do nanosatélite os procedimentos para realizar as conexões.
    - Verificação 1: Conexão realizada conforme protocolo de comunicação.
- Procedimento 2: Verificar corrente elétrica e tensão elétrica de ambos os subsistemas conectados no barramento.
    - Método 2: ; Utilizar medidor de tensão para avaliar valores no barramento.
    - Verificação 2: Corrente elétrica e tensão elétrica transmitidas no barramento estão conforme as especificações de comunicação entre os subsistemas.
- Procedimento 3: Verificar os endereçamentos dos subsistemas conectados no barramento.
    - Método 3: Utilizar programa Arduino ou outro programa que permita ler endereçamentos conectados no barramento I2C.
    - Verificação 3: Endereços no barramento conforme especificações.
- Procedimento 4: Enviar um comando do OBC (mestre) para a PL (escravo) verificando se há comunicação no barramento e se a informação enviada pelo mestre foi recebida pelo escravo.
    - Método 4: Enviar comando pelo ambiente do desenvolvedor do software do OBC.

      Utilizar osciloscópio ou analisador lógico para verificar tráfego no barramento I2C.

- Verificação 4: Falha. Não houve tráfego de informação no barramento.
  - Solução 4: Alterar velocidade de transmissão de dados entre os subsistemas.

    Obs.: Esta falha está prevista na planilha de dependabilidade na Subclasse XIII (Acidental), Tipo de Falha (Software) e Grupo 1 (Falhas de Desenvolvimento), como apresentado mais adiante na Figura 6.24.

- Procedimento 5: Enviar uma sequência de comandos do OBC para PL conforme especificação dos requisitos.
  - Método 5: Enviar comandos pelo ambiente do desenvolvedor do software do OBC.

    Utilizar osciloscópio ou analisador lógico para verificar tráfego no barramento I2C.
  - Verificação 5: Falha. Alguns comandos não foram executados na sequência prevista.
    - Solução 5: Analisar o fluxo das informações no subsistema OBC conforme as especificações dos requisitos.

      Obs.: Esta falha está prevista na planilha de dependabilidade na Subclasse I (Desenvolvimento), Tipo de Falha (Software) e Grupo 1 (Desenvolvimento). Durante as evoluções dos testes de simulações no STAE foi verificado o fluxo das informações conforme as especificações dos requisitos. O comportamento do modelo simulado estava conforme especificado.

Estas falhas identificadas nas verificações 4 e 5 foram relatadas na tabela FMEA. Esta tabela foi preenchida com as informações referentes aos diagnósticos destas falhas sendo apresentadas na Tabela 6.9.

Tabela 6.9 - FMEA identificando falhas no teste.

a) Procedimento identificando possivel falha	b) Subclasse / Combinação / Identificação da falha	c) Efeitos da falha (consequência)	d) Local afetado pela falha	e) Mecanismo de controle para detecção da falha
4. Enviar um comando do OBC (mestre) para a PL (escravo)	I / 1 / .1	Osciloscópio recebe sinais SDA e SCL incoerentes	OBC e PL não trocaram informações	Osciloscópio e regulador de tensão
5.Enviar uma sequência de comandos do OBC para a SLP	I / 1 / .1	Analisador lógico captura sinais SDA e SCL conforme protocolo. Algumas informações não são transmitidas na sequência especificada	OBC e SLP não trocaram as informações especificadas	Analisador lógico I2C

Fonte: Produção do autor.

O procedimento de teste número 4 desta tabela FMEA foi analisado para verificar se a falha detectada e a mitigação que deveria ser realizada estavam previstas na planilha de dependabilidade e foram realizadas durante a evolução da sistemática de testes. Constatou-se que a falha detectada estava prevista na planilha de dependabilidade, identificada na subclasse XIII, e a mitigação para evitar essa falha também estava relacionada nessa planilha.

Apesar de a mitigação da falha estar relacionada na planilha, para efeito de validação dos testes que evidenciam a ocorrência da falha, propositalmente esta mitigação não foi implementada.

Assim os procedimentos 4 e 5 permitiram validar a identificação dessas falhas durante a evolução dos testes. A identificação da falha 4 na planilha é apresentada na Figura 6.25 a título de exemplo.

Figura 6.25 – Planilha de dependabilidade com possível falha 4 identificada.

Classes de Falha		Subclasses	Combinação (1) Software	Mitigação
Capacidade	XIII	Acidental	Falha de comunicação	Verificar velocidade de transmissão de informações

Fonte: Produção do autor.

Uma vez identificada a falha, verifica-se a adequação do procedimento relacionado na Tabela para evitá-la. Seguindo a sistemática de teste proposta, a mitigação relacionada é implementada, os testes são reaplicados para se constatar que a robustez à ocorrência da falha foi efetiva. Assim, se a possível falha identificada, catalogada e realizada foi devidamente tratada por meio da mitigação, o procedimento é considerado validado. Caso contrário, há necessidade de fazer nova análise de requisitos.

O procedimento de teste de número 2 referente à tensão elétrica está relacionado com o procedimento de número 5 referente a fluxo de informação, por dependabilidade. Exemplificando: no caso de haver problema de comunicação no barramento (falha de interferência física na interação) e os componentes de software dos subsistemas não estiverem configurados para essa situação (falha de software) poderá haver uma falha no sistema.

Após a realização de cada mitigação dos testes entre o OBC e a placa Arduino simulando a carga útil Sonda de Langmuir (SLP), os resultados apresentaram as seguintes observações referentes aos subsistemas OBC real e SLP simulado:

- O subsistema OBC real está atendendo inicialmente aos requisitos pré-estabelecidos de interoperabilidade com a SLP.
- O subsistema SLP simulado contém o comportamento funcional com robustez atendendo aos requisitos de interação com o OBC.

Demonstra-se assim a efetividade do uso da taxonomia de dependabilidade combinada às técnicas de transformação de modelos e geração automática de códigos e casos testes na abordagem sistemática de teste desenvolvida nessa tese.

6.2.3.21. Módulo 5

As cargas úteis do NanosactC-BR2 estão em desenvolvimento. Essas cargas úteis deverão ser testadas futuramente uma a uma com o OBC em interoperabilidade. O objetivo desse teste é testar e validar a interoperabilidade dos subsistemas reais no barramento de comunicação.

6.2.3.22. Módulo 6

Este último módulo tem como objetivo testar os requisitos de robustez dos subsistemas reais no barramento de comunicação real do Cubesat injetando estímulos de falhas por meio de um Mecanismo Emulador de Falhas (FEM).

6.2.3.23. Injeção de falhas

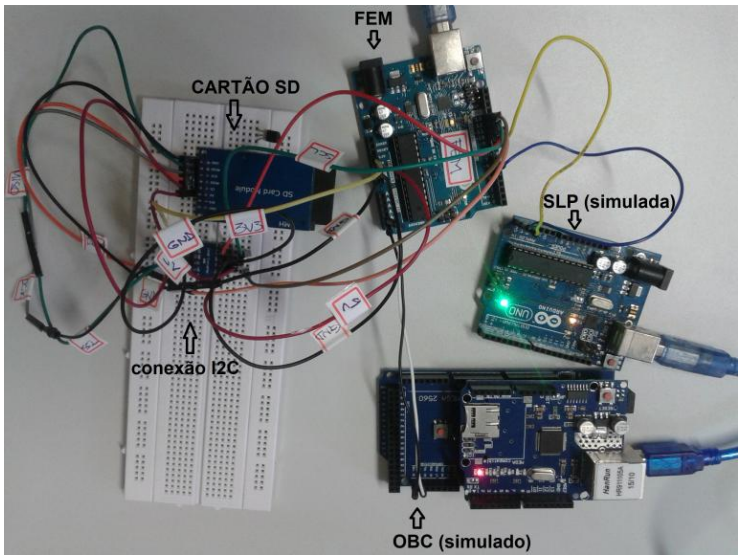
O injetor de falhas FEM, configurado em placa Arduino, neste experimento realiza testes nas seguintes configurações: i) FEM entre o OBC e a SLP simulados, e ii) FEM entre o OBC real e a SLP simulada.

O FEM é configurado manualmente com uma porta SDA e outra SCL para realizar comunicação por meio do protocolo I2C com um subsistema e configurado com outra porta SDA e outra SCL para realizar comunicação com o outro subsistema. Para um subsistema o FEM realiza comunicação como escravo e com o outro subsistema como mestre.

Em ambos os testes as mesmas falhas previamente determinadas durante a evolução dos testes são utilizadas para testar a robustez desses subsistemas submetidos ao teste de robustez.

Para a realização desse teste foi acrescentado na conexão com o FEM um cartão SD contendo os estímulos de teste. O cartão SD foi utilizado ao invés de acessar a base de dados MySQL para testar mais um meio de acessar uma base de dados contendo estímulos. A Figura 6.26 apresenta o OBC e SLP simulados sendo testados por meio de injeção de falhas utilizando o FEM.

Figura 6.26 – OBC e SLP (simulados) testados com o FEM.



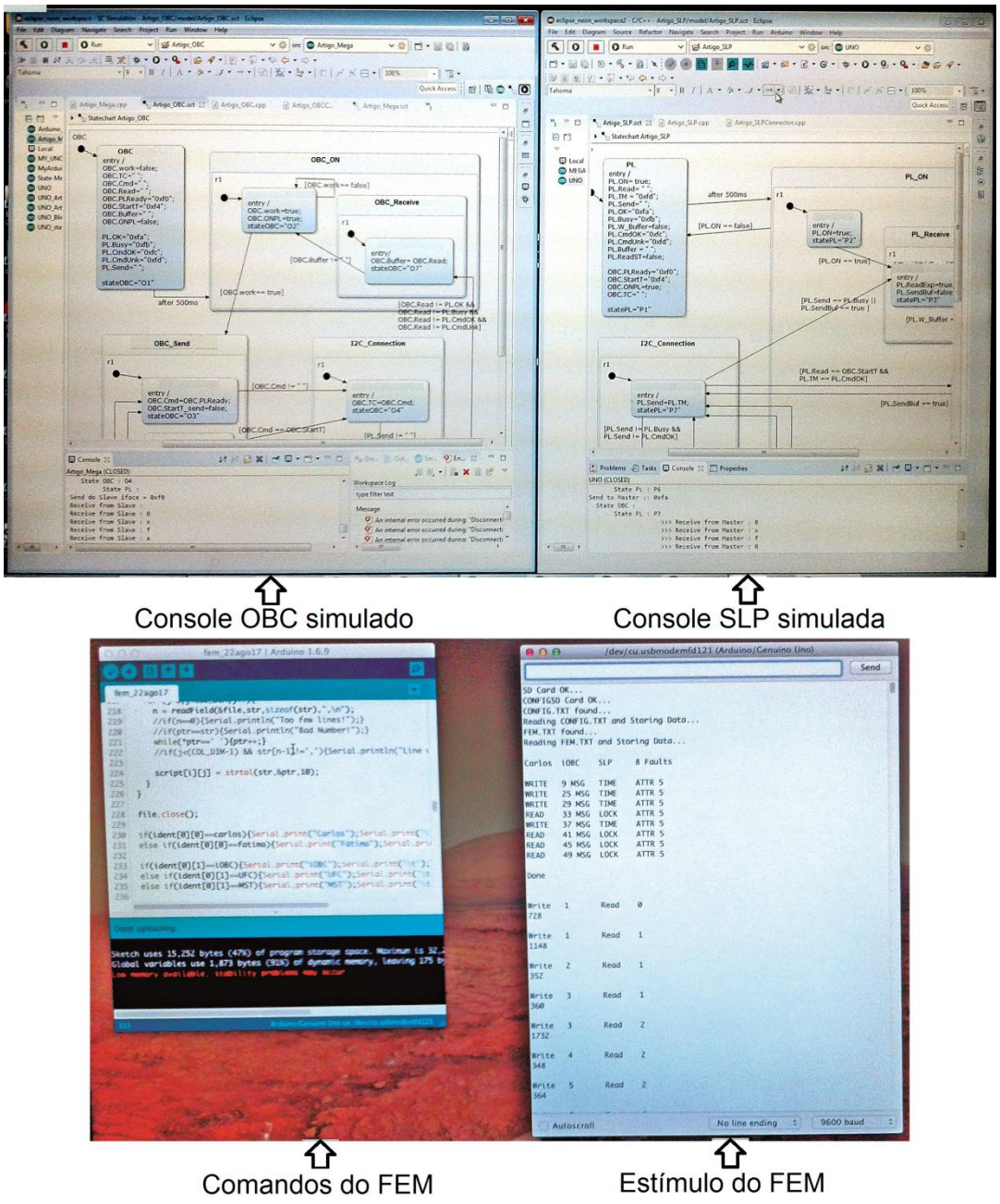
Fonte: Produção do autor.

A Tabela A.4 apresentada no Apêndice A contém uma das codificações utilizadas para configurar o FEM.

Um dos testes realizados no FEM foi injetar um estímulo de falha na informação enviada pelo OBC de forma que a informação recebida pela SLP não estivesse na especificação dos requisitos. O objetivo foi verificar a robustez de comportamento da SLP simulada.

A Figura 6.27 apresenta na parte superior dois ambientes de testes utilizando a ferramenta Yakindu. O ambiente do lado esquerdo é uma console do OBC simulado enviando comando para a SLP simulada. O ambiente do lado direito é a SLP simulada recebendo o comando do OBC simulado. A figura na parte inferior apresenta do lado esquerdo algumas linhas de comandos do FEM e do lado direito apresenta o FEM lendo o cartão SD com o estímulo de enviar uma informação não especificada do OBC para a SLP.

Figura 6.27 – Ambientes Yakindu (superior) e comandos do FEM (inferior).



Fonte: Produção do autor.

O teste demonstrou que os códigos gerados embarcados na SLP simulada foi capaz de identificar que o comando recebido é do tipo não reconhecido enviando o comando (*CmdUnknown*) resposta para o OBC. O OBC enviou novamente o comando. O FEM não injetou novamente a falha. A SLP reconheceu o novo comando e as atividades seguiram conforme as especificações.

Esta possível falha provocada estava relacionada na planilha e dependabilidade e na tabela FMEA e a robustez necessária já havia sido implementada no modelo da SLP simulada, apresentando que os procedimentos para evitar as falhas foram realizados conforme as evoluções da sistemática de testes.

Outro teste realizado foi o FEM alterar a velocidade de transmissão entre os subsistemas conectados ao barramento.

Outros tipos de falhas também realizados foram: i) falha de valor, implementada por meio de uma falha *bitflip*, ou seja, invertendo um *bit* da mensagem, ii) falha de fornecimento, implementada alterando a informação da mensagem, iii) falha de tempo, inserção de atrasos nas mensagens trocadas, e iv) falha específica do protocolo, para esta implementação é bloqueado a linha de barramento I2C de dados (BATISTA et al., 2018).

Os estímulos de falhas provocadas ocorreram conforme previstos na planilha de dependabilidade e na tabela FMEA, demonstrando que a abordagem apresentada para tratamento de falhas foi realizado conforme os procedimentos estabelecidos.

7 CONCLUSÕES

Uma abordagem para sistematização de testes de subsistemas comunicantes intensivos em software embarcado foi desenvolvida nesta tese de doutorado com o propósito de antecipar a identificação e tratamento de falhas de interoperabilidade no ciclo de desenvolvimento de missões espaciais que utilizam o padrão CubeSat. A sistematização conta um Sistema de Teste de Arquitetura Escalável (STAE) e compreende: i) a concepção de modelos comportamentais de subsistemas comunicantes – cujo propósito é verificar os requisitos de interoperabilidade dos subsistemas por meio da validação dos modelos (MIL - *Model-in-the-loop*), ii) a geração automática de código computacional a partir dos modelos validados, por ferramentas MDE – cujo propósito é a verificação dos requisitos de interoperabilidade em ambiente simulado, com o uso do barramento de comunicação real (SIL - *Software-in-the-loop*), iii) a evolução dos modelos para que casos de testes abstratos possam ser gerados por ferramentas MBT, iv) a execução de casos de testes para validar os subsistemas reais (HIL - *Hardware-in-the-loop*) na fase de integração, e v) a possibilidade de injetar falhas por meio de um Mecanismo Emulador de Falhas (FEM), o qual permite testar a interação dos subsistemas reais em termos de requisitos de robustez especificados. Para auxiliar na especificação de requisitos de robustez, a abordagem apresentada utiliza uma planilha de dependabilidade, que aplica os conceitos da árvore de dependabilidade e análise causa-efeito para mitigações de falha. A efetividade da abordagem desenvolvida no âmbito desta tese foi demonstrada por meio de seu uso nos testes de interoperabilidade dos subsistemas de software embarcados no nanosatélite NanosatC-BR2, em desenvolvimento no INPE, como parte do processo de V&V da missão.

A temática de tratamento de falhas em satélites de pequeno porte (nanosatélites) é explorada na literatura por abordagens de modelagem de requisitos, conceitos de segurança e testes de missão espacial. Porém, essas abordagens em geral não exploram as técnicas MBT e MDE de forma integrada, aos conceitos de dependabilidade, interoperabilidade e robustez.

Na sistemática de testes apresentada, métodos e técnicas MBT e MDE se complementam permitindo que testes de interoperabilidade, específicos para modelos dos subsistemas comunicantes, possam ser reutilizados ao longo do processo de V&V, graças ao STAE.

Assim, destaca-se como uma das principais vantagens da abordagem apresentada a reutilização dos conjuntos de testes, gerados a partir dos modelos comportamentais dos subsistemas de software comunicantes, em diferentes cenários de integração, o que reduz o esforço humano de especificações dos casos de testes de interoperabilidade dos software embarcados comunicantes em cada etapa do processo de V&V.

Outra contribuição significativa da tese é a planilha de dependabilidade e tabela FMEA desenvolvidas, por proporcionarem ao testador uma forma estruturada de analisar o software embarcado sob a ótica de levantamento e tratamento de possíveis falhas. A planilha de dependabilidade consiste de uma lista de falhas previstas na árvore de dependabilidade que podem ser usadas como guia para verificar e validar por meio da rastreabilidade a possível fonte da falha e propagação das falhas que possam ocorrer.

Essa abordagem permite simulações do comportamento dos subsistemas antes desses subsistemas serem testados em fase de integração.

O STAE permite a sua reutilização em diferentes fases de uma mesma missão fazendo implementações nas fases de teste de validação, adicionando ou removendo os subsistemas do satélite no *loop*. Também existe a possibilidade do mesmo sistema de teste ser usado em diferentes satélites da mesma família. Os recursos propostos para implementar o STAE são *COTS*, eficazes e flexíveis o suficiente para testar diversas cargas úteis em missão nanosatélite que adotam o padrão Cubesat.

Entretanto, a abordagem desenvolvida requer alguns esforços extras do engenheiro de sistemas para a implementação do STAE e sua instanciação para cada cenário de integração. Ainda, o uso das ferramentas de modelagem e técnicas MDE para transformação de modelos requerem aprendizagem por parte do testador. Os códigos computacionais gerados também nem sempre

são exatamente como esperado. Há necessidade de uma análise desses códigos verificando se algumas alterações devem ser realizadas para atender aos requisitos. As ferramentas Uppaal e Yakindu (no ambiente Eclipse) também necessitam certa atenção em suas instalações. Necessitam ser instaladas com uma determinada sequência de execuções.

Outra consideração importante é sobre a capacidade das placas computacionais programáveis Arduino atenderam aos testes realizados nesta proposta de tese (Cenários A, B, C, D). Caso haja necessidade de mais recursos de processamento ou armazenamento de informações em memória do que os previstos para o STAE implementado nesta tese pode-se utilizar a placa Arduino Galileo ou a placa *Raspberry*, por exemplo.

Especificamente, como resultados da experimentação da abordagem proposta nos módulos 1, 2, 3, 4 e 6 dirigido ao nanosatélite NanosatC-BR2 pode-se destacar que:

- A abordagem proposta foi efetiva, pois na elaboração do módulo 4 possibilitou verificar que uma das possíveis falhas previstas na planilha de dependabilidade foi identificada na integração do OBC com a SLP (simulada).
- O propósito da abordagem de antecipar falhas no ciclo de desenvolvimento foi atingido, pois a simulação do *Model-in-the-loop* (MIL) permite verificar se há necessidade de alterações na modelagem para cumprimento dos requisitos antes mesmo do software ser desenvolvido, antecipando problemas que só seriam evidenciados quando testado o *Hardware-in-the-loop* (HIL).
- A modelagem também permite uma documentação visual do comportamento do sistema. A sistematização dos testes apresentada permite ser realizada diversas vezes, tantas quantas forem necessárias de forma dinâmica.

Para a continuação da utilização da abordagem apresentada, está prevista a atividade para a experimentação do módulo restantes 5 de forma a verificar e

validar todos os procedimentos para tratamento de falhas, inicialmente, experimentado em satélites de pequeno porte. A planilha de dependabilidade e tabela FMEA devem ser preenchidas buscando identificar possíveis falhas que possam ocorrer na interação entre os subsistemas comunicantes. Após os testes, as falhas identificadas devem estar relacionadas nessa planilha e tabela.

E ainda, como proposta para trabalhos futuros, pode-se constatar que os cenários a serem experimentados podem ser escaláveis, pois, permitem alterações e inclusão de mais cenários conforme as necessidades. Um exemplo de trabalho futuro seria incluir mais um cenário para testar comandos enviados de uma estação de solo para o ambiente de teste. Poderia ser incluído conexão *Bluetooth* no ambiente de teste para realizar os procedimentos da abordagem apresentada.

Uma contribuição acadêmica desejável para aumentar a automatização da abordagem sistemática de testes desenvolvida nesta tese seria a concepção de uma ferramenta capaz de transformar os modelos MIL em SIL e vice versa. Além e agilizar o processo, esta ferramenta minimizaria o risco de erros humanos no mapeamento dos modelos MIL para os modelos SIL, a partir dos quais a geração automática de códigos é realizada.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **NBR ISO 14620-1**: sistemas espaciais – requisitos de segurança Parte 1: segurança de sistema. Rio de Janeiro, 2009.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **NBR ISO 14620-2**: Sistemas espaciais – requisitos de segurança Parte 2: operação de centro de lançamento. Rio de Janeiro, 2009.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **NBR ISO 14620-3**: Sistemas espaciais – requisitos de segurança Parte 3: sistemas de segurança de vôo. Rio de Janeiro, 2009.

ABRAHÃO, S.; BORDELEAU, F.; CHENG, B., H., C.; KOKALY, S. User experience for model-driven engineering: challenges and future directions. In: INTERNATIONAL CONFERENCE ON MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS (MODELS), 20., 2017, Austin, TX, EUA, **Proceedings…** IEEE, 2017. p. 1-8. Disponível em: https://ieeexplore.ieee.org/document/8101269. Acesso em: 20 ago. 2018.

ADDAIM, A.; KHERRAS, A.; E. B. Design of low-cost telecommunications Cubesat-class spacecraft. In: ARIF, T. (Ed.). **Aerospace technologies advancements**. 2010. p. 1-5. Disponível em: https://www.researchgate.net/publication/221906779_Design_of_Low-cost_Telecommunications_CubeSat-class_Spacecraft. Acesso em: 5 jan. 2017.

ALI, S.; HEMMATI H.; HOLT, N. E.; ARISHOLM, E.; BRIAND, L. C. **Model transformations as a strategy to automate model-based testing:** a tool and industrial case studies. 2010. p. 1-10. Disponível em: https://www.semanticscholar.org/paper/Model-Transformations-as-a-Strategy-to-Automate-A-Ali-Hemmati/ac88b13d5fec1c9a4ab52d2f1857d1d7c8c28a08. Acesso em: 15 jul. 2017.

ALMEIDA, D. P. **Modelagem da Interoperabilidade entre computador de bordo e cargas úteis do NanosatC-Br2 com apoio da ferramenta Uppaal e análise & projeto da Daughterboard do computador de bordo.** 2016. Trabalho (Conclusão de Curso em Engenharia Mecânica) - Universidade de São Paulo, São Carlos, 2016.

AMBROSIO, A. M. **CoFI – uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em aplicações espaciais**. Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2005. Disponível em: http://mtc-m16.sid.inpe.br/col/sid.inpe.br/MTC-m13@80/2005/09.06.13.34/doc/publicacao.pdf . Acesso em: 03 mar. 2017.

AVIZIENIS, A.; LAPRIE, J. C.; RANDELL, B.; LANDWEHR, C. **Basic concepts and taxonomy of dependable and secure computing.** 2004. 33p. Disponível em: https://www.nasa.gov/pdf/636745main_day_3-algirdas_avizienis.pdf. Acesso em: 15 maio 2016.

ASUNDI, S. A.; FITZ-COY, N. G. **Cubesat mission design based on a systems engineering approach**. 2013. 5p. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6496900. Acesso em: 26 nov. 2015.

BATISTA, C. L. G.; MARTINS, E.; MATTIELLO-FRANSCISCO, M. F. On the use of a failure emulator mechanism at nanosatellite subsystems integration tests. In: LATIN-AMERICAN TEST SYMPOSIUM (LATS), 19., 2018, São Paulo. **Proceedings…** IEEE, 2018, p. 1-4. Disponível em: https://ieeexplore.ieee.org/abstract/document/8347242. Acesso em: 7 mar. 2019.

BINDER, R. V.; LEGEARD, B.; KRAMER, A. Model-based testing: where does it stand?. **Communications of the ACM**, v.58, p.1-3, 2015. Disponível em: https://www.researchgate.net/publication/272414727_Model-Based_Testing_Where_Does_It_Stand. Acesso em: 6 maio 2018.

CHARTRES, J.; SANCHEZ, H.; HANSON, J. **EDSN development lessons learned**. 2014. 11p. Disponível em: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140017733.pdf. Acesso em: 5 out. 2015.

DIAS NETO, A. C.; SUBRAMANYAN, R.; VIEIRA, M.; TRAVASSOS, G. H. A Survey on model-based testing approaches: a systematic review. In : ACM INTERNATIONAL WORKSHOP ON EMPIRICAL ASSESSMENT OF SOFTWARE ENGINEERING LANGUAGES AND TECHNOLOGIES, 1., EUA, 2007. **Proceedings…** ACM, 2007. p. 1-2. Disponível em: https://dl.acm.org/citation.cfm?id=1353681. Acesso em: 21 out. 2017.

ECLIPSE FOUNDATION. **Ambiente de desenvolvimento MDE**. 2016. Disponível em: https://eclipse.org/home/index.php. Acesso em: 20 jun. 2016.

EUROPEAN COMMITTEE FOR ELECTROTECHNICAL STANDARDIZATION. **EN 50128:** railway applications – communications, signalling and processing systems – software for railway control and protection systems. Brussels, 2001, p. 13-25. Disponível em: https://standards.globalspec.com/std/1678027/EN%2050128. Acesso em: 7 maio 2017.

EMSISTI SISTEMAS ESPACIAIS E TECNOLOGIA. **Missão NanosatC-BR2:** documento de requisitos de interfaces do computador de bordo do satélite NanosatC-BR2. São José dos Campos: EMSISTI, 2016. p. 7-32.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-S-ST-00-01C**: ECSS system: glossary of terms. 2012. 42p. Disponível em: https://ecss.nl/standard/ecss-s-st-00-01c-glossary-of-terms-1-october-2012/. Acesso em: 3 dez. 2015.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-Q-HB-80-03**: space product assurance, software dependability and safety.2012. p. 8-28. Disponível em: https://ecss.nl/hbstms/ecss-q-hb-80-03a-software-dependability-and-safety/. Acesso em: 23 nov. 2015.

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION. **ECSS-M-ST-10C Rev. 1**: Space project management, project planning and implementation. 2009. p. 19-33. Disponível em: https://www.skatelescope.org/public/2011-11-18_WBS-SOW_Development_Reference_Documents/ECSS-M-ST-10C_Rev.1(6March2009).pdf. Acesso em: 15 dez. 2015.

FERNÁNDEZ-ISABEL, A.; FUENTES-FERNÁNDEZ, R. **A model-driven engineering process for agent-based traffic simulations**. 2015. p. 2-4. Disponível em: http://www.albertofernandezisabel.com/wp-content/uploads/A-MDE-Process-for-Agent-Based-Traffic-Simulations.pdf. Acesso em: 13 jul. 2018.

GRADES, E. A. **Test models and coverage criteria for automatic model-based test generation with UML state machines**. Dissertação (Mestrado em Informática) - Humboldt Universität zu Berlin, Berlin, 2009. Disponível em: https://pdfs.semanticscholar.org/478b/1518f7f35e0658fd20a41cb567deb44901ed.pdf. Acesso em: 7 mar. 2017.

GUNDY-BURLET, K. Validation and verification of LADEE models and software. **Aerospace Sciences Meeting**, v.51, p.1-5, 2013. Disponível em: https://arc.aiaa.org/doi/10.2514/6.2013-592. Acesso em: 26 jun. 2017.

HELVAJIAN, H.; JANSON, S. W. **Small satellites**: past, present and future. Virginia: American Institute of Aeronautics and Astronautics, 2008. p. 14-17. Disponível em: https://www.worldcat.org/title/small-satellites-past-present-and-future/oclc/772698699. Acesso em: 20 set. 2015.

HERPEL, H.J.; KEREP, M., LI, J.; XIE, J. JOHANSEN, B.; KVINNESLAND, K.; KRUEGER, S., BARRIOS, P. Model based testing of satellite on-board software -an industrial use case. In: IEEE AEROSPACE CONFERENCE EUA, 2016. **Proceedings...** IEEE, 2016. p. 1-5. Disponível em: https://ieeexplore.ieee.org/abstract/document/7500845. Acesso em: 29 out. 2017.

HSUEH, M. C.; TSAI, T. K.; IYER, R. K. **Fault injection techniques and tools**. Los Alamitos IEEE Computer Society Press, 1997. p. 1-3. Disponível em: https://ieeexplore.ieee.org/document/585157. Acesso em: 17 mar. 2018.

IACONO, D.; BRANCATI, F.; ROSSI, F.; BONDAVALLI, A. Thinking outside the vehicle: the impact of connected vehicles on safety and security. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS, 45., 2009. **Proceedings…** 2009. p. 1-3. Disponível em: http://www.lbd.dcc.ufmg.br/colecoes/dsn/2015/019.pdf. Acesso em: 25 out. 2018.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ICS 43.040.10**. road vehicles - functional safety - Part 8: supporting processes. 2009. Disponível em: https://www.iso.org/standard/51364.html. Acesso em: 09 abr. 2018.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **Missão do NanosatC-BR1**. São José dos Campos: INPE, 2019. Disponível em: http://www.inpe.br/crs/nanosat/missao/nanosatc_br1.php. Acesso em: 15 maio 2019.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **NANOSATC-BR system user manual**. São José dos Campos: INPE, 2011. p. 8-12.

INNOVATIVE SOLUTION IN SPACE (ISIS). **NCBR1-payload user manual**. Delft: ISIS, 2013. p. 8-25.

JACKLIN, S. A. **Survey of verification and validation techniques for small satellite software development**. Washington: NASA, 2015. p. 1-5. Disponível em: https://pdfs.semanticscholar.org/80b2/1dd1800ff794a6588fcd25cc6bf09683a4a1.pdf. Acesso em: 16 maio 2016.

KASLOW, D.; AYRES, B.; CHONOLES, M.; J., GASSTER, S. Developing a Cubesat Model-Based System Engineering (MBSE) reference model – interim status. In: IEEE AEROSPACE CONFERENCE, 2015. **Proceedings...** IEEE, 2015. p. 1-6. Disponível em: https://ieeexplore.ieee.org/abstract/document/7118965. Acesso em: 23 ago. 2018.

LANGER, M.; BOUWMEESTER, J. Reliability of Cubesats – statistical data, developers' beliefs and the way forward. In: ANNUAL AIAA/USU CONFERENCE ON SMALL SATELLITES, 30., 2016. **Proceedings…** AIAA, 2016. p. 1-9. Disponível em: https://repository.tudelft.nl/islandora/object/uuid%3A4c6668ff-c994-467f-a6de-6518f209962e. Acesso em: 29 ago. 2018.

LOPES, M., **Engenharia de requisitos.** São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2015. Notas de aula.

LXTREME. **Analisador lógico**: about projects. Disponível em: https://www.lxtreme.nl/. Acesso em: 3 nov. 2017.

MATTIELLO-FRANCISCO, M. F. **Sistemas computacionais em aplicações espaciais**. São José dos Campos: INPE, 2003. p. 6-8. Disponível em: http://mtc-m16.sid.inpe.br/col/sid.inpe.br/jeferson/2003/10.13.15.25/doc/publicacao.pdf. Acesso em: 17 out. 2015.

MATTIELLO-FRANCISCO, M. F. **InRob:** uma abordagem para testes de interoperabilidade e de robustez de subsistemas de tempo-real intensivos em software. 2009. Tese (Doutorado em Engenharia Eletrônica e Computação) - Instituto Tecnológico da Aeronáutica (ITA), São José dos Campos, 2009.

MATTIELLO-FRANCISCO, M. F.; MARTINS, E.; CAVALLI, A. R.; YANOD, E. T. An approach for testing interoperability and robustness of real-time embedded software. **Journal of Systems and Software**, v.85, n.1, p.3-15, 2012. Disponível em: https://www.researchgate.net/publication/220377079_InRob_An_approach_for_testing_interoperability_and_robustness_of_real-time_embedded_software. Acesso em: 12 maio 2016.

MARTINS, E.; SABIÃO, S. B.; AMBROSIO, A. M. **"**Condata: a tool for automating specification-based test case generation for communication systems". **Software Quality Journal**, v. 8, n. 4, p. 303-319, 1999. Disponível em: https://link.springer.com/article/10.1023/A:1008930105477. Acesso em: 8 abr. 2017.

MCROBERTS, M.. **Arduino básico.** [S.l.]: Novatec, 2011. p. 22-38. Disponível em: https://edisciplinas.usp.br/pluginfile.php/4287597/mod_resource/content/2/Ardu%C3%ADno%20B%C3%A1sico%20-%20Michael%20McRoberts.pdf. Acesso em: 25 jul. 2015.

MONTECCHI, L.; LOLLINI, P.; BONDAVALLI, A. Dependability concerns in model-driven engineering. In: IEEE INTERNATIONAL SYMPOSIUM ON OBJECT/COMPONENT/SERVICE-ORIENTED REAL-TIME DISTRIBUTED COMPUTING (ISORC), 2011. **Proceedings...** IEEE, 2011. Disponível em: http://www.ic.unicamp.br/~leonardo/publications/dependability-concerns-in-model-driven.html. Acesso em: 25 set. 2017.

MONTECCHI, L. **Model-driven engineering and its application for dependability analysis**: introduction to MDE. São José dos Campos: INPE, jun. 2016. Notas de aula.

MYSQL. **MySQL 5.1 reference manual**. 2013. Disponível em: http://dev.mysql.com/doc/refman/5.1/en/index.html. Acesso em: 5 maio 2013.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION **NASA hears MarCO CubeSats loud and clear from Mars**. Disponível em: https://www.jpl.nasa.gov/news/news.php?feature=7295&utm_source=iContact&utm_medium=email&utm_campaign=nasajpl&utm_content=daily20181127-3. Acesso em: 26 jan. 2019.

NOVAL, J. J. S.; FLORES, R. D.; KASTENSMIDT, F. L. **Payload SMDH-UFRGS-INPE, NanosatC-BR2:** cargas úteis. São José dos Campos: INPE, 2016. p. 5-10.

PELESKA, J. Industrial-strength model-based testing - state of the art and current challenges. **Computer Science - Software Engineering**, p.1-8, 2013. Disponível em: https://arxiv.org/pdf/1303.1006.pdf. Acesso em: 6 mar. 2017.

PERROTIN, M.; YABAR, C. **Modelling and automatic code generation.** 2014. Disponível em: https://indico.esa.int/event/53/contributions/2671/attachments/2146/2494/11_-_Automatic_Code_Generation.pdf. Acesso em: 16 jan. 2018.

PHILIPS SEMICONDUCTORS. **AN10216-01:** I2C manual. 2003. Disponível em: https://www.nxp.com/docs/en/application-note/AN10216.pdf. Acesso em: 23 jun. 2015.

PHPMYADMIN. **Documentação do PhPMyAdmin:** versão 3.2.0.1. 2009. Disponível em: http://localhost/phpmyadmin/Documentation.html. Acesso em: 5 maio 2012.

PIMENTEL, A. R. **Projeto de software usando a UML:** apostila para curso de projeto de sistemas orientado a objetos ssando a UML. Curitiba: UFPR, 2015. p. 4-11. Disponível em: http://www.inf.ufpr.br/andrey/ci167/apostilaUml.pdf. Acesso em: 28 maio 2017.

PINHEIRO, A. C.; SIMÃO, A.; AMBROSIO, A. M. FSM-based test case generation methods applied to test the communication software on board the ITASAT university satellite: a case study. **Journal of Aerospace Technology and Management**, v.6, n.4, p. 1-6, 2014. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S2175-91462014000400447. Acesso em: 29 out. 2017.

PUENTE, J. A. DE LA; GARRIDO, J.; ZAMORANO, J.; ALONSO, A. Model-driven design of real-time software for an experimental satellite. **IFAC Proceedings**, v.47, n. 3, p. 1592-1598, 2014. Disponível em: https://www.sciencedirect.com/science/article/pii/S1474667016418409. Acesso em: 17 mar. 2018.

RABASA, G. O. **Methods for dependability analysis of small satellite missions**. [S.l.]: Archivio Istituzionale della Ricerca, 2015. p. 7-39. Disponível em: https://www.researchgate.net/publication/306398823_Methods_for_dependability_analysis_of_small_satellite_missions. Acesso em: 4 fev. 2018.

SERIFI, V.; DASIC, P.; JECMENICA, R.; LABOVIC, D. Functional and information modeling of production using IDEF methods. **Journal of Mechanical Engineering**, v.55, p.131-140, 2008. Disponível em: https://www.researchgate.net/publication/235636672_Functional_and_information_modeling_of_production_using_IDEF_methods. Acesso em: 25 fev. 2018.

SHEKOOFA, O.; POURYAIE, N.; SOHRABZADEH, K. P.; REZVANI, M. Improving the EGSE of power subsystem by design and development of a battery emulator for space applications. In: INTERNATIONAL CONFERENCE ON RECENT ADVANCES IN SPACE TECHNOLOGIES, 4., 2011. Turquia. **Proceedings…** 2011. p. 1-2. Disponível em: https://ieeexplore.ieee.org/document/5158294. Acesso em: 26 set. 2017.

SCHIEFERDECKER, I.; GROSSMANN, J.; SCHNEIDER, M. Model-based security testing. **Computer Science - Software Engineering**, p.2-7, 2012. Disponível em: https://arxiv.org/pdf/1202.6118.pdf. Acesso em: 22 nov. 2017.

SCHIMIDT, D.; C. **Model-driven engineering**. IEEE Computer Society, 2006. Disponível em: https://www.fing.edu.uy/inco/grupos/coal/uploads/Main/mdepaper.pdf. Acesso em: 25 set. 2017.

SCHUCH, N.; DURÃO, O.; SILVA, M.; MATTIELLO-FRANCISCO, F.; SILVA, A. NANOSATC-BR Status: a joint Cubesat-based program developed by INPE and UFSM. In: IAA CONFERENCE ON UNIVERSITY SATELLITE MISSIONS AND CUBESAT WORKSHOP, 4., 2017. **Proceedings…** 2017. p. 2-4. Disponível em: https://www.researchgate.net/publication/322746790_NANOSATC-BR_STATUS_-_A_JOINT_CUBESAT-BASED_PROGRAM_DEVELOPED_BY_INPE_AND_UFSM. Acesso em: 23 jan. 2018.

SINGH, V.; RAMASAMY, S. An exploration of model based testing. **International Journal of Scientific & Engineering Research**, v. 6, n.2, p.2-4, 2015. Disponível em: https://www.researchgate.net/publication/273126286_An_exploration_of_model_based_testing. Acesso em: 4 mar. 2018.

SISTEMAS ESPACIAIS E TECNOLOGIA. **Documento de controle de interfaces do computador de bordo do satélite NanosatC-BR2**. [S.l.]: SET, 2015. p. 6-38.

SOUZA, H. C. S. **Construção automatizada de casos de teste usando engenharia dirigida por modelos**. 2009. Dissertação (Mestrado em Engenharia de Eletricidade) - Universidade Federal do Maranhão, São Luís, 2009.

SRINIVAS, N.; PANDITI, N.; SCHMIDT, S.; GARRELFS, R. MIL/SIL/PIL **Approach a new paradigm in model based development**. 2014. Disponível em: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/solutions/automotive/files/in-expo-2014/mil-sil-pil-a-new-paradigm-in-model-based-development.pdf. Acesso em: 17 dez. 2018.

UPPAAL 4.0: **Small tutorial**. 2009, Disponível em: http://www.uppaal.org/. Acesso em: 23 jul. 2016.

UTTING, M.; PRETSCHNER, A.; LEGEARD, B. **A taxonomy of model-based testing**. Hamilton, Nova Zelândia: The University of Waikato, 2006. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.520.8941&rep=rep1&type=pdf. Acesso em: 29 ago. 2015.

WELLER, A. C.; MARTINS, E.; MATTIELLO-FRANCISCO, M. F. InRob-UML: uma abordagem para testes de interoperabilidade e robustez baseados em modelos. In: BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE, 2015. Belo Horizonte **Proceedings…** 2015. Disponível em: http://cbsoft.org/articles/0000/0527/SAST.pdf. Acesso em: 17 mar. 2017.

YAKINDU. **Guia do usuário**. 2017. Disponível em: https://www.itemis.com/en/yakindu/state-machine/documentation/user-guide/. Acesso em: 15 dez. 2016.

APÊNDICE A - TABELAS DO ESTUDO DE CASO NANOSATC-BR2.

Tabela A.1 - Interação entre OBC e SLP.

OBC comandos	**Operação (ação)**	**Etapa (*Step*) OBC**	**Condição**	**SLP informações**	**Operação (ação)**	**Etapa (*Step*) SLP**
	Housekeeping, comandos, entre outras operações	1 *IDLE*			Operando	1 *ON*
0xf0 (hex) ou 240 (dec)	Saber se a SLP está disponível para responder comandos (*SlaveReady*)	2 *Send*				2 *Receive*
		3.1 *Receive*	SLP pronta para receber comandos	0xfa (hex) ou 250 (dec)	Disponível (*SlaveOK*)	3.1 *Send*
		Ou 3.2 *Receive*	# SLP ocupada.	Ou 0xfb (hex) ou 251 (dec)	Ocupada (*SlaveBusy*) Realizando experimento	Ou 3.2 *Send*
0xf1 (hex) ou 241 (dec)	Informa que será enviado a data e hora da interação (*SetDateTime*)	4 *Send*				4 *Receive*
		5.1 *Receive*	SLP reconheceu o comando do OBC	0xfb (hex) ou 252 (dec)	Reconheceu comando (*CmdOK*)	5.1 *Send*
		Ou 5.2 *Receive*	# SLP não reconheceu o comando do OBC. # SLP retorna a ler o primeiro comando. # OBC retorna a enviar o primeiro comando. # No caso do OBC não reconhecer o comando, retorna a enviar o primeiro comando.	Ou 0xfd (hex) ou 253 (dec)	Não reconheceu comando (*CmdUnknown*)	Ou 5.2 Send
Data e Hora	A data e hora são enviadas em seis bytes sendo um byte enviado por vez. O formato de envio é: DD (dia), MM (mês), AA (ano), hh (hora), mm (minuto), e ss (segundo)	6 *Send*				6 *Receive*
		7 *Receive*	SLP terminou de receber Data e Hora da interação	Após o recebimento do sexto byte a SLP reponde 0xfe (hex) ou 254 (dec)	Data e a hora foram recebidas (*DTOK*)	7 *Send*

continua.

Tabela A.1 - Continuação.

		8 *Receive*	SLP ocupada coletando dados do experimento	0xfb (hex) ou 251 (dec)	Ocupada (*SlaveBusy*) para a realização do experimento	8 *Send*
					Acquisition Experiment	9 *Acquisition_ Experiment*
			SLP ocupada coletando dados do experimento		Permanece Ocupada (*SlaveBusy*) com experimento	9.1
			SLP terminou coleta de dados do experimento		Término do experimento	9.2
0xf0 (hex) ou 240 (dec)	Saber se a SLP está disponível para responder comandos (*SlaveReady*)	9 *Send*	SLP terminou de coleta os dados do experimento		Após término do experimento (apôs 5 minutos do início do experimento)	10 *Receive*
		10 *Receive*	SLP pronta para receber comando	0xfa (hex) ou 250 (dec)	Disponível (*SlaveOK*)	11 *Send*
0xf2 (hex) ou 242 (dec)	Informa que será enviado o número de bloco de dados do experimento da SLP que serão lidos (*SetBloTx*)	11 *Send*				12 *Receive*
		12.1 *Receive*	SLP reconheceu o comando do OBC	0xfb (hex) ou 252 (dec)	Reconheceu comando (*CmdOK*)	13.1 *Send*
		Ou 12.2 *Receive*	# SLP não reconheceu o comando do OBC. # SLP retorna a ler o primeiro comando. # OBC retorna a enviar o primeiro comando. # No caso do OBC não reconhecer o comando, retorna a enviar o primeiro comando.	Ou 0xfd (hex) ou 253 (dec)	Não reconheceu comando (*CmdUnknown*)	Ou 13.2 Send
Número de blocos	*BloTx*	13 *Send*				14 *Receive*
		13.1 Não recebe retorno da SLP. Prepara para enviar novo comando				

continua.

Tabela A.1 - Continuação.

0xf3 (hex) ou 243 (dec)	Informa que será enviado o índice do número de bloco de dados do experimento da SLP que serão lidos (*SetBloIndex*)	14 *Send*				15 *Receive*
		15.1 *Receive*	SLP reconheceu o comando do OBC	0xfb (hex) ou 252 (dec)	Reconheceu comando (*CmdOK*)	16.1 *Send*
		Ou 15.2 *Receive*	# SLP não reconheceu o comando do OBC. # SLP retorna a ler o primeiro comando. # OBC retorna a enviar o primeiro comando. # No caso do OBC não reconhecer o comando, retorna a enviar o primeiro comando.	Ou 0xfd (hex) ou 253 (dec)	Não reconheceu comando (*CmdUnknown*)	Ou 16.2 *Send*
Índice do número de blocos	*BloIndex*	16 *Send*				17 *Receive*
		16.1 Não recebe retorno da SLP. Prepara para enviar novo comando				
0xf4 (hex) ou 244 (dec)	Informa que será solicitado o envido dos dados do experimento da SLP conforme índice e número de blocos (*StartTx*)	17 *Send*				18 *Receive*
		18.1 *Receive*	SLP reconheceu o comando do OBC	0xfb (hex) ou 252 (dec)	Reconheceu comando (*CmdOK*)	19.1 Send

continua.

Tabela A.1 - Conclusão.

		Ou 18.2 *Receive*	# PL não reconheceu o comando do OBC. # SLP retorna a ler o primeiro comando. # OBC retorna a enviar o primeiro comando. # No caso do OBC não reconhecer o comando, retorna a enviar o primeiro comando.	Ou 0xfd (hex) ou 253 (dec)	Não reconheceu comando (*CmdUnknown*)	Ou 19.2 *Send*
			SLP lendo os dados do experimento solicitados		*Read Experiment*	20 *Read_ Experiment*
		19 *Receive*	SLP envia os dados solicitados	Dados	Envio dos dados do experimento conforme solicitado	21 *Send*
	Grava dados do experimento	20 *Write*			Não terminou envio dos dados	
		3.1	SLP pronta para receber nova Data e Hora para aquisição de novos dados do experimento	0xfa (hex) ou 250 (dec)	Término do envio dos dados solicitados. (*SlaveOK*)	3.1

Fonte: Produção do autor.

Tabela A.2 - Estados, eventos, ações e condições do OBC.

Estados **OBC**	Eventos	Ações	Descrição	Condições	Descrição
OBC_IDLE	*SlaveOK*(250) *Step* 3.1/10		SLP pronta para receber comando. OBC enviará Data e Hora (SetDataTime / 3.1) ou número de bloco de dados (*SetBloTx* / 10)	SendPL=250 e **WriteOBC**=0	SLP pronta para receber comando e OBC não iniciou gravação de dados (**WriteOBC**)
	SlaveBusy(251) *Step* 3.2	**y**+=1	Incrementa variável contadora de tempo (y) para aguardar SLP terminar leitura de dados. Aguardo de 5 minutos. OBC enviará comando perguntando se *SlaveReady*	SendPL=*SlaveBusy*(251) e **y**<=**N**	SLP ocupara lendo experimento (*SlaveBusy*) e Contador de tempo **y** menor ou igual ao tempo de aguardo **N** (= 5 minutos).
	SlaveBusy(251) *Step* 3.2/8		OBC enviará comando perguntando se SlaveReady	SLP=251 e **WriteOBC** not= 1 e **y**=1	SLP ocupada lendo experimento, Variável indicando que OBC não iniciará gravação de dados (**WriteOBC**) e Inicializa variável contadora de aguardo de tempo de comunicação com a SLP.
	CmdOK(252) e SendOBC not= *StartTx*(244) *Step* 5.1/12.1/15.1	i+=1	Incrementa variável contadora de bytes de Data e Hora (*i*). OBC enviará (DDMMAAhhmmss / 5.1) ou (*BloTX* / 12.1) ou (*BloIndex* / 15.1)	SendPL=252 e SendOBC not=244	SLP informou que reconheceu comando do OBC (252) e OBC não solicitou leitura de dados (244)
	CmdOK(252) ou SendBlo=1 *BloTx* ou *BloIndex* *Step* 13.1/16.1	i+=1	Incrementa variável contadora de bytes de Data e Hora (i). OBC enviará (*SetBloIndex* ou *StartTx*).	(SendPL=252 ou SendBlo=1) e (**i**<6 ou **l** not=0) e (SendOBC not =241, 242, 243, 244)	SLP informa que reconheceu o comando do OBC (252) ou OBC enviará número de bloco de dados a serem lido (*SendBlo*) e Variável contadora de bytes de data e hora menor que seis (**i**) ou será informado número e índice de bloco de dados a ser lido (**l**) e OBC envia informação.
	CmdUnknown(253) *Step* 5.2/12.2/15.2/18.2		SLP não reconheceu comando do OBC. OBC ficará enviando comando perguntando se *SlaveReady*	SendPL not=250 e not=251 e not=252 e not=254	OBC não reconheceu comando da SLP
OBC_Send	*CmdUnknown*(253) ou *SlaveBusy*(251) *Step* 2/9	SendOBC= *SlaveReady*(240) SendDT=0	OBC envia comando perguntando se SLP está pronta para receber comando (240) e Inicializa variável indicando que será enviado Data e Hora (**SendDT**)	(SendPL not=250 e not=252 e not=254 e WriteOBC=0) ou SendPL=251	OBC não reconhece comando da SLP e não iniciou gravação de dados (**WriteOBC**) ou SLP ocupada realizando experimento (252)

continua.

Tabela A.2 - Continuação.

	SlaveOK(250) *Step* 4	SendOBC= *SetDateTime*(241) i=0 l=0	OBC informa que será enviado data e hora do momento da interação. Inicializa variável para contar seis bytes do envio da Data e Hora (i). Inicializa variável indicara que será enviado o número de bloco de dados a serem lido (l)	SendPL=*SlaveOK*(250) e **y**=0	SLP está livre para troca de informação. Inicializa contador de tempo para espera de leitura de dados da SLP (**y**).
	CmdOK(252) *Step* 6.1	SendOBC= **DD** SendDT=1	Envio do primeiro byte referente ao dia (**DD**). Variável indicadora que inicio envio de dados de data e hora	SendPL =252 e **i**=1 e **l**=o	SLP reconheceu comando do OBC informando que será enviada Data e Hora. Índice de Data indicando primeiro byte. Indica que o número de blocos ainda não foi iniciado.
	CmdOK(252) *Step* 6.2	SendOBC= **MM**	Envio do segundo byte referente ao mês (**MM**)	SendPL=252 e **i**=2 e **l**=0	SLP reconheceu comando do OBC informando que será enviada Data e Hora. Índice de Data indicando segundo byte. Indica que o número de blocos ainda não foi iniciado.
	CmdOK(252) *Step* 6.3	SendOBC= **AA**	Envio do terceiro byte referente ao ano (**AA**)	SendPL=252 e **i**=3 e **l**=0	SLP reconheceu comando do OBC informando que será enviada Data e Hora. Índice de Data indicando terceiro byte. Indica que o número de blocos ainda não foi iniciado.
	CmdOK(252) *Step* 6.4	SendOBC=**hh**	Envio do quarto byte referente à hora (**hh**)	SendPL=252 e **i**=4 e **l**=0	SLP reconheceu comando do OBC informando que será enviada Data e Hora. Índice de Data indicando quarto byte. Indica que o número de blocos ainda não foi iniciado.
	CmdOK(252) *Step* 6.5	SendOBC= **mm**	Envio do quinto byte referente ao minuto (**mm**)	SendPL=252 e i=5 e l=0	SLP reconheceu comando do OBC informando que será enviada Data e Hora. Índice de Data indicando quinto byte. Indica que o número de blocos ainda não foi iniciado.
	CmdOK(252) *Step* 6.6	SendOBC=**ss** **SendDT**=0	Envio do sexto byte referente ao segundo (**ss**). Variável indicadora que terminou o envio de data e hora	SendPL=252 e i=6 e l=0	SLP reconheceu comando do OBC informando que será enviada Data e Hora. Índice de Data indicando sexto byte. Indica que o número de blocos ainda não foi iniciado

continua.

Tabela A.2 - Continuação.

	SlaveOK(250) *Step* 11	SendOBC= *SetBloTx*(242) SendBlo=0 **I**=1 i=6	OBC informa que será enviado número de bloco de dados do experimento da SLP que deverá ser enviado. Inicializa variável indicadora que será informado o número e índice de bloco de dados a ser lido (**I**). Indica que os seis bytes de Data e Hora já foram enviados (i)	SendPL=250 e **y** not=0	SLP está disponível para troca de informação (250) e OBC aguardou a SLP coletar dados do experimento (**y** not=0)
	CmdOK(252) *Step* 13	SendOBC= ***BloTx*** **SendBlo**=1 **I**+=1	OBC informa número de blocos de dados do experimento que a SLP deverá enviar (***BloTx***). Inicializa variáveis indicadoras que número e índice de bloco de dados estão sendo enviados (**SendBlo**) (**I**).	SendPL=252 e **I**=1	SLP reconheceu comando (252) e iniciou envio do número de blocos de dados a serem lidos (**I**=1).
	CmdOK(252) *Step* 14	SendOBC= ***SetBloIndex***(243) **SendBlo**=0 **I**+=1	OBC informa que será enviado o índice do bloco de dados que deverão ser enviados pela SLP. O número de blocos já foi informado (**SendBlo**=0) Incrementa variável indicando que agora será informado o índice do bloco (**I**)	SendPL=252 e **I**=2	SLP reconheceu comando (252) e indica que será informado o índice do bloco (**I**=2)
	CmdOK(252) *Step* 16	SendOBC= ***BloIndex*** **SendBlo**=1 **I**+=1	OBC informa o índice do bloco de dados que deverá ser enviado. Inicializa variável que indica informação sobre blocos a serem lidos (**SendBlo**). Incrementa variável indicando que agora será informado o índice do bloco (**I**)	SendPL=252 e **I**=3	SLP reconheceu comando (252) e índice do bloco de dados está sendo enviado (**I**=3)
	CmdOK(252) *Step* 17	SendOBC= *StartTx*(244) **SendBlo**=0 **y**=0	OBC informa que será solicitada a leitura dos dados do experimento da SLP. Inicializa variável indicando que número de blocos e índice já foi informado (**SendBlo**=0). Inicializa variável contadora de tempo para espera da SLP coletar dados do experimento (**y**)	SendPL=252 e **I**=4	SLP reconheceu comando (252) e os dados do experimento serão solicitados conforme índice e número de blocos de dados (**I**=4)

continua.

Tabela A.2 - Conclusão.

OBC_Receive	*DTOK*(254) *Step* 7	**y**=1	OBC ficará enviando comando perguntando se SlaveReady e Inicializa variável contadora de aguardo de tempo de comunicação com a SLP.	SendPL=254	SLP recebeu Data e Hora
	CmdOK(252 e *StartTX*(244) *Step* 18.1	**WriteOBC**=1 **i**=0	OBC iniciará gravação dos dados do experimento (**WriteOBC**=1). Inicializa variável contadora do bloco de dados a serem lidos (**i**)	SendPL=252 e SendOBC=244 e **WriteOBC**=0	SLP reconheceu comando enviado pelo OBC (252), OBC solicitou envio dos dados do experimento da SLP e OBC ainda não gravou dados do experimento da SLP.
OBC_Write	*StartTx*(244) *Step* 19	**OBCData**[**i**] =SendPL i+=1	Os dados lidos do buffer da SLP são transferidos e gravados na área de armazenamento do OBC (**OBCData**[i]. Incrementa variável contadora do índice de dados a serem lidos (**i**)	**WriteOBC**=1 e SendOBC = 244 e i<***BloTx***+6	Leitura e gravação dos dados do experimento da SLP (**WriteOBC**). Solicitação de envio de dados (244). O número de blocos solicitados será lido (***BloTX***) mais a data e hora da interação (seis bytes).
	Step 20		OBC grava dados do experimento	**i**<***BloTx***+6	Envio dos dados ainda em andamento
	Dados do experimento *Step* 20	**y**=0	Inicializa variável contadora de tempo para espera a SLP coletar dados do experimento (**y**)	**i**>=***BloTx***+6	Término da gravação dos dados
	not = *CmdOK*(252)	**WriteOBC**=0	OBC inicializa variável para gravação de dados (**WriteOBC**)	SendPL not=252 e **i**>=N	Término de envio dos dados

Fonte: Produção do autor.

Tabela A.3 - Estados, eventos, ações e condições da SLP.

Estados **SLP**	Eventos	Ações	Descrição	Condições	Descrição
SLP_ON	*CmdUnknown*(253) *Step* 5.2/13.2/16.2/19.2 *Step* 1		Não reconheceu comando do OBC e está em operação aguardando comando	SendOBC not=240, 241, 242, 243, 244 e SendPL=0	Não reconheceu comando do OBC e SLP não enviou informação
SLP_Acquisition_Experiment	SendPL=*SlaveBusy*(251) *Step* 9			SendPL=*SlaveBusy*(251) e **x**=6	SLP ocupada com experimento (251) e a variável que apontará o tamanho do buffer da SLP (**x**) indica que os seis bytes de Data e Hora já foram registrados
	SendPL=*SlaveBusy*(251) *Step* 9.1	SendPL=251 **SLPBuffer**[**j**-1]=**j**-1 **j**+=1 **x**+=1	SLP fica no estado ocupado realizando experimento (251). SLP armazena em seu buffer dados do experimento a partir do sétimo byte, pois os seis primeiros estão a Data e Hora. j = contador de índice do buffer. (Neste exemplo estão sendo armazenados apenas números do contador do buffer) **x** = contador do tamanho do buffer da SLP.	**x**<**N**	Armazenamento dos dados do experimento até completar o buffer da SLP, o qual tem capacidade para 300 pacotes de 100 bytes. **N** = variável indicando tamanho do buffer em pacotes (**N**=300)
SLP_Receive	SendPL=*SlaveOK*(250) *Step* 9.2	SendPL=*SlaveOK*(250) **SendBuffPL**=0	SLP pronta para receber informação do OBC. SLP inicializa variável indicadora que serão enviados dados do buffer para o OBC (**SendBuffPL**).	**x**>=**N**	Dados do experimento armazenado na capacidade do buffer da SLP
	j *Step* 6	**SLPBuffer**[**j**-1]=SendOBC **j**+=1	OBC envia Data e Hora para serem armazenados no buffer da SLP. j = variável contadora do número do índice do buffer da SLP. Ao término do recebimento da Data e Hora SLP envia *DTOK*.	**j**>0 e **j**<7	SLP lendo Data e Hora fornecidas pelo OBC
	BloTxSLP *Step* 14	***NBlo***=SendOBC	SLP recebeu e armazenou o número de blocos de dados do experimento para serem transmitidos (***NBlo***). **SendBloSLP** = variável indicadora que SLP armazena o número de bloco de dados	**SendBloSLP**=1 e ***NBlo***=0 e SendOBC not=241, 244, 0	SLP receberá o número de bloco de dados do experimento (**SendBloSLP**), a variável que receberá o valor foi inicializada (***NBlo***=0) e SLP irá ler número de bloco de dados

continua.

Tabela A.3 - Continuação.

	SendOBC	**SendData**=1	SLP receberá o índice dos blocos de dados a serem transmitidos	(***NBlo*** not=0 e **SendBloSLP**=1 e SendOBC not= 243 e SendOBC not=244) ou (***NBlo*** not=0 e **Sendindex**=0 e SendOBC not=244	SLP recebeu número dos blocos (***Nblo***) e OBC envio comando indicando que enviará o índice dos blocos de dados
	BloIndex *Step* 17	**IndexBlo**=SendOBC **Sendindex**=0	SLP armazena em variável número do índice dos blocos (***IndexBlo***). **SendIndex** = variável indicadora que o índice do bloco de dados já foi recebido	SendPL=252 e SendOBC not=244 e **Sendindex** =1	SLP reconheceu comando do OBC e o índice do bloco de dados está sendo recebido
	SlaveReady(240) *Step* 3.1/11		OBC enviará *SetDataTime* (3.1) ou *SetBloTx* (11)	SendOBC=*SlaveReady*(240) e SendPL=*SlaveOK*(240) ou (SendOBC=*StartTX*(244) e SendPL=*SlaveOK*(250))	SLP está pronta para receber comandos do OBC ou OBC solicitará a transmissão dos dados
	CmdOK(252) *Step* 5.1/13.1/16.2		Será enviado (DDMMAAhhmmss / 5.1), ou (***BloTx*** / 13.1), ou (***BloIndex*** / 16.1)	SendPL=*CmdOK*(252) e SendOBC not *StartTx*(244)	SLP reconheceu comando do OBC e OBC ainda não solicitou transmissão dos dados
	SlaveOK(250) *Step* 3.1/11 *CmdUnknown*(253) *Step* 5.2/13.2/16.2/19.2		Será informado *SlaveReady*	SendPL=*SlaveOK*(250) ou SendPL=*CmdUnknown*(253)	SLP pronta para receber comandos do OBC ou SLP não reconheceu comando do OBC
SLP_Send	*SlaveReady*(240) *Step* 2/10	SendPL=250 **j**=0	SLP pronta para receber comandos do OBC (*SlaveOK*) Índice de gravação do experimento iniciado (**j**).	SendOBC=*SlaveOK*(240)	SLP pronta pra receber comandos

continua.

Tabela A.3 - Continuação.

	SetDataTime(241) *Step* 4	SendPL=*CmdOK*(252) SendBloSLP=0 **Sendindex**=0 **SendData**=0 ***IndexBlo***=0 ***NBlo***=0 **j**=1 **x**=0 **k**=0	SLP reconheceu comando (*CmdOK*). Inicializou variável indicadora que será transmitido número de bloco de dados a serem lidos (**SendBloSLP**). Inicializou variável indicadora que será recebido índice do número de blocos (**SendIndex**). Iniciou variável para enviar dados do experimento. Inicializou variável registradora do número do índice (***IndexBlo***). Inicializou variável registradora do número de blocos (***NBlo***). Inicializou variável contado do índice do buffer da SLP (**x**). Inicializou variável contadora do índice dos dados a serem transmitidos (**k**).	SendOBC=241 e **j**=0	OBC fornecerá a Dada e Hora da interação e Índice de gravação do experimento iniciado
	j (indicando Data e Hora já transmitidas) *Step* 7	SendPL=*DTOK*(254)	SLP indica que Data e Hora já foram recebidas (DDMMAAhhmmss).	**j**=7	SLP recebeu Data e Hora da interação
	SetBloTx(242) *Step* 12	SendPL=*CmdOK*(252) **SendBloSLP**=1	SLP reconheceu comando do OBC. Será recebido número de blocos a serem transmitidos (**SendBloSLP**)	SendOBC=242 e SendPL not= 252	OBC informa que enviará o número do bloco de dados a serem lidos do experimento da SLP
	SetBloindex(243) *Step* 15	SendPL=CmdOK(252) **SendBloSLP**=0 **Sendindex**=1	SLP reconheceu comando (252). O número de blocos de dados já foi enviado (**SendBloSLP**) O número do índice do bloco de dados será enviado (**Sendindex**).	SendOBC=***SetBloindex***(243) e **Sendindex** not= 1	OBC informa que enviará o número do índice do bloco de dados a ser fornecido pela SLP e ainda não recebeu número do índice
	StartTx(244) *Step* 18	SendPL=*CmdOK*(252) **k**=0	SLP reconheceu comando (252). Variável contadora do índice de blocos a serrem transmitidos foi inicializada (**k**)	SendOBC=*StartTx*(244) e **SendData**=1	OBC informa que solicitará a leitura dos dados do experimento da SLP e serão enviado os dados do experimento

continua.

Tabela A.3 - Conclusão.

	CmdUnknown Step 5.2/13.2/16.2/19.2	SendPL=*CmdUnknown*(253)	SLP não reconheceu comando recebido pelo OBC	SendOBC not=240 e SendOBC not=241 e SendOBC not=242 e SendOBC not=243 e SendOBC not=244 e **NBlo** not=0 e **Sendindex** not=0	SLP não reconheceu comando recebido
	DTOK(254) *Step* 8	SendPL=(*SlaveBusy*(251) **x**=6	Data e Hora foram recebidas pela SLP. Variável contadora do índice do buffer da SLP foi inicializada no sexto byte (**x**)	SendPL=254 e **x**=0	SLP recebeu Dada e Hora do OBC
	SLPBuffer *Step* 21	**k**=**k**+**IndexBlo** SendPL=**SLPBuffer**[**k**] **k**+=1	Posiciona o índice dos blocos de dados, solicitados pelo OBC, no buffer da SLP. SLP enviando dados do experimento para o OBC. **k** = variável contadora do índice do buffer da SLP	**k**=6	Variável contadora de índice indica que data e hora já foram transmitidas
	SLPBuffer *Step* 21	SendPL=**SLPBuffer**[**k**] **k**+=1	SLP enviando dados do experimento para o OBC. **k** = variável contadora do índice do buffer da SLP	**k** not=6	Variável contadora de índice indica que data e hora inda não foram transmitidas
	k	SendPL=*SlaveOK*(250) **x**=0 **NBlo**=1**j**=0	SLP transmitiu dados solicitados e está pronta para receber comandos. Variável contadora de índice do buffer da SLP inicializada (**x**). Inicializa variável que receberá o número de bloco de dados. Inicializa índice de data e hora.	**k**>=**NBlo**+6 e SendPL not= 250	SLP terminou de transmitir dados solicitados pelo OBC. **k** = índice do buffer da SLP ***NBlo*** = número de blocos solicitados (+ 6 porque os seis primeiros bytes são Data e Hora) e SLP ainda não está pronta pra receber comandos do OBC (*not OK*)
SLP_Read_Experiment	SendOBC=*StatTx*(244) Step 19.1			SendOBC=244 e **k**<***NBlo***+6	OBC solicitou envio dos dados do experimento da SLP (244) e os blocos de dados solicitados ainda não foram transmitidos

Fonte: Produção do autor.

Tabela A.4 - Codificação do FEM.

Nas linhas de comandos está descrito o que é realizado após duas barras (//).

```
#include <SoftI2CMaster.h> // biblioteca para definir porta SCL e SDA
#include <Wire.h> // biblioteca I2C
#define SCL 6 // segunda porta SCL
#define SDA 7 // segunda porta SDA
// SoftI2CMaster é a biblioteca chamada para definir as portas SCL e SDA
SoftI2CMaster i2c = SoftI2CMaster(SCL,SDA,0);
#define DATA_SIZE 32 // tamanho da variável
const uint8_t SLV_ADDR = 0x3C; // endereço do esrcravo
const uint8_t MST_ADDR = 0x03; // endereço do escravo
byte x[DATA_SIZE],y[DATA_SIZE]; // tamanho da variável
unsigned int mst_msg=0,slv_resp=0,fault;
// injecao de bitflip
 byte bitflip(byte who, int where){
   //Serial.print("\n");
   //Serial.println(who,BIN);
   bitWrite(who,where,!bitRead(who,where));
   //Serial.println(who,BIN);
   return who;
 }
// provision faults
 byte provision(byte who){
   return who;
 }
// time related faults
 void timeout(int who){
   delay(who);
 }
void setup() {
  Serial.begin(9600); // velocidade da porta serial
  Wire.begin(SLV_ADDR); // inicia I2C
  i2c.begin();
  Wire.onReceive(callReceive); // chama função
  Wire.onRequest(callRequest); // chama função
  randomSeed(analogRead(0));
}
void loop(){
//  fault = random(0x7F);
}
```

```
void callReceive(int numBytes){
  mst_msg++;
  i2c.beginTransmission(SLV_ADDR); // inicia transmissão I2C
  for(int i=0;i<numBytes;i++){
    x[i]=Wire.read(); // lê I2C
    if(mst_msg==6){x[i]=bitflip(x[i],0);} // bitflip
     i2c.write(x[i]); // escreve I2C
   }
  i2c.endTransmission(); // fim transmissão I2C
  Serial.print("\n"); // envia informação para serial
  Serial.print("Write");
  Serial.print("\t");
  Serial.print(mst_msg);
  Serial.print("\t");
  Serial.print("Read");
  Serial.print("\t");
  Serial.print(slv_resp);
  if(mst_msg==6){
    Serial.print("\t");
    Serial.print("Bit-Flip on Writing from Master");
  }
}
void callRequest(){
  slv_resp++;
  i2c.requestFrom(SLV_ADDR,DATA_SIZE); // solicita informação do escravo
  delay(100000);
  for(int i=0;i<DATA_SIZE;i++){
     x[i]=i2c.read();
    if(slv_resp==2){x[i]=bitflip(x[i],7);} // bitflip
    Wire.write(x[i]); // escreve no I2C
  }
  Serial.print("\n"); // envia informação para serial
  Serial.print("Write");
  Serial.print("\t");
  Serial.print(mst_msg);
  Serial.print("\t");
  Serial.print("Read");
  Serial.print("\t");
  Serial.print(slv_resp);
  if(slv_resp==2){
    Serial.print("\t");
    Serial.print("Bit-Flip on Reading from Slave");
  }
}
```

Fonte: Batista et al. (2018).

ANEXO A - ARTIGO PUBLICADO: IAA-LACW, II LATIN AMERICAN IAA CUBESAT WORKSHOP, FLORIANÓPOLIS, BRAZIL, FEBRUARY 28 - MARCH 3, 2016, "ON THE USE OF NANOSATC-BR TEST SYSTEM FOR PAYLOAD OPERATIONAL REQUIREMENTS VERIFICATION"

IAA-BR-16-0S-0P

Carlos Augusto P L Conceição, Ana Maria Ambrósio**, Fátima Mattiello-Francisco***.*

The need for a global quick and efficient communication, observation and understanding of events on Earth and conquest of space, motivates space technology development. The CubeSat standard, also known as U-Class nanosatellite platform, has enabled the flight qualification of innovative space technologies developed in academic environment and / or emerging companies in the sector. For over three decades leading research and development satellites in Brazil, the National Institute for Space Research (INPE) has supported over the past five years the development of nanosatellites projects in INPE´s regional centers at Northeast and South of Brazil in cooperation with local universities. In this context, NanosatC-Br family has been developed. First satellite, NanosatC-Br1, is a 1U Cubesat launched in July 2014 for purposing of both collecting the Earth's magnetic field data and measuring in flight the radiation resilience of integrated circuits designed in Brazil. The qualification of embedded software systems is one of the main challenges of the second mission, a 2U Cubesat named NanosatC-Br2 that is planned to be launched in 2016. The use of existing components on the market (COTS) added to the standardization of on board subsystems in nanosatellites platforms have allowed to reduce significantly the space mission development cycle enabling new space technologies being qualified on flight at low-cost. However, the verification and validation activities (V&V) at different stages of the space project lifecycle are still required and onerous in terms of resources and time. At least functional and dependability aspects of the payload integration with the satellite platform need to be systemically tested. Aiming to avoid the development of new test environment every new mission of a nanosatellite family, which spends time, hardware and software resources, this article presents a reusable Test System for NanoSatC-BR family. The reusability issues of Test System are addressed in two perspectives: (i) reuse the Test System at different stages of the same mission; (ii) reuse the Test System in different satellites of the same family. The proposed Test System architecture supports the V&V process focusing on interoperability features between the NanosatC-Br onboard computer and its payloads, aided by fault injection mechanisms. The goal is to verify and validate the behavior of these subsystems in nominal and exception scenarios through communication in the satellite bus. Experiments using the proposed Test System prototype for NanosatC-Br2 payload operational requirements evaluation are discussed in the article.

INTRODUCTION

Due to the short duration of the nanosatellite development cycle and the low cost project, the use of nanosatellite missions has been growing for qualification purposes of new technologies in space. However, nanosatellites present only 48% of success in carrying out missions, and the main reason to get this low percentage is related to the low number of tests performed [2].

The process of V&V adopted in space mission's project cycle has its importance in ensuring the quality of development of the space segment: the satellite platform, integration with payloads and operation. The different V&V techniques are used to verify compliance with the requirements of the subsystems in the different phases of the life cycle of a space project seeking improvement, safety and confidence in the operation. In the V&V process tests are used for verifying the dynamic behavior of systems. The use of computational tools in the tests is essential to support measurements that highlight the occurrence of faults in the development of subsystems. Tests are widely used to verify functional and dependability aspects of the subsystems in isolation or integrated. Thus expected interoperability among payloads and the satellite platform subsystems can be verified during the integration activities by means of measurements in the communication channel [1].

This article presents a reusable Test System for a nanosatellite family whose goal is to support verifying the expected behavior of the components (subsystems) under communication perspective, in nominal and exception scenarios in different stages of the development.

This paper is organized as following: Section 1 presents the nanosatellites family (NanoSatC-Br) whose program is developed by INPE in cooperation with Brazilian universities; Section 2 presents the architectural standard adopted by nanosatellites Br-1 and Br 2, in particular the use of I2C standard as communication backbone in the CubeSat; Section 3 describes the proposed Test System and four possible scenarios for reusing the system; Section 4 presents the Test System prototype environment that includes measurement facilities and discusses experiments using this prototype. Finally, section 5

concludes the paper with discussions about the reusability issues of Test System in two perspectives: (i) reuse the Test System at different stages of the same mission; (ii) reuse the Test System in different satellites of the same family.

1. NANOSATC-BR PROGRAM

The National Institute for Space Research (Instituto Nacional de Pesquisas Espaciais - INPE) has developed research in nanosatellites platforms seeking to acquire knowledge in innovative technologies and increase graduate student interest in space engineering. Currently, INPE has in orbit the NanosatC-Br1 launched in July 2014 for purposing of both collecting the Earth's magnetic field data and measuring in flight the radiation resilience of integrated circuits designed in Brazil. The second satellite of this family is the NanosatC-Br 2, a 2U Cubesat under development, which is planned to be launch in 2016. The qualification of embedded software systems is one of the main challenges of the second mission. It will be the first Brazilian nanosatellite with on board data handling and attitude control subsystems developed in the country [6].

The NanosatC-Br family uses the standard Cubesat. The BR-1, 1U platform based, was fully acquired as COTS and the payloads on board this first scientific mission in nano category in Brazil were developed in universities. They are: (i) one magnetometer; (ii) one ASIC circuit custom designed by the customer; (iii) one algorithm designed to run in the FPGA [4].

The BR-2 mission uses a 2U platform. The data handling software and the attitude control system will be national solutions developed at INPE in collaboration with the Technological Institute of Aeronautics (Instituto Tecnológico de Aeronáutica - ITA) and emerging software companies. There will be 5 payloads on board BR-2 concentrated in three boards: (i) Langmuir probe: measuring the numeric density of electrons, its kinetic temperature and the spectral distribution of plasma irregularities; (ii) Attitude Determination System (DAS): scientific payload, identified as a single printed circuit board comprising 3 microcontrollers a 3-axis magnetometer XEN-1210 model oscillating crystals, resistors, capacitors and connectors; (iii) NCBR1_v2

Experiments (SMDH, FPGA, Magnetometer) compacted in the same board: is an evolution of the mission NanosatC-Br1 [9].

2. ARCHITECTURE OF NANOSATC-BR

The NanosatC-Br architecture consists of the following subsystems and their functions, as diagram in Figure 1:

- On-board computer (OBC): dispatching commands and collecting housekeeping and payload data through the satellite bus;
- ISIS TRXUV VHF/UHF Transceiver: enables the CubeSat to have a full duplex system with telemetry, telecommand & beacon capabilities on a single board;
- NanoPower P30U power supply: designed power demands from 1-30W;
- ISIS AntS Electrical model: inputs for the implementation of the antennas (Engineering Model);
- ISIS Generic interface system (IGIS): standard set of hardware interfaces between integrated satellites and electronic ground support equipment;
- BOB - "break-out board": assistance in communication between the satellite boards;
- ISIS Antenna System (AntS): detachable antenna system VHF / UHF (Flight Model);
- NanoPower Solar panels: supply power to the satellite.

Interoperability between the satellite subsystems, including payloads, is done through the I2C communication bus [2]. The onboard computer has a fundamental role on communication management being responsible for data handing and housekeeping tasks.

EGSE (Electrical Ground Support Equipment) is provided by Cubesat platform supplier for all NanoSat-BR family engineering models. EGSE allows to boot satellite components and to startup tests from the OBC and payloads.

Functional tests are performed by means of telecommand and telemetry with

the support of radio transmission equipment RF Checkout. However, the facilities provided for test execution need to be improved in terms of controllability and observability.

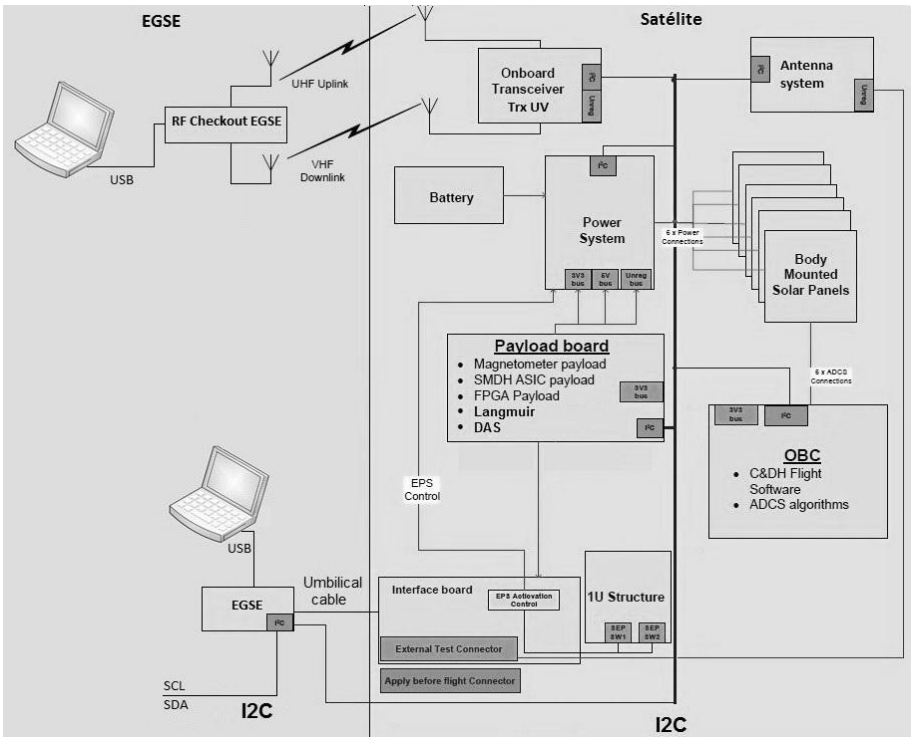


Fig.1. NanosatC-Br communication architecture using I2C (modified) [3].

3. PROPOSED TEST SYSTEM

The proposed Test System supports the V&V process by means of interoperability measurements among the NanosatC-Br onboard computer and its payloads, aided by fault injection mechanisms. The goal is to verify and validate the behavior of these subsystems in nominal situation and exception conditions through the communication bus I2C.

The Test System is based on a simple and low-cost architecture which uses components available on the market. The basic components are computer boards ARDUINO [7] that can be added to this architecture whether necessary for further testing, allowing reuse of the Test System at different stages of development of the satellite. There are also available on the market many ARDUINO computer applications and libraries that support the emulation of payload functions. In addition, free software tools can be found to support, for

instance, performance analysis in the communication channel.

The Test System consists of: (i) one Arduino board (board 1) to emulate the on-board computer, (ii) one Arduino board (board 2) to emulate payloads, (iii) one Arduino board (board 3) to analyze the traffic of information on the I2C bus and inject failures in order to check the behavior of the satellite subsystems in normal and adverse conditions, (iv) one CI 74HC595, which has the shift register function, and (v) one computer to perform the functions that the Arduino boards will be submitted.

Figure 2 extends the architecture of NanoSatC-Br2 (Figure 1) with the proposed Test System.

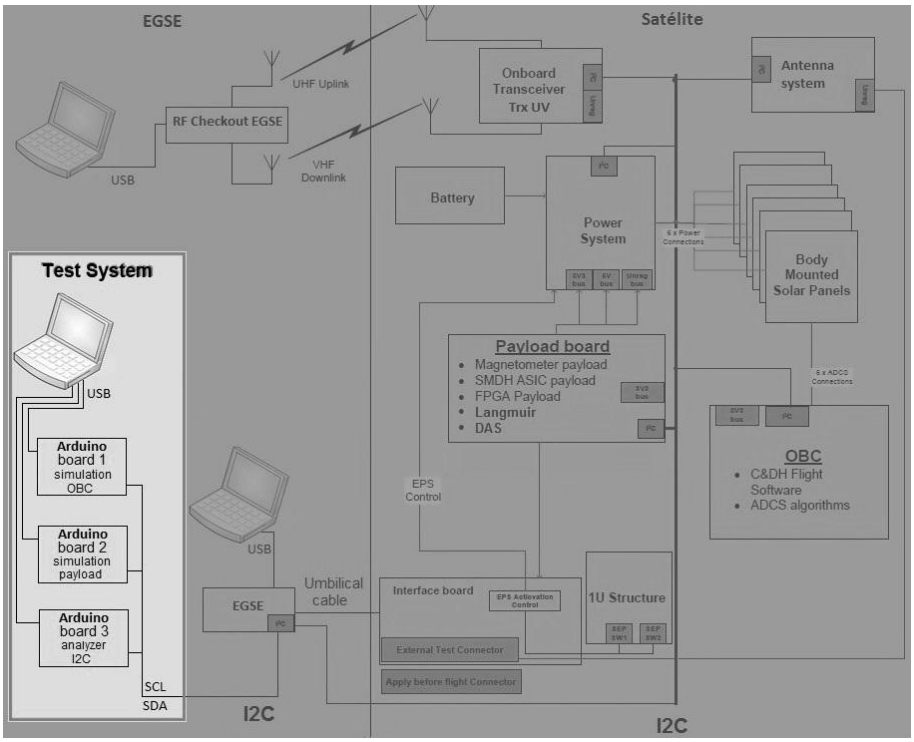


Fig.2. Test system connected to the I2C.

One can observe that I2C is considered the backbone of the Test System architecture, allowing it to be expandable and reconfigured to support V&V activities in different satellite development stages, from mission conception to acceptance / integration of payloads. For instance, in mission conception phase, payload requirements can be evaluated in terms of data rate in I2C by means of payload and OBC emulations using only the Test System architecture

highlighted in Figure 2. In advanced phases of system development, the Test System can be expanded to the engineering model, replacing emulated components incrementally by hardware in the loop. Therefore four test scenarios are designed to reuse the proposed system.

3.1. Test scenarios

A test scenario is the configuration of the resources provided in the Test System architecture for a particular purpose.

- Scenario A - supports the verification of subsystem operating requirements in the mission conception phase. The purpose is to analyze the demands of telemetry and telecommand of payloads. In this scenario the Arduino boards emulate the OBC (board 1) and payloads (board 2);
- Scenario B – the goal is to test the interoperability of subsystems on the bus considering the OBC hardware in the loop, but an Arduino board (board 2) still emulates the payloads;
- Scenario C – the goal is to test the interoperability of subsystems on the bus by placing the payload hardware in the loop, but an Arduino board (board 1) still emulates the OBC;
- Scenario D - supports the verification of the operational requirements of the mission subsystem. The goal is to analyze the telemetry and telecommand demands of the payloads, considering in this scenario both hardware OBC and payloads in the loop.

In all scenarios there will be an additional Arduino board (board 3) that will perform the reading of information that is traveling in the I2C bus for measurement purposes.

Figure 3 shows the reuse of the Test System architecture for the scenarios described above:

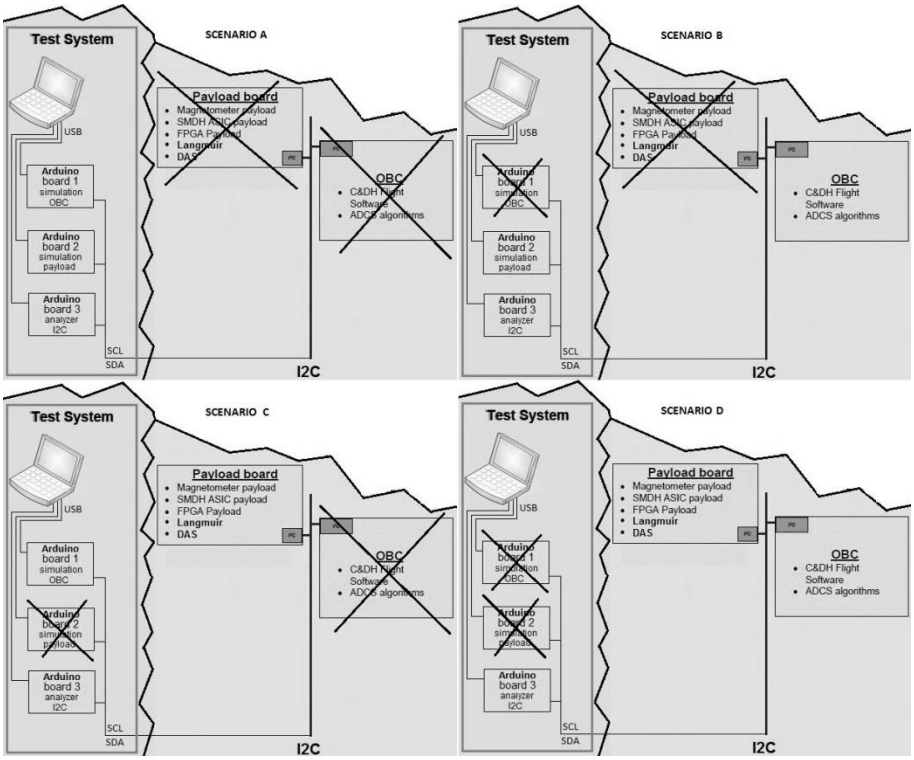


Fig. 3. Reusable Test System focusing on the I2C bus.

3.2. Scenarios Configuration

This section describes the steps necessary for the Test System configuration in order to support the execution of each scenario.

Scenario A:

- Set three Arduino boards on an I2C bus. One board represents OBC being the master (board 1), another represents a payload being the slave (board 2) and third (board 3) will read the information that travel on the I2C in order to check whether they are in conformance to the design requirements.
- Send information from board 1 to board 2 simulating a telecommand sent by OBC to a payload.
- Check that the information received by the board 2 is conform the

information sent by the board 1.

- Send the information from board 2 to the board 1 simulating a telemetry sent by a payload to the OBC.
- Check that the information received by the board 1 is conform the information sent by the board 2.

Scenario B:

- Set two Arduino boards and connect to I2C bus of the satellite engineering model in such a way that real OBC will be in the loop as master. One board will be set to slave (board 2) with a payload address and the other board (board 3) will read the information that is traveling on the bus and inject faults in order to check the satellite OBC behavior to receive information failed.
- Send information from board 2 simulating a payload sending telemetry to the OBC.
- Check that the information received by the OBC is correct.
- Send information from board 2, read this information by board 3, inject a particular fault and relay that information to the OBC.
- Check whether the OBC behaviors conform expected in such a foreseen condition.

Scenario C:

- Set two Arduino boards and connect to I2C bus of the satellite engineering model in such a way that real payload will be in the loop as slave. One board will be set to master with the OBC address (board 1) and the other board (board 3) will read the information that is traveling on the bus and will inject faults in order to check the satellite payload robustness behavior facing failed information.
- Send information from board 1 simulating telecommand sent by OBC to a payload.
- Check that the telecommand received by the payload is correct.

- Send information from board 1, read this information by board 3, injects fault and relay that information to a payload.
- Check whether the payload behaviors conform expected in such a foreseen condition.

Scenario D:

- Set up an Arduino board (board 3) in order to read the information that is traveling in the I2C bus connected to the satellite engineering model configuring real OBC and payload hardware in the loop, respectively, as master and slave.
- Send remote information through the RF Checkout.
- Check the information being received by the OBC and payloads through the board 3.
- Selects the information (telecommand) sent from OBC to payload, injects fault through the board 3 and forward this information changed to the payload address. Analyzes the payload robustness behavior facing such failed information.
- Selects the information (telemetry) sent from payload to OBC, injects fault through the board 3 and forward this changed information to OBC. Analyzes the OBC robustness behavior facing such failed information.

4. TEST SYSTEM PROTOTYPE

A Test System prototype was implemented to demonstrate the feasibility of the four scenarios described above.

The resources used in the prototype were:

- Three boards Arduino UNO to simulate both the OBC and a payload, and also perform simple fault injection mechanism.
- One notebook with Windows 7 operating system to perform the functions of Arduino boards as designed.

- Computing applications supplied by the IDE Arduino that emulate the behavior of the elements under test according to the operational requirements of the payloads and OBC.
- One CI 74HC595, which has the shift register function. This component allows one to both read all the bits that are traveling in the I2C bus and change any bit injecting a purposeful fault.

4.1. Results

Scenarios A to D were successfully performed. The Arduino boards got communication with the I2C bus NanosatC-Br enabling reading, analysis and fault injection. There was no need for major changes to the existing architecture and this simple implementation allows it to be reused in various phases of testing.

In order to capture the information in the I2C bus and get a graphical visualization of this information a logic analyzer LogicSniffer was used.

This logic analyzer captures information in I2C bus. Once started Capture command the analyzer will identify the SDA (data) and SCL (clock) signals of the I2C protocol [8]. The following information is captured: addressing message, signal of read or write, the message packet and the end of each package. In the figure above the beginning of the message is marked, as well as the address sending, a message packet identifying the begin and end, other information such as time of dispatch of the package and the package in binary, decimal and ASCII.

Figure 4 shows the analysis results capturing one telecommand on the I2C bus.

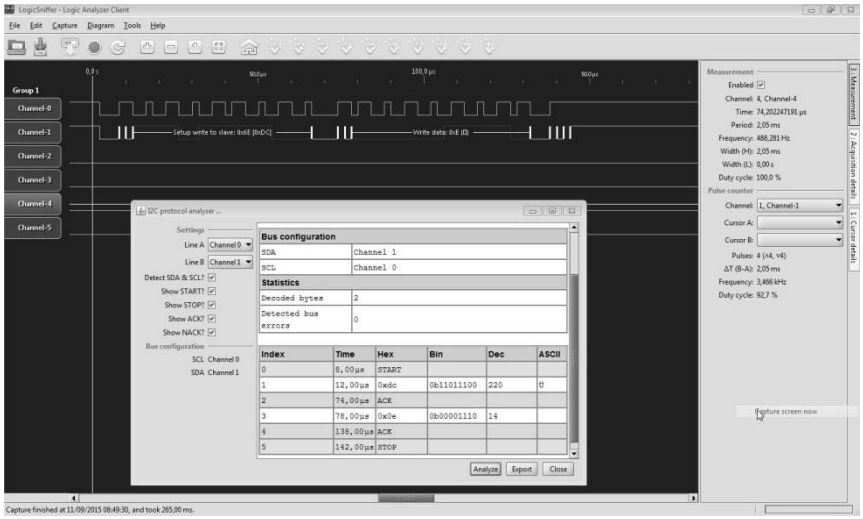


Fig. 4. Reading a telecommand in the I2C bus through the Logic Analyzer.

5. CONCLUSIONS

The Test System architecture and the scenarios proposed in this paper suggest that the use of Arduino boards combined with the Cubesat standard engineering model available in the market will be able to support the execution of nanosatellite functional tests as part of the V&V activities. Focusing on I2C communication bus as backbone for the interactions between payload elements and OBC used in Cubesat standard, the proposed Test System architecture allows evaluating interoperability and robustness of the elements that interact in the channel in nominal and known exception condition.

From the perspective of (i) reuse the Test System at different stages of the same mission, one can say that the Test System supports early verification of the operational demands over the communication channel required by the nanosatellite payloads in interaction with the OBC, during the mission conception phase. In addition, during the subsystems integration phase, the Test System supports the acceptance testing of each subsystem either in isolation or integrated, incrementally validating the system interoperability behavior with hardware in the loop. From the perspective of (ii) reuse the Test System in different satellites of the same family, the experiments show that the resources used to implement the Test System prototype are COTS, cost-effective and flexible enough to support testing many payloads in nanosatellite mission configurations that adopt the standard CubeSat.

Model-based Testing approaches for automatic test case generation aiming interoperability and robustness testing [10] will be aggregated to the Test System in the near future in order to systematize the test cases suite related to each scenario and the use of fault injection engines on the I2C bus.

REFERENCES

[1] Drusinsky, D., Michel J., B., et.al.: A Visual Tradeoff Space for Formal Verifica-tion and Validation Techniques, 2008.

[2] HOFFMANN, C. T.: Proposta de arquitetura para executar testes em módulos elétricos de cubesat, Dissertação (Curso de Pós-graduação em Ciências e Tec-nologias Espaciais. Área de Sistemas Espaciais, Ensaios e Lançamentos) - Ins-tituto Tecnológico de Aeronáutica, São José dos Campos, 2015.

[3] Innovative Solution In Space BV: IGIS User Manual, 2011.

[4] Innovative Solution In Space BV: ISIS: NANOSATC-BR System User Manual,2011.

[5] Innovative Solution In Space BV: ISIS: NCBR1 - Payload User Manual, ISIS.NCBR1-PL.UM, 2013.

[6] Instituto Nacional de Pesquisas Espaciais: 2014, http://www.inpe.br/crs/nanosat/NanoSatCBR1.php.

[7] McRoberts, M.: Arduino basico, Novatec, 2011, https://novatec.com.br/livros/arduino/capitulo9788575222744.pdf.

[8] Philips Semiconductors: AN10216-01, I2C MANUAL, 2003, http://www.nxp.com/documents/application_note/AN10216.pdf.

[9] Sistemas Espaciais e Tecnologia: Documento de Controle de Interfaces do Com-putador de Bordo do satélite NanosatC-BR2, 2015.

[10] Weller, A., C., Martins, E., Mattiello-Francisco, F.: InRob-UML: uma Abordagem para Testes de Interoperabilidade e Robustez baseados em Modelos, 2015.

ANEXO B – ARTIGO PUBLICADO: WDES-LADC, WORKSHOP ON DEPENDABILITY IN EVOLVING SYSTEMS, CALI, COLOMBIA, BRAZIL, 19 – 21 OCTOBER 2016, "DEPENDABILITY VERIFICATION OF NANOSATELLITE EMBEDDED SOFTWARE SUPPORTED BY A REUSABLE TEST SYSTEM"

Carlos A P L Conceicao, Carlos L G Batista, Fátima Mattiello-Francisco

Abstract— The use of CubeSats has increased tremendously over the 15 years since the standard creation because the low cost and reduced project development cycle. However, one of the most concerns in reducing a project delivery time is the collateral effect in test process, resulting in failures in the mission operation. This paper proposes the combined use of the Model-Driven Engineering (MDE) and Model-Base Testing (MBT) approaches in the Validation and Verification (V&V) process of a nanosatellite mission, focusing on an evolved way to measure the dependability requirements of the interoperable on-board software. The proposal counts on a reusable Test System (TS) based on Arduinos that are integrated in the engineering model of the Cubesat architecture via I2C bus. From the behavioral models of both the on-board computer and the satellite payloads, source code can be generated in order to be embedded in the Arduinos, prototyping in the TS the expected behavior of the interactions between the specified subsystems. These models can also be useful to derive test suites following a MBT approach. Thus the TS can support the execution of different test cases at different stages of development of the software intensive subsystems. The proposed V&V process is discussed in the context of a particular nanosatellite named NanosatC-Br2 under development at INPE.

Keywords— dependability; CubeSat; interoperability; verification; failure

I. INTRODUCTION

The first proposed CubeSat satellite platform emerged around 15 years ago aiming allow students to design, develop, test and even place a spacecraft in orbit [5]. The U-Class satellites have been used in academic environments and emerging companies of the sector seeking to qualify innovative space technologies at reduced costs and time. However, because the short period of the project development cycle not all tests required are performed leading to the occurrence of many cases of mission failure. Testing is recognized as a fundamental activity for assuring quality of a system showing dependability evidences of a mission operation [4].

The dependability of a system seeks to prevent failures in the services provided and reduce their severity to an acceptable level covering the attributes of reliability, availability, maintainability, security, integrity and confidentiality [1]. Thus dependability requirements must be specified and tested, following a well established verification and validation (V&V) process.

Model-Based Testing (MBT) is an approach that relies on the existence of abstract models of an application to generate, execute and evaluate tests [3]. It has been applied with success in testing of real-time systems, with special emphasis on avionic, railway, and automotive domains. The MBT approach has been effective in failures detection and performance analysis [2].

Recent experiments using MBT in the verification and validation (V&V) process of space system embedded software have demonstrated cost-effective on identifying system faults and project artifacts non-conformances [10]. Focusing on the integration phase of communicating software intensive subsystems, the methodology Interoperability and Robustness (InRob) [7] proposes the use of behavioral models of the interoperable subsystems in formal notation in order to generate test suites aiming to check whether two or more implementations interact as expected. InRob also addresses test case generation to check the robustness of the implementations under testing in the presence of failures. InRob methodology was recently adapted to Unified Modeling Language (UML) [12]. The InRob-UML methodology addresses real-time aspects of interoperability between two communicating subsystems and checks how these implementations inter-operate in the presence of invalid entries or under hazardous environmental conditions in the communication channel.

The Model-Driven Engineering approach (MDE) enforces the continued use of models in different stages of the project development cycle, in order to support the process of V&V of complex systems [8].

The process of V&V adopted in space mission project cycle has its importance in ensuring the quality of development of the space segment: the spacecraft and its integration with payloads and operation. Different V&V techniques are

used to verify compliance with the subsystems requirements at different phases of the project life cycle aiming improvement, safety and confidence in the mission operation [4].

The early stages of the development require a special attention to the requirements. Usually two thirds of the time in a space mission development is committed to an early attention to analysis, testing, review and validation. But just one third of the costs are being spent at this time [11]. If there are misunderstandings at the requirements definition and the necessary tests are not performing to validate the mission development cycle stages, it may result in a component failure, system failure, mission loss or even the loss of a life. Dependability requirements shall be specified and verified earlier in the development cycle.

The National Institute for Space Research (INPE) has been developing research in nanosatellites platforms looking for acquires knowledge and develops new technologies to meet national and international needs. Currently, INPE has in orbit the NanosatC-Br1, a 1U CubeSat. The mission goal is to collect data from the Earth's magnetic field and test integrated circuits designed in Brazil for resistance to radiation in flight.

The second satellite of this family, the NanosatC-Br2, is a 2U CubeSat, for science and technology purposes, which is in development and expected to be launched in 2017 [7].

II. OBJECTIVE

This paper proposes the combined use of the MDE and MBT approaches in the V&V process of NanosatC-Br2, by means a reusable Test System (TS) in the different development stages of communicating software intensive subsystems.

The reusable Test System (TS) is based on Arduinos that are integrated in the NanosarC-Br2 backbone (I2C bus). The TS supports the traffic analysis of the message through the satellite I2C bus.

In the early stage of the development cycle, the TS will support the verification of dependability requirements of the communicating embedded software. From

functional requirements specifications of the on-board computer subsystem and the satellite payloads subsystems, behavioral models are built in.

From these models, source code can be generated in order to embed them in the Arduinos, prototyping in the TS the expected behavior of the interactions between the specified subsystems.

The objective is to verify and validate the behavior of these satellite subsystem prototypes in nominal situation and exception conditions under the communication channel point of view.

These behavioral models can also be useful to derive test suites following MBT approach like InRob, which will be necessary to validate the subsystems communication in the integration phase considering hardware in the loop.

Thus the TS can also support the execution of different suites of test cases at different stages of development of the embedded software on-board computer subsystem.

III. PROPOSED TEST SYSTEM

The proposed TS uses a simple and low-cost architecture. The basic resource is COTS Arduino boards that can be added in TS architecture as necessary. In addition, there are several computer applications already developed for Arduino architecture and available at open source communities that will be useful for test execution and measurements on the tests.

Figure 1 shows the TS architecture as an extension of the CubeSat I2C bus.

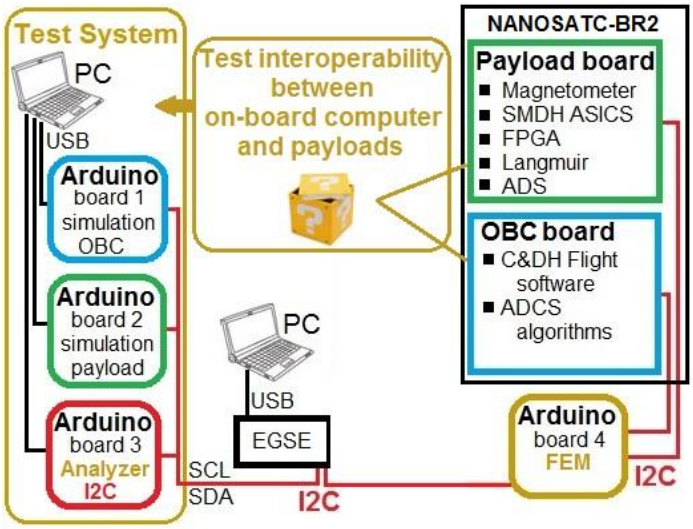


Figure 1. TS Architecture

The Arduino boards 1 and 2 are configured to simulate the behavior of both the on-board computer (OBC) and payload subsystems respectively. The board 3 is configured with a logic analyzer to analyze the traffic on the I2C bus (Figure 3).

The EGSE (Electrical Ground Support Equipment), usually provided by Cubesat suppliers as part of the nanosatellite engineering models, composes the TS architecture. It allows to boot satellite components and to control testing on the OBC and payloads.

Other important component of the TS architecture is the fault injection facility. Configured as a Fault Emulator Mechanism (FEM) in the satellite communication bus, it is also implemented in an Arduino (board 4 in Figure 1) in order to support robustness testing of the subsystems. The fault injection is an approach which consists of deliberately inserting faults into the system in such way that the behavior of the system can be evaluated under faults condition [12].

Thus the TS architecture supports the analysis of the communication between the satellite subsystems through the I2C bus. It allows to configure TS

components according to the evolution of the testing phases without major changes and/or costs.

IV. SCENARIOS FOR TS REUSING

Four scenarios were conceived to demonstrate the reuse of the TS in an evolved way driven by models. They focus respectively on the V&V activities related to: A) dependability requirements specification; B) subsystem acceptance; C) subsystem integration under nominal condition; D) subsystem integration under exception conditions.

Each scenario is represented in Figure 2 and described in terms of the TS architectural elements, which configuration can also include EGSE and the satellite engineering model in addition to Arduinos boards. The engineering model comprises CubeSat hardware and real software embedded in OBC and payload boards.

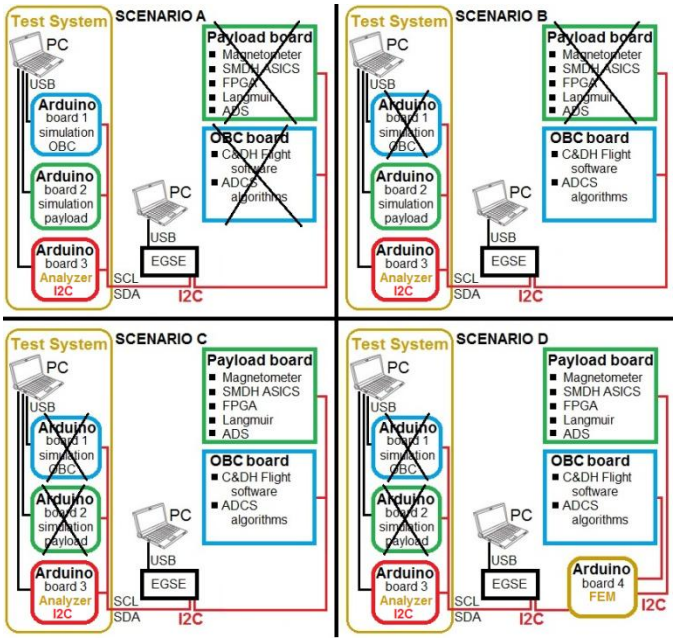


Figure 2. Test scenarios based on the I2C bus

Scenario A: The Arduino boards 1 and 2 are configured to simulate by software the subsystems OBC and a chosen payload respectively. From the subsystems requirements formally represented in state based behavioral models, specific tools can generate source code in order to embed in the Arduinos, simulating

the subsystem behavior. Using MBT approach, test suites can be automatically derived from those models with test purpose techniques. Because these Arduinos are interconnected through the satellite I2C bus and the board 3 analyzes the traffic, the test cases execution allows evaluating the subsystems requirements. Thus, regarding interoperability, whether a subsystem does not interact as expected, showing any inconsistency in the test report, improvements on the subsystem requirement specification will proceed.

Scenario B: Only one Arduino board is configured to simulate a subsystem by software, as in Scenario A. Other Arduino is substituted by the real subsystem, configuring the hardware in the loop (HIL) and the use of satellite I2C bus. This scenario purpose is to support HIL acceptance testing, aiming to show evidence of HIL requirements compliance. An important issue in this scenario is the reuse of the test suites already specified in Scenario A, reducing cost and effort in acceptance phase.

Scenario C: Both OBC and payload are real subsystems at the satellite I2C bus. Arduino board (board 3) monitors the traffic through the bus checking the data packets and transmission time between the subsystems. The objective of this scenario is to check if these subsystems interact in conformance to the behavioral models designed and verified in scenario A under nominal condition. Regarding the test suites, interoperability test suites already specified in Scenario A are reused, reducing cost and effort in the integration phase;

Scenario D: Sets up both real OBC and one real payload using a FEM (board 4) that is placed in serial on the I2C. As the two subsystems are already tested under nominal condition at the integration phase, this scenario goal is to support robustness testing. The FEM has an important role in this scenario because the fault injections are not intrusive in the subsystems under testing. It supports the verification of the subsystems interactions under exception conditions that were specified. Regarding the test suites properly specified for robustness testing, InRob methodology can be used to guide automatic test case generation focusing on real time aspects of the communicating subsystems.

One can observe that the Scenario A enables prototyping two or more satellite subsystems using software implementations embedded in Arduino boards, allowing verify the interoperability requirements by simulation.

In order to support the traffic monitoring on the satellite I2C bus, the free logic analyzer (Logic Analyzer - LogicSniffer) was configured on the Arduino board (board 3) and plugged on the bus.

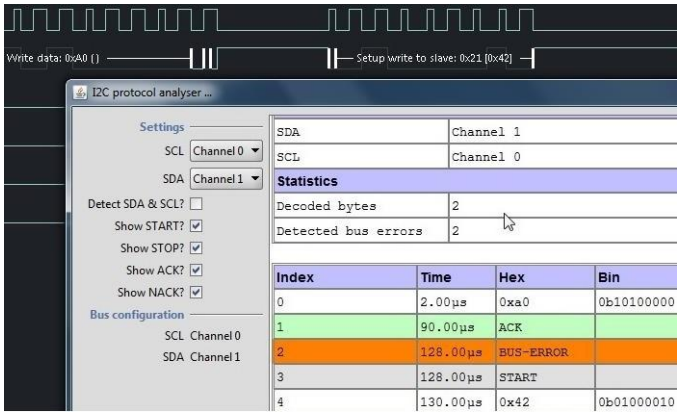


Figure 3. Reading the information on the I2C bus

The Figure 3 shows some information packets being read at the satellite I2C bus with a deliberated interference voltage, which causes errors in the package as highlighted in index 2 [9].

V. DENPENDABILITY ANALYSIS

Inspired on the dependability tree taxonomy proposed in [1], which addresses the attributes of dependability, fault categories and the concept of environments that might be affected by failures, a study was performed in order to identify dependability attributes related to nanosatellite.

The means to achieve dependability are: fault prevention, fault tolerance, fault removal and fault forecasting.

Regarding environment, the space segment is the target because the nanosatellite operation starts when it is accepted for use in orbit. The stakeholders' environments can consist of the following elements: the physical world, users, service providers, intruders may be human or system developer and infrastructure tools.

According to the taxonomy, eight fault classes are considered relevant. Each fault class is divided into two sub-classes as defined below and showed in Figure 4:

1. Phase of creation or occurrence: I) Development faults, II) Operational faults;
2. System boundaries: III) Internal faults, IV) External faults;
3. Phenomenological cause: V) Natural events faults , VI) Human-made faults;
4. Dimension: VII) Hardware faults, VIII) Software faults;
5. Objective: IX) Non-malicious faults, X) Malicious faults;
6. Intent: XI) Non-deliberate faults, XII) Deliberate faults;
7. Capability: XIII) Accidental faults, XIV) Incompetence faults;
8. Persistence: XV) Permanent faults, XVI) Transient faults.

These fault classes can be combined resulting various possibilities, however not all combinations are feasible neither might occur. An example is a natural event fault not related to incompetence faults as well as some other relationships.

As result, 31 feasible fault combinations represented in Figure 4 by Arabic numerals were grouped in the following 9 fault types with partial overlapping in some cases: A) Software Flaws, B) Logic Bombs, C) Hardware Errata, D) Production Defects, E) Physics Deterioration, F) Physical Interference, G) Intrusion Attempts, H) Viruses and Worms and I) Input Mistake.

One can observe that the dependability tree shown in Figure 4 does not contain all faults previously identified because the natural events fault will be not considered in dependability analysis in the nanosatellite context.

Thus the fault classes of interest are represented in three groups partially overlapped, referred as: Development Faults; Physical Faults; and Interaction Faults. The development faults group covers three fault types (A, C and D)

among the 9 types listed above. The physical faults group covers four fault types (C, D, F and G) and the interaction faults group covers two fault types (F and G).

A fault branch of the tree demonstrates that a particular fault type represented in the leave might be caused by one or more fault classes depending on the relationship with the previous or subsequent level.

The dependability tree supports the analysis of the dependability requirements modeled in Scenario A. The models show evidences of unexpected situations which may result in inconsistencies and failures.

For instance, in Scenario D, a fault injection in the I2C bus by FEM will emulate failure type F (Physical Interference).

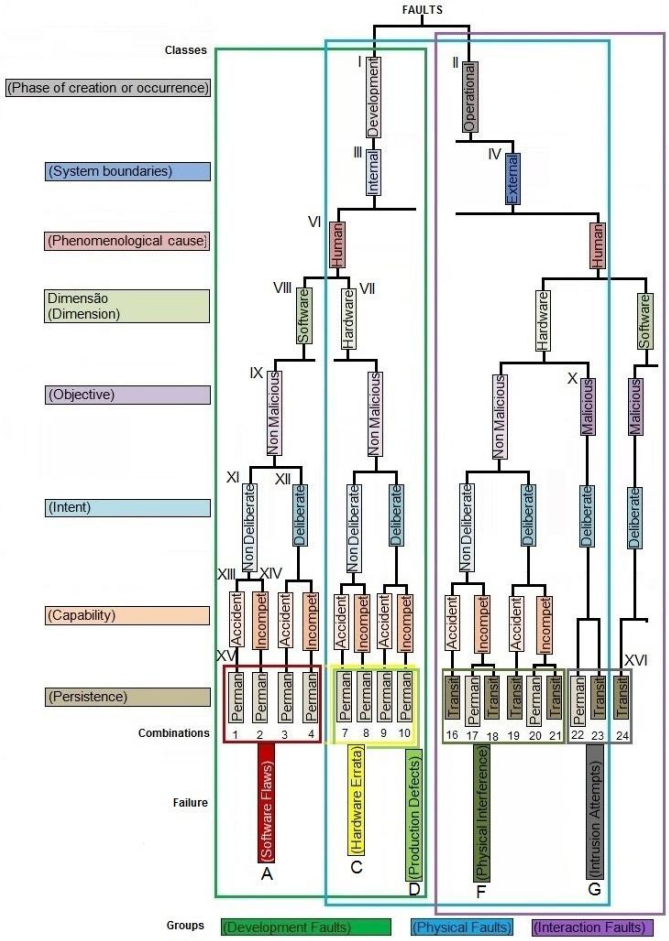


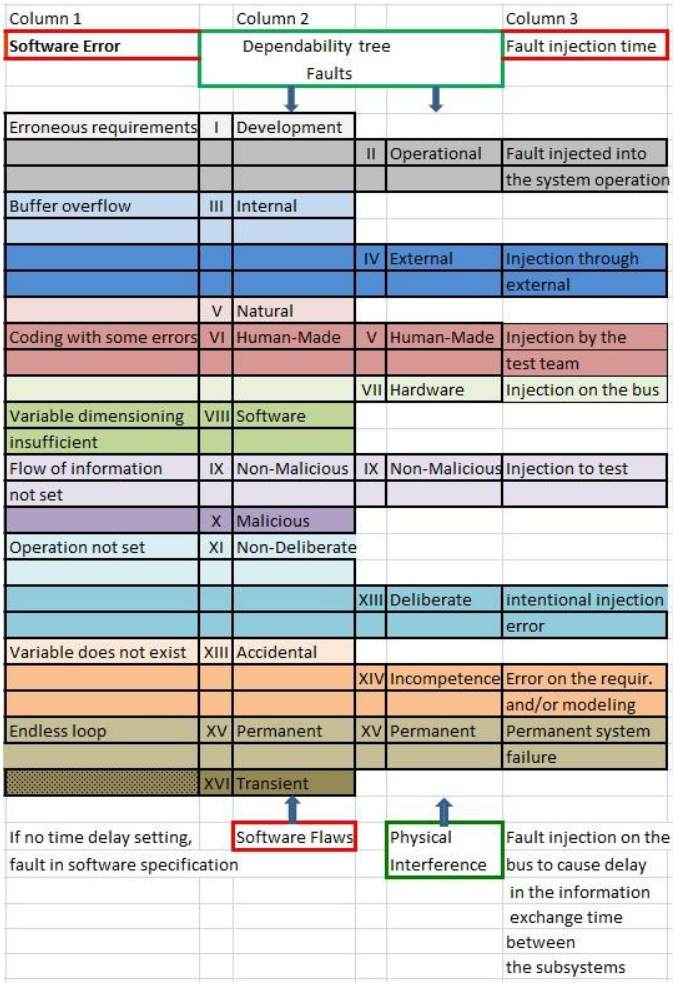
Figure 4. Dependability tree (modified) [1]

After FEM injecting a fault, some other fault may occur. If the purpose of the injected fault is to provoke a delay in sending information from subsystem A to B and there is no requirement to B deals with this situation in the modelling, such evidence will occur. The system as a whole can get in failure.

The cause-effect analysis of an injected fault using the dependability tree is discussed below in Table I.

In a particular scenario that a time fault is injected causing a software error, depending on which level the injection takes place in the tree, different possible causes shall be analyzed. The column 3 show the level that the fault is injected; the column 2 shows the fault class and sub class related to that fault according to the dependability tree; and the column 1 shows that the type of error caused by the injected fault shall be analyzed in terms of particular aspects. For instance, software error caused by fault injected at operational level means that functional implementation shall be checked in terms of requirement specification.

TABLE I. FAULT INJECTION



Column 1		Column 2			Column 3
Software Error		Dependability tree Faults			Fault injection time
Erroneous requirements	I	Development			
			II	Operational	Fault injected into the system operation
Buffer overflow	III	Internal			
			IV	External	Injection through external
	V	Natural			
Coding with some errors	VI	Human-Made	V	Human-Made	Injection by the test team
			VII	Hardware	Injection on the bus
Variable dimensioning insufficient	VIII	Software			
Flow of information not set	IX	Non-Malicious	IX	Non-Malicious	Injection to test
	X	Malicious			
Operation not set	XI	Non-Deliberate			
			XIII	Deliberate	intentional injection error
Variable does not exist	XIII	Accidental			
			XIV	Incompetence	Error on the requir. and/or modeling
Endless loop	XV	Permanent	XV	Permanent	Permanent system failure
	XVI	Transient			
If no time delay setting, fault in software specification		Software Flaws		Physical Interference	Fault injection on the bus to cause delay in the information exchange time between the subsystems

VI. TS IN NANOSATC-BR2 PROJECT

NanosatC-Br2 is a 2U-class nanosatellite being developed by CRS/INPE in cooperation with Brazilian Federal Universities and emergent companies. The mission goal is to qualify in orbit innovative technologies in addition to the scientific purposed experiments that collect data from the Earth's magnetic field and ionosphere. As mentioned in figures 1 and 2, there are 5 payload experiments on board NanosatC-Br2 named: Magnetometer; PFGA; Langmuir Probe (SLP); SMDH and ADS [6]. In addition, the On Board Computer (OBC) software comprises 2 components: Control and Data Handling (C&DH) and Attitude and Determination Control System (ADCS) algorithm that are technological targets for qualification in orbit.

This section discusses TS reuse in the context of NanosatC-Br2 V&V process, following the 4 scenarios described in section IV. The discussion addresses the combined use of MDE and MBT approaches considering real subsystems requirements.

For the sake of space, the requirements are related to two subsystems interoperation only: SLP payload subsystem and OBC subsystem. The communication channel is the I2C bus and follows the standard protocol master-slave, being both OBC the master and payload boards slaves.

Focusing on the interoperability between C&DH function embedded in OBC and SPL, the basic requirements are:

(i) SLP shall wait for an OBC command to start the data collection;

(ii) SLP shall collect data during 5 seconds, storing in SLP local memory;

(iii) OBC shall command SLP to start data transference packets no early than

5 seconds after (ii);

(iv) SLP data transference to OBC shall be controlled by OBC, which sends a

sequence of N packet requests, where N is a configurable parameter

received from ground station;

(v) OBC shall wait a specified time T (timeout) for SLP response at each

command sent. In case the response does not come, OBC shall restart

the communication and last SLP buffer must be erased.

Regarding the first use of TS in scenario A, these NanosatC-Br2 requirements were written in natural language. Based on these requirements behavioral models were specified in formal notation in order to be evaluated. OBC and SLP functional behaviors were modeled using state based diagrams. The modeling supported the review of the interoperability requirements between these subsystems. It was possible to categorize and detail each subsystem

states, events for state transitions, actions and guard conditions. The EGSE allowed observing the information traffic through the I2C bus either among Arduino boards and/or satellite engineering models boards as well. Beacon packets were transmitted from the OBC embedded software, which was simulated in TS Arduino, to the NanosatC-Br2 engineering model in order to be delivered to ground station, following scenario B in Figure 2. Also a data packet was transmitted from OBC board as master and received by board 2, which simulates a slave subsystem of the satellite, according to scenario A.

The test result in scenario A showed evidences that crash or lock on the I2C bus might occur due to no stable voltage between master and slave elements. A transistor was used for controlling the flow of electricity, solving the problem.

From MDE perspective, it was possible to prototype the subsystems. Existing software tools aided the modeling process and supported the source code generation to be embedded in Arduino board. The execution of the codes using TS allows simulating the behavioral models in compliance with the specified requirements. UML tools can also be used in many cases. Software based tools like UPPALL and Eclipse environment [8] help to generate computer codes based on state models.

From MBT perspective, available tools might aid to validate the models using model checking techniques, for instance. UPPALL tool supports the verification of requirements properties in the model. Test case suites can be automatically generated using InRob-UML.

Regarding the TS reuse in the 3 other proposed scenarios, the challenge lies in a suitable modelling transformation.

VII. MITIGATION OF FAILURE

Failure modes can be identified in each test scenario, following a FMEA table (Failure Mode and Effect Analysis). The technique consists on identifying possible causes for the detected failure, based on the fault effects in the dependability tree, which subsystems may be affected, what need to do to prevent failure and what mitigations should be done to prevent the failure mode.

Table II shows some mitigations associated with possible failures detected in the four proposed scenarios.

TABLE II. FMEA

Analyzed item	Fault type / identifying combination belonging	Class fault (nº)	Fault identfied	Fault effect (failure)	Place affected by fault	Control mechanisms for fault detection	Mitigation to avoid fault
Scenario A: OBC and SLP modeling	Software Flaws (A) / 1	Development Faults (I)	Erroneous Software Requirements	The subsystem does not properly perform what is specified in requirements	Data transfer	Data modeling	Review requirements and modeling using MDE and MBT
Scenario B: OBC and SLP (Only one of the subsystem in the loop. The other is Arduino board)	Software Flaws (A) / 1	Permanent Faults (XV)	No control in information transmission time	Subsystem still awaiting information	Exchange of information stop	Using RTC	Controlling exchange of information time
Scenario C : OBC and SLP	Software Flaws (A) / 1	Permanent Faults (XV)	Software without detection go into an infinite loop	Subsystems stop to exchange information	Internal failures of communication between subsystems	Data flow	Analyze possible ways of information flow modeling
Scenario D: OBC and SLP	Physical Interference (F) / 19	External Faults (IV)	Interference voltage	Subsystem does not perform data transfer as requirements. Loss of information	Subsystems	Data modeling and EGSE	Analyze the modeling possible physical external interference and physical faults

For instance, in scenario A, an erroneous requirement can lead to a modeling fault (development fault class). The TS will be useful to show failure evidences during the prototype execution guided by suitable test suites that were generated using MBT tools. Thus, the mitigation comprises review the requirements, improving the models (last column in Table 1).

VIII. CONCLUSION

The proposed Test System architecture in combination with MDE and MBT approaches demonstrate to successfully address the dependability verification issues highlighted in the studies and scenarios presented in this work. Interoperability and robustness requirements can be verified along nanosatellites development in a cost-effective V&V process.

The reuse of the TS architecture driven to integration testing is the key element for time and effort reduction. At low cost the four proposed scenarios cover whole V&V process.

The Test System can be reused in different phases of a particular nanosatellite development cycle and also several nanosatellites of the same family as well. The modular architecture shows that the resources used to implement the TS prototype are COTS, effective and flexible enough to support testing many payloads in nanosatellite project based on CubeSat standard.

The selection of possible failures types and fault classes from the dependability taxonomy helps to establish a view of the interrelationship of failures modes and what consequences may trigger. This view is very useful for failure mitigation.

The combined use of MDE and MTB approaches promises to save effort in development and testing process. However the available tools to modelling, code generation and test suite derivation need improvements. Moreover, investments in model transformation are necessary.

REFERENCES

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl Landwehr, TECHNICAL RESEARCH REPORT, Basic Concepts and Taxonomy of Dependable and Secure Computing, 2004, pp. 1-8, Available in: <http://www.nasa.gov/pdf/636745main_day_3-algirdas_avizienis.pdf> Access: january 2016 .

[2] BINDER, R. V., LEGEARD, B., AND KRAMER, A. Model-based testing: Where does it stand?, 2015, pp. 52-55, Available in: <https://www.researchgate.net/publication/272414727_Model-Based_Testing_Where_Does_It_Stand>, Access: june 2016.

[3] EL-FAR, I. K., AND WHITTAKER, J. A. Model-based software testing. In Encyclopedia of Software Engineering, J. J. Marciniak, Ed., vol. 1. John Wiley & Sons, Inc., 2002, pp. 825–837, Available in: < http://www.crcnetbase.com/doi/abs/10.1081/E-ESE-120046903 >, Access: june 2016.

[4] European Cooperation for Space Standardization, Space product assurance, Software dependability and safety, ECSS-Q-HB-80-03A, 2012, pp. 10-34, Available in: < https://www.google.com.br/search?q=InRob%3A+Anapproachfortestinginteroperabilityandrobustnessofreal-timeembedded+software&ie=utf-8&oe=utf-8&client=firefox-b-ab&gfe_rd=cr&ei=wJXRV6_JCYmq8we40KLwBQ#q=Space+product+assurance%2C+Software+dependability+and+safety>, Access: february 2016.

[5] Helvajian, H.,Janson W., Small Satellites: Past, Present, and Future, 2008, pp.25-39, Available in: < http://www.aerospace.org/publications/aerospace-books/small-satellites-past-present-and-future/>, Access: dezember 2015.

[6] Innovative Solution In Space BV: IGIS User Manual, 2011, pp. 8-26.

[7] Mattiello-Franciscoa, F., Martins, E. Cavalli, A. R., Yano, E. T., InRob: An approach for testing interoperability and robustness of real-time embedded software, 2012, pp. 1-4, Available in: <http://www.sciencedirect.com/science/article/pii/S0164121211000550>, Access: november 2015.

[8] Montecchi, L., Model-Driven Engineering and its Application for Dependability Analysis, Introduction to MDE, Workshop, INPE, Brazil, 2016.

[9] Philips Semiconductors: AN10216-01, I2C MANUAL, 2003, pp. 4-25, Available in: <http://www.nxp.com/documents/application_note/AN10216.pdf>, Access: july 2015.

[10] Utting, M., Pretschner, A., Legeard, B., A TAXONOMY OF MODEL-BASED TESTING, 2006, pp. 2-5, Available in: < http://wwwbruegge.in.tum.de/lehrstuhl_1/files/teaching/ws0708/ManagementSoftwareTesting/uow-cs-wp-2006-04.pdf>, Access: dezember 2015.

[11] TEST IN SPACE, DESCRIPTION OF SERVICE MULTI-PAYLOAD SATELLITE PROGRAM, pp. 4-6, Available in:

<http://www.testinspace.com/pdf/TIS_Description_of_Service.pdf>, Access: march 2016.

[12] Weller, A., C., Martins, E., Mattiello-Francisco, F., InRob-UML: uma Abordagem para Testes de Interoperabilidade e Robustez baseados em Modelos, 2015, pp. 71-77, Available in: <http://plutao.sid.inpe.br/col/sid.inpe.br/plutao/2015/12.04.12.15/doc/1_weller.pdf?metadatarepository=&mirror=dpi.inpe.br/plutao@80/2008/08.19.15.01.21>, Access: february 2016.

ANEXO C – ARTIGO PUBLICADO: WETE, 9º WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, SÃO JOSÉ DOS CAMPOS, SP, BRASIL, 15 E 16 DE AGOSTO DE 2018, "MÉTODO PARA REMOÇÃO DE FALHAS PARA SISTEMAS INTENSIVOS EM SOFTWARE A BORDO DE NANOSATÉLITES"

CONCEICAO, C. [1], MONTECCHI, L. [2]. MATTIELLO-FRANCISCO, F.[3]

[1]Aluno de Doutorado do Curso de Engenharia e Gerenciamento de Sistemas Especiais

2Universidade Estadual de Campinas, Campinas, SP, Brasil

[3]Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brasil

conceicaocarlos1@gmail.com

Resumo. *O uso do padrão Cubesat no desenvolvimento de satélites permitiu reduzir o ciclo de projeto missões espaciais na categoria nanosatélites, viabilizando qualificar em voo novas tecnologias espaciais a baixo custo da plataforma e de lançamento. Entretanto, continua onerosa a realização de todas as atividades de verificação e validação (V&V), essenciais nas diferentes fases do ciclo de vida do projeto espacial. Aspectos funcionais e de dependabilidade na integração das cargas úteis com a plataforma do satélite precisam ser bem especificados e testados. Buscando antecipar a verificação de requisitos por meio de simulação e testes comportamentais do software embarcado no subsistema computador de bordo do satélite, na sua interação com os subsistemas cargas úteis da missão em situação nominal e de exceção, o presente artigo apresenta um método para remoção de falhas em sistemas intensivos em software. O método denominado STAE utiliza um Sistema de Teste com Arquitetura Escalável de baixo custo reutilizável em projetos de nanosatélites padrão Cubesat.*

Palavras-chave: Cubesat; Falhas; Interoperabilidade; Dependabilidade; Verificação.

1. Introdução

As primeiras propostas de satélites padrão Cubesats surgiram com o objetivo de permitir que estudantes desenvolvessem e testassem no espaço satélites de pequeno porte com capacidades similares ao primeiro satélite colocado em órbita [Helvaijan and Janson, 2008]. Entretanto, por serem desenvolvidos em curtos períodos e com baixo orçamento, nem todos os testes necessários para

sua qualificação são realizados, que levam a falha na missão [Swartwout, 2016].

No ciclo de projeto de missões espaciais, o processo de V&V adotado no desenvolvimento do segmento espacial possui papel chave na garantia da qualidade da missão. As diferentes técnicas de V&V existentes abrangem de maneiras complementares as atividades de verificação e validação dos requisitos dos produtos desenvolvidos nas diferentes fases do ciclo de vida de um projeto espacial, buscando melhoria, segurança e confiabilidade nas missões [Test in Space, 2007].

Com o avanço da tecnologia, cada vez mais funcionalidades de satélites são implementadas por software. Sistemas embarcados se tornam mais complexos e difíceis de serem testados.

A aplicação de abordagem Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE) em diferentes domínios vem vencendo o desafio do uso continuado de modelos nas diferentes fases do ciclo de desenvolvimentos de projetos, com o propósito de apoiar o processo de V&V de sistemas complexos [Montecchi, 2006].

Em complemento, a abordagem Teste Baseado em Modelo (*Model-Based Testing* - MBT) tem sido utilizada para sistematizar e automatizar os testes de sistemas, possibilitando avaliar o desempenho da implementação e antecipar a detecção e remoção de falhas no ciclo de desenvolvimento. Casos de testes são automaticamente gerados a partir de modelos comportamentais dos sistemas, com o uso de ferramentas e algoritmos que se encarregam de aplicar nos modelos os critérios de cobertura selecionados pelo testador. Abordagens MBT tem se mostrado eficazes na identificação de falhas de sistema, reduzindo esforços de testes [Petrenko and Schlingloff, 2013].

Este artigo apresenta um método, o qual utiliza um Sistema de Teste com Arquitetura Escalável (STAE) de baixo custo, reutilizável no ciclo de desenvolvimento do projeto, que permite antecipar a identificação e remoção de possíveis falhas de integração de subsistemas comunicantes intensivos em software, embarcados em missões de nanosatélites. Assim, o método auxilia o

processo de V&V de software embarcado em Cubesat. O STAE é fruto da combinação das abordagens MDE e MBT. A abordagem MDE apoia transformar de forma automatizada a modelagem comportamental dos requisitos em códigos computacionais executáveis. A abordagem MBT auxilia na geração automatizada de casos de teste a partir dos modelos comportamentais do sistema da modelagem dos subsistemas de um Cubesat.

Com o STAE é possível verificar requisitos de sistemas de software comunicantes a bordo de Cubesat por meio da simulação dos modelos de interoperabilidade na arquitetura de teste - *model in the loop* (MIL). Utilizando os conceitos de dependabilidade da Taxonomia de [Avizienis et al., 2004], aspectos não funcionais do sistema são analisados no modelo nominal de interoperabilidade com o propósito de enriquecer a especificação funcional com requisitos de robustez [Mattiello-Francisco, 2009]. Os modelos estendidos são utilizados para gerar casos de testes que são aplicados nos testes de integração da versão real dos sistemas comunicantes – *hardware in the loop* (HIL). Assim, o método possui o atributo de apoiar análise de falhas e realizar testes com uma arquitetura de teste reutilizável em vários cenários de integração, conforme a evolução do projeto do satélite, nas diferentes fases do desenvolvimento da missão e/ou de uma mesma família de satélites.

O artigo está estruturado como segue. Seção 2 apresenta o método STAE com os recursos utilizados para sua implementação. A seção 3 apresenta uma aplicação do método STAE no processo de V&V do software *On Board Computer* (OBC) do NanosatC-BR2, um Cubesat 2U em desenvolvimento no INPE/CRS [Schuch et al., 2017]. A seção 4 conclui o artigo apresentando as vantagens encontradas na implementação do STAE.

2. Método STAE

O método STAE é implementado de maneira incremental utilizando modelos comportamentais empregando abordagens MDE e MBT. Estas evoluções são implementadas no processo de V&V durante as fases de desenvolvimento compondo quatro cenários, os quais desempenham respectivamente as seguintes funções baseados nos requisitos de interação dos subsistemas sob

teste: a) verificar os requisitos de interoperabilidade de subsistemas intensivos em software que se comunica por meio de modelos de estado que representam o comportamento nominal dos subsistemas comunicantes com foco nas interações (model in the loop - MIL), verificar a robustez dos subsistemas comunicantes por meio de códigos simulados e uso do barramento real de comunicação (SIL) e validar os casos de testes, b) validar a implementação dos subsistemas reais no loop (HIL) separadamente, c) validar a interoperabilidade dos subsistemas reais no loop, e d) injetar falhas no canal de comunicação para testar a robustez entre os subsistemas reais comunicantes. O desenvolvimento do método com a evolução das modelagens de dois subsistemas (S1 e S2) é apresentado na Figura 1 a seguir:

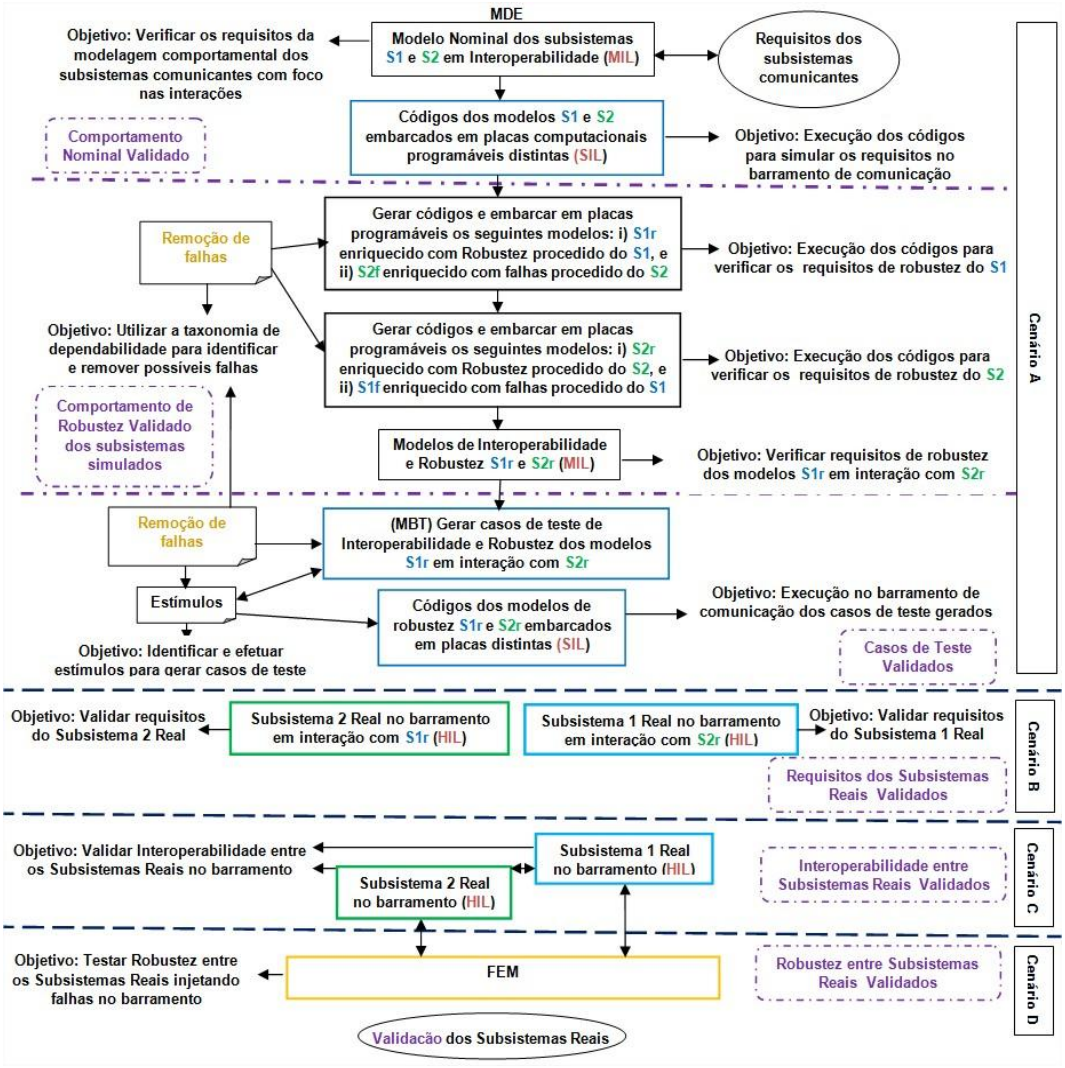


Figura 1. Evolução das modelagens.

Nos quatro cenários do método STAE são executadas as seguintes realizações durante a evolução das modelagens:

- Cenário A: As evoluções realizadas neste cenário têm por objetivo, verificar e validar as seguintes condições: i) comportamento nominal dos subsistemas comunicantes com foco nas interações, ii) comportamento de robustez dos subsistemas simulados, e iii) casos de teste:

  i. O objetivo inicial é verificar os requisitos de interoperabilidade especificados para a interação de dois subsistemas comunicantes, partindo-se da representação do comportamento nominal de cada subsistema em modelo de estado. Testes são realizados por meio da execução dos modelos destes subsistemas, adotando-se o conceito "*Model-In the Loop*" (MIL) utilizando ferramenta computacional baseada em conceito MBT. Uma vez validada a especificação, códigos computacionais de cada subsistema comunicante são automaticamente gerados, utilizando ferramenta baseada em conceito MDE, a partir dos modelos MIL validados. Estes códigos são embarcados em placas computacionais programáveis distintas no Sistema de Teste com o propósito de simular o comportamento de interoperabilidade especificado na presença do barramento de comunicação real, adotando-se o conceito de "*Simulation-In the Loop*" (SIL);

  ii. Na sequencia são implementados requisitos de robustez nos modelos dos subsistemas simulados. Para identificação dos requisitos de robustez a serem implementados é utilizada a taxonomia de dependabilidade, Inicialmente esta taxonomia auxilia na identificação de possíveis falhas na interação entre os subsistemas comunicantes. As possíveis falhas identificadas são relacionadas em uma Planilha de Dependabilidade e uma tabela FMEA, desenvolvidas para o método STAE, baseadas nos conceitos de dependabilidade. Nesta planilha e tabela são relacionados procedimentos com mitigações

para remoção de cada possível falha identificada. As falhas identificadas são implementadas em um dos modelos validado gerando um modelo enriquecido com falhas e as mitigações são implementadas no outro modelo gerando um modelo enriquecido com robustez. O objetivo é testar e verificar os requisitos dos modelos com robustez provocando falhas por meio dos modelos enriquecidos com falhas. A próxima operação é testar os modelos com robustez validados em interoperabilidade cujo objetivo é verificar os requisitos de robustez dos subsistemas simulados em interação (MIL). Estes testes permitem remover as possíveis falhas identificadas validando o comportamento de robustez dos subsistemas simulados;

iii. As falhas identificadas e relacionadas na Planilha de Dependabilidade agora são utilizadas como estímulos nos modelos com robustez validados buscando gerar casos de teste de forma automatizada utilizando uma ferramenta baseada em conceito MBT. Após os testes atenderem as especificações dos requisitos, são gerados de forma automatizada, por meio de ferramenta baseada em conceito MDE, os códigos destes modelos e embarcados em placas computacionais distintas. O objetivo é executar e validar os casos de teste no barramento de comunicação (SIL).

- Cenário B: Cada subsistema real do satélite é conectado separadamente ao barramento de comunicação com a placa contendo os códigos do subsistema simulado de robustez validado correspondente ao outro subsistema real que efetua interoperabilidade (HIL). O objetivo é verificar os requisitos de cada um dos subsistemas reais separadamente. Uma questão importante neste cenário é a reutilização dos conjuntos de testes já especificados no Cenário A, reduzindo o custo e o esforço na fase de aceitação;
- Cenário C: O objetivo deste cenário é verificar se os subsistemas reais interagem conforme os modelos comportamentais desenvolvidos e verificadas no Cenário A, sob condição nominal. Em relação ao

conjunto de teste, os casos de teste de interoperabilidade já especificados no Cenário A são reutilizados e testados, reduzindo o custo e esforço na fase de integração. Neste cenário os subsistemas reais são conectados em interoperabilidade no barramento de comunicação (HIL);

- Cenário D: Validados os subsistemas reais em interoperabilidade, este último cenário testa a robustez dos subsistemas reais. Nos cenários anteriores as falhas são injetadas por meio dos modelos simulados. Neste cenário, para testar a robustez dos subsistemas reais é utilizado um Mecanismo Emulador de Falhas (FEM) conectado entre os subsistemas reais no barramento de comunicação. O objetivo é injetar falhas no barramento e verificar a capacidade de cada um dos subsistemas de resistir a falhas externas.

No método STAE são utilizadas os seguintes recursos de desenvolvimento:

- A ferramenta de modelagem selecionada para a implementação do MBT foi a Uppaal. Esta ferramenta além de ser gratuita possui os seguintes recursos: i) utiliza o formalismo timed automata, ii) permite simular as interações entre os subsistemas comunicantes modelados, iii) gera diagrama de sequencia permitindo verificar a mudança dos estados durante a simulação, e iv) gera de forma automatizada casos de teste, entre outros;
- A ferramenta MDE de modelagem utilizada foi a Eclipse, a qual possui a ferramenta Yakindu possibilitando gerar códigos computacionais de forma automatizada para placas computacionais Arduino, entre outros recursos;
- As placas computacionais utilizadas são Arduino por serem de baixo custo e serem disponibilizadas na internet diversas bibliotecas gratuitas para estas placas facilitando a implementação de recursos como protocolo de comunicação, entre outros.

3. Aplicação do Método STAE

O Instituto Nacional de Pesquisas Espaciais (INPE) vem desenvolvendo pesquisas em plataformas de nanosatélites buscando adquirir conhecimento e desenvolver novas tecnologias para atender necessidades de âmbito nacional. Atualmente o INPE/CRS desenvolve um Cubesat 2U, denominado NanosatC-BR2 [Schuch et al., 2017], cuja missão é coletar dados de campo magnético da Terra e testar em voo a resistência de circuitos à radiação entre outras novas tecnologias projetadas no Brasil.

O método apresentado neste artigo foi proposto para apoiar a verificação e validação de requisitos de subsistemas comunicantes intensivos em software nas diferentes fases do ciclo de desenvolvimento de um nanosatélite. Para melhor explicar o método proposto, apresenta-se sua aplicação no desenvolvimento do NanosatC-BR2, nos testes dos subsistemas Computador de bordo (OBC) e uma carga útil (*Payload* - PL), denominada Sonda de Lagmuir (SLP). O protocolo de comunicação deste nanosatélite é o I2C. As seções seguintes estão estruturadas de modo a cobrir os cenários que compreendem o método STAE.

3.1 Cenário A

O método STAE é desenvolvido a partir dos requisitos dos subsistemas comunicantes. Os requisitos básicos especificados destes subsistemas para a realização da modelagem de comportamento nominal foram:

- O subsistema OBC assume a função de mestre e seu comportamento nominal na interação com a carga útil SLP deve ser representado pelos seguintes estados no modelo de estado: 1) estar em operação avaliando informações recebidas pela SLP (estado *IDLE*), 2) enviar comandos (*Command)* para a SLP (estado *Send*), 3) receber informações (*Information*) da SLP para definir a próxima operação (estado *Receive*), e 4) gravar na área de memória os dados lidos do experimento da SLP (estado *Write*);

- O subsistema carga útil SLP assume o papel de escravo e seu comportamento nominal na interação com o subsistema OBC deve ser representado pelos seguintes estados no modelo de estado: 1) estar em operação (estado *ON*) aguardando início da comunicação com o OBC, 2) receber comandos (*Command*) do OBC (estado *Receive*) para avaliar a próxima operação a ser realizada, 3) enviar informações (*Information*) solicitadas pelo OBC (estado *Send*), 4) realizar aquisição dos dados do experimento e armazenar no buffer (estado *Acquisition_Experiment*), e 5) ler os dados do experimento (estado *Read_Experiment*) e enviar para o OBC (estado *Send*).

Outras informações identificadas e selecionadas para este artigo foram às condições excepcionais entre os subsistemas. As condições selecionadas foram: 1) no caso do OBC receber uma informação não reconhecida deve informar que houve problema (*Problem*) e deve reenviar o primeiro comando, 2) no caso da SLP receber um comando não reconhecido deve informar que houve problema (*Problem*) e deve ler novamente o primeiro comando reenviado pelo OBC, 3) no caso da SLP reconhecer o comando recebido deve informar que está certo (*OK*), 4) quando a SLP terminar a aquisição do experimento deve informar que está pronta (*Ready*) para ler novos comandos e deve enviar os dados para o OBC, e 5) quando a SLP terminar de enviar os dados para o OBC deve informar que está tudo certo (*OK*) e deve retornar a aquisição de novos dados do experimento.

Para a realização da modelagem dos subsistemas na Uppaal foram definidas algumas variáveis indicadoras de ações dos subsistemas, de armazenamento e transferência de dados: As variáveis definidas foram: i) *WriteOBC* (gravar dados do experimento), ii) *OBCData* (receber dados do experimento), iii) SendDataOBC (enviar dados para OBC), iv) *PLBuffer* (armazenar dados do experimento), v) *SendOBC* (enviar informações para SLP), e vi) *SendPL* (enviar informações para OBC).

A ferramenta de modelagem Uppaal utilizada no método STAE realiza a sincronização entre as máquinas de estado utilizando os seguintes indicadores:

i) identificador de canal seguido do caractere exclamação (!) é definido como output para escrever informação no canal de uma máquina de estado para outra, ii) identificador de canal seguido do caractere interrogação (?) é definido como input para ler informação no canal. Duplas escrita e leitura no mesmo canal geram a sincronização. Na modelagem o sincronismo foi realizado por meio dos identificadores *SendfromOBC!* para o OBC enviar comandos à SLP e *SendfromPL?* para receber informações. A Figura 2 a seguir apresenta as modelagens dos subsistemas OBC e SLP em interação desenvolvidas na UPPAAL:

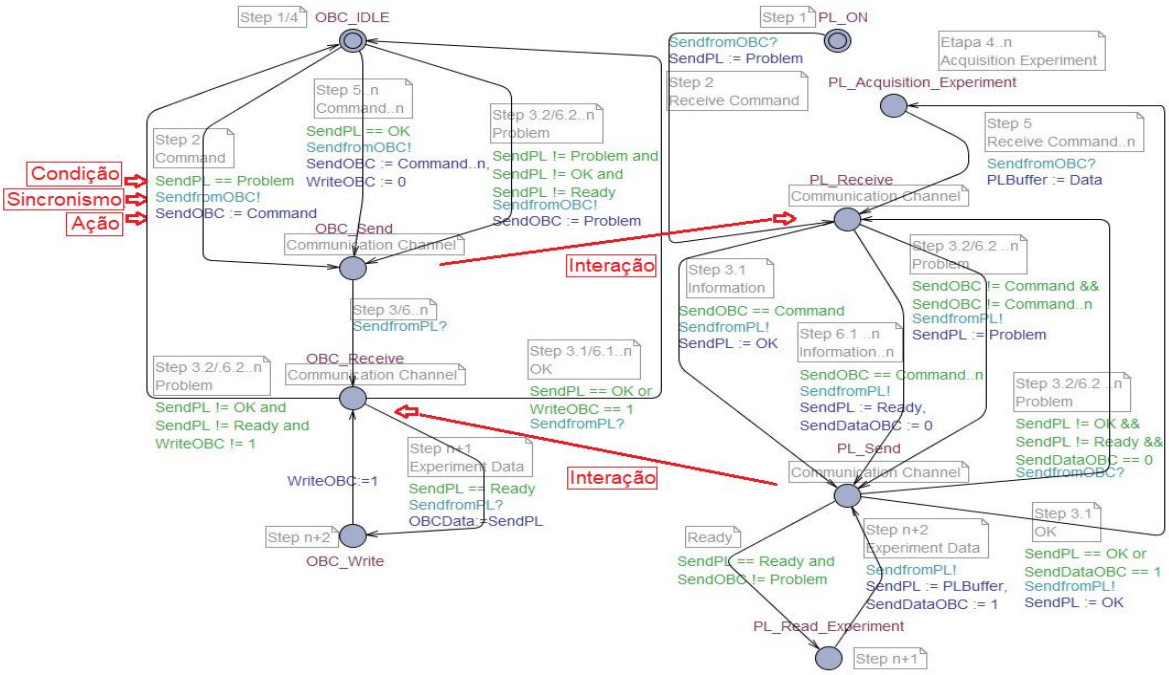


Figura 2. Modelagens dos subsistemas OBC e SLP em interação.

Os modelos dos subsistemas em interação validados por simulação (MIL) na Uppaal foram implementados na ferramenta Yakindu, a qual possibilitou gerar os códigos computacionais de cada um dos subsistemas modelados. Os códigos foram implementados e executados em placas Arduino distintas para cada subsistema interconectados no barramento de comunicação. O objetivo foi verificar os requisitos do comportamento nominal destes modelos em interação em ambiente simulado no barramento real (SIL).

Algumas possíveis falhas foram identificadas na interação entre os subsistemas comunicantes. Algumas destas falhas foram: i) tempo de taxa de transmissão ocasionar problema de comunicação, ii) informações não especificadas nos requisitos trafegar no barramento, entre outras. As possíveis falhas identificadas foram catalogadas na Planilha de Dependabilidade e na tabela FMEA relacionando as mitigações necessárias para remoção destas falhas. Estas falhas enriqueceram os modelos nominais com falhas e as mitigações com robustez respectivamente gerando modelos mutantes. Os modelos mutantes enriquecidos com falhas foram utilizados para testar a robustez dos modelos mutantes enriquecidos com robustez. O objetivo foi remover as falhas identificadas e validar os modelos enriquecidos com robustez.

Os modelos validados com robustez foram implementados na Uppaal com o objetivo de submeter novos testes com os estímulos das falhas identificadas e gerar os casos de testes. Com as validações dos casos de teste, estes modelos foram implementados na Yakindu com o propósito de gerar os códigos computacionais e embarcar em placas computacionais distintas para cada modelo. Os estímulos foram implementados nos códigos com o objetivo de testar e validar os casos de teste destes modelos enriquecidos com robustez em interação no barramento de comunicação (SIL).

3.2 Cenário B

Para a realização deste cenário o subsistema real OBC foi conectado ao barramento de comunicação com a placa contendo os códigos do subsistema simulado SLP validado com robustez. O objetivo foi verificar se os requisitos do subsistema real OBC em interação com os requisitos validados da SLP atendiam as especificações dos requisitos e não apresentavam falhas de interoperabilidade no barramento (SIL).

A carga útil SLP real não foi testada no barramento por esta em desenvolvimento. Desta forma, também não foi realizado o teste no Cenário C.

3.3 Cenário D

Neste cenário foi realizado o teste do injetor de falha FEM. O FEM foi conectado entre o subsistema real OBC e o subsistema simulado SLP no barramento de comunicação. Um dos testes realizados no FEM foi injetar uma falha na informação enviada pelo OBC de forma que a informação recebida pela SLP não estivesse na especificação dos requisitos. O objetivo foi verificar o comportamento de robustez da SLP. Esta possível falha estava relacionada na Planilha e Dependabilidade e na tabela FMEA e a robustez necessária já havia sido implementada no modelo da SLP simulada, validando o método para remoção de falhas.

4. Conclusão

O método STAE apresentado proporciona identificação e remoção de possíveis falhas na missão espacial, por meio da utilização de conceitos de dependabilidade, interoperabilidade e robustez conjuntamente.

A utilização das abordagens MBT e MDE permitem simulações do comportamento dos subsistemas antes destes subsistemas serem testados em fase de integração.

O STAE permite a sua reutilização em diferentes fases de uma mesma missão fazendo implementações nas fases de teste de validação, adicionando ou removendo os subsistemas do satélite no barramento reduzindo o custo e o esforço. Existe a possibilidade do mesmo sistema de teste ser usado em diferentes satélites da mesma família.

Referências

Avizienis, A., Laprie, J. C., Randell, B., Landwehr, C. (2004) "TECHNICAL RESEARCH REPORT, Basic Concepts and Taxonomy of Dependable and Secure Computing".

Helvajian, H. and Janson, S. W. (2008) "Small Satellites: Past, Present and Future".

Mattiello-Francisco, M. F., InRob (2009) "Uma abordagem para testes de interoperabilidade e de robustez de subsistemas de tempo-real intensivos em software".

Montecchi, L. (2006) "Model-Driven Engineering and its Application for Dependability Analysis, Introduction to MDE".

Petrenko, A., K. and Schlingloff, H. (2013) "Eighth Workshop on Model-Based Testing".

Schuch, N., Durão, O., Silva, M., Mattiello-Francisco, F., Silva, A., (2017) "NANOSATC-BRSTATUS-A joint Cubesat-based program developed by INPE and UFSM".

Swartwout, M. (2016) "Secondary spacecraft in 2016: Why some succeed (and too many do not).

TEST IN SPACE (2007) "Description of service multi-payload Satellite program".

ANEXO D – MANUAL PARA INSTALAÇÃO, CONFIGURAÇÃO E EXECUÇÃO DA ABORDAGEM SISTEMÁTICA DE TESTES DE SOFTWARE COMUNICANTES EMBARCADOS EM NANOSATÉLITES COM FOCO EM FALHAS DE INTEROPERABILIDADE

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

ABORDAGEM SISTEMÁTICA DE TESTES DE SOFTWARE COMUNICANTE EMBARCADO EM NANOSATÉLITES COM FOCO EM FALHAS DE INTEROPERABILIDADE

Carlos Augusto Paiva Lameirinhas da Conceição

Manual para instalação, configuração e execução dos recursos computacionais utilizados nesta Abordagem apresentada na Tese de Doutorado de Conceição, C. A. P. L.

INPE
São José dos Campos
2019

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE SIGLAS E ABREVIATURAS

ACK	Reconhecido (*Acknowledge*)
COTS	Componentes Comerciais (*Commercial Off-The-Shelf*)
DAS	Sistema de Determinação de Atitude (*Determination Attitude System*)
EGSE	Equipamentos de Suporte em Solo para Teste Elétrico (*Electrical Ground Support Equipment*)
FEM	Mecanismo Emulador de Falha (*Fault Emulator Mechanism*)
FME	Análise dos Modos de Falha e seus Efeitos (*Failure Mode and Effect Analysis*)
FPGA	Matriz com portas configuráveis (*Field Programmable Gate Array*)
HIL	Hardware no loop (*Hardware-in-the-loop*)
INPE	Instituto Nacional de Pesquisas Espaciais
I2C	Circuito Inter-Integrado (*Inter-Integrated Circuit*)
MBT	Teste Baseado em Modelo (*Model-Based Testing*)
MDE	Engenharia Orientada a Modelos (*Model-Driven Engineering*)
MIL	Modelo no loop (*Model-in-the-loop*)
NACK	Não reconhecido (*No Acknowledge*)
OBC	Computador de Bordo (*On-Board Computer*)
PC	Computador Pessoal (*Personal Computer*)
PL	Carga útil (*Payload*)
SCL	Sinal de Relógio (*Signal Clock*)
SDA	Sinal de Dados (*Signal Data*)
SIL	Software no loop (*Software-in-the-loop*)
STAE	Sistema de Teste com Arquitetura Escalável

TRXUV	Transmissão e Recepção de Dados (*Data Transmission and Reception*)
V&V	Verificação e Validação

SUMÁRIO

Pág.

1 INTRODUÇÃO

A abordagem desenvolvida na Tese de Doutorado de Conceição, 2019, tem como objetivo sistematizar testes de software comunicantes embarcados em nanosatélite com foco em falhas de interoperabilidade apoiando o processo de verificação e validação (V&V) no início do ciclo de desenvolvimento em missões espaciais de nanosatélites. A abordagem desenvolvida utiliza um Sistema de Teste com Arquitetura Escalável (STAE), o qual permite a sua reutilização de forma sistemática e evolutiva em diferentes fases de uma mesma missão ou em satélites da mesma família. A abordagem utiliza a combinação das técnicas MBT e MDE e conceitos de dependabilidade dirigido à interoperabilidade entre esses subsistemas.

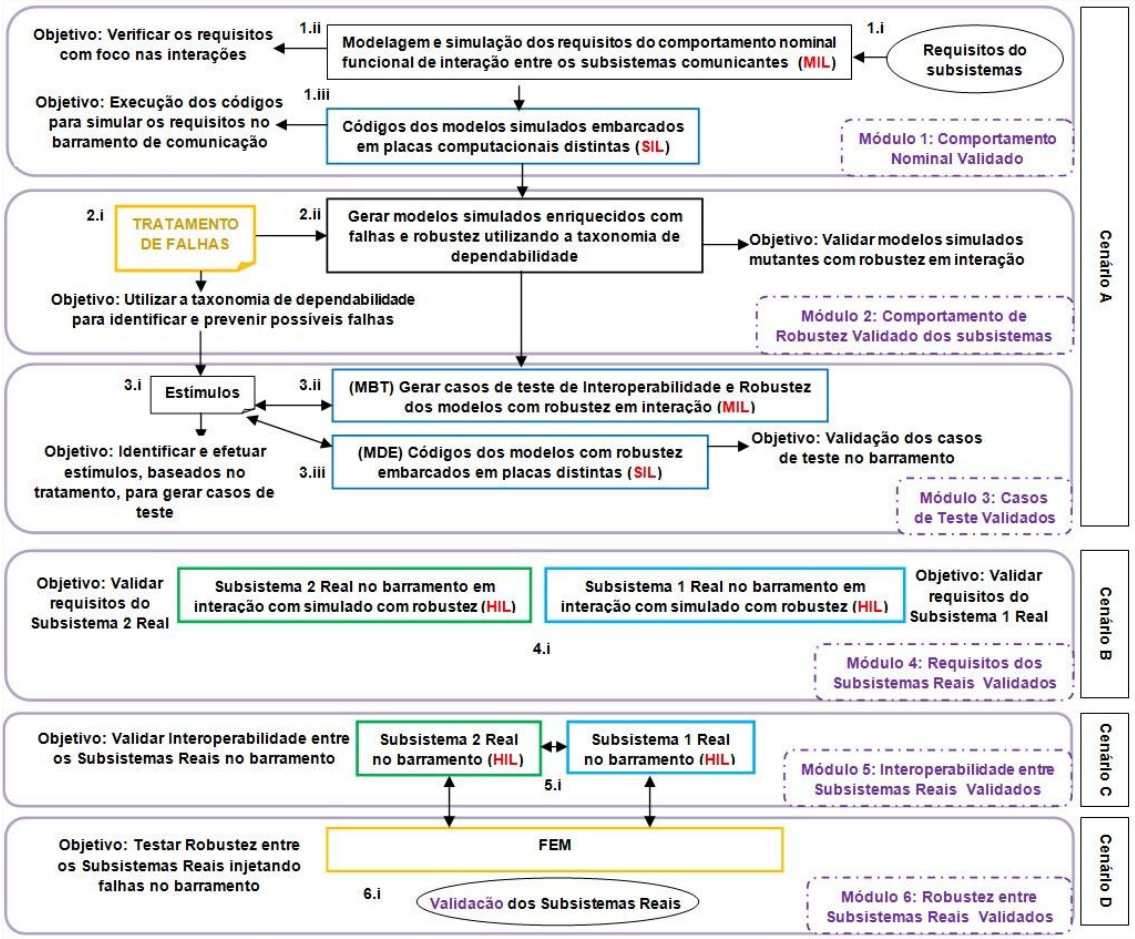
2 OBJETIVO

Este manual tem como objetivo apresentar inicialmente de forma sucinta os procedimentos para sistematizar testes de software. Posteriormente são apresentados os recursos computacionais utilizados e como realizar as instalações, configurações e execuções desses recursos. Esses recursos foram utilizados no estudo de caso do nanosatélite NanosatC-BR2.

3 EVOLUÇÃO DA SISTEMÁTICA DE TESTES

As evoluções da sistemática de testes são divididas em seis principais módulos. Esses módulos são realizados para atenderem os seguintes objetivos: 1) validar os requisitos do comportamento nominal funcional dos subsistemas comunicantes modelados e simulados relativos às interações, 2) validar os subsistemas simulados implementados com robustez baseados no tratamento de falhas, 3) validar os casos de testes definidos no tratamento de falhas e testados nos subsistemas com robustez simulados validados, 4) validar os requisitos dos subsistemas reais separadamente, 5) validar a interoperabilidade dos subsistemas reais no barramento, e 6) validar a robustez dos subsistemas reais injetando falhas no barramento. A Figura 3.1 apresenta o fluxo de procedimentos para a realização da abordagem para tratamento de falhas.

Figura 3.1 - Evolução da sistemática de testes.



Pode ser observado que a sistematização evolui utilizando os conceitos MBT e MDE conjuntamente abordando os níveis MIL, SIL e HIL.

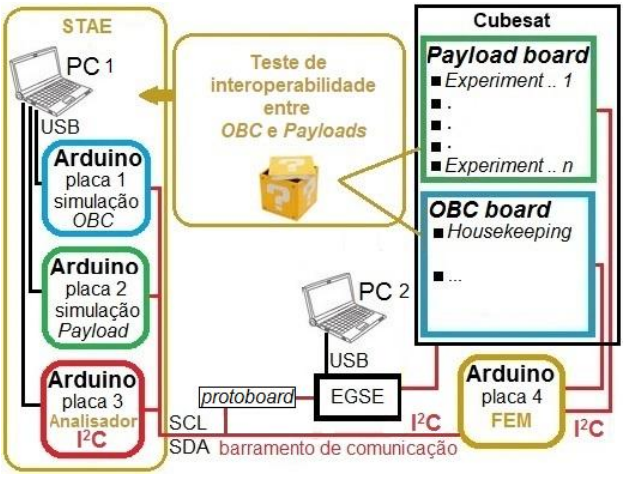
3.1 Arquitetura do STAE

A arquitetura do STAE é composta de: i) uma placa Arduino para emular o computador de bordo, ii) uma placa Arduino para emular as cargas úteis, iii) uma placa Arduino para analisar o tráfego das informações no barramento I2C, iv) uma placa Arduino para injetar falhas com a finalidade de verificar o comportamento dos subsistemas do Cubesat em condições normais e adversas, v) placas dos subsistemas reais do satélite em teste (computador de bordo e cargas úteis), vi) um computador (*Personal Computer* - PC 1) com os ambientes de desenvolvimento necessários para a realização da evolução dos testes do STAE, vii) um computador (PC 2) com o ambiente para realizar as operações do EGSE (Equipamento de Suporte Elétrico de Solo - *Electrical*

Ground Support Equipment) fornecido pelo desenvolvedor do satélite em teste, viii) um EGSE, e ix) uma placa denominada *protoboard*, a qual é uma placa com orifícios e conexões condutoras para montagem de circuitos elétricos experimentais.

A Figura 3.2 apresenta a arquitetura do STAE. O barramento de comunicação utilizado na abordagem apresentada é o I2C.

Figura 3.2 - Arquitetura do STAE.



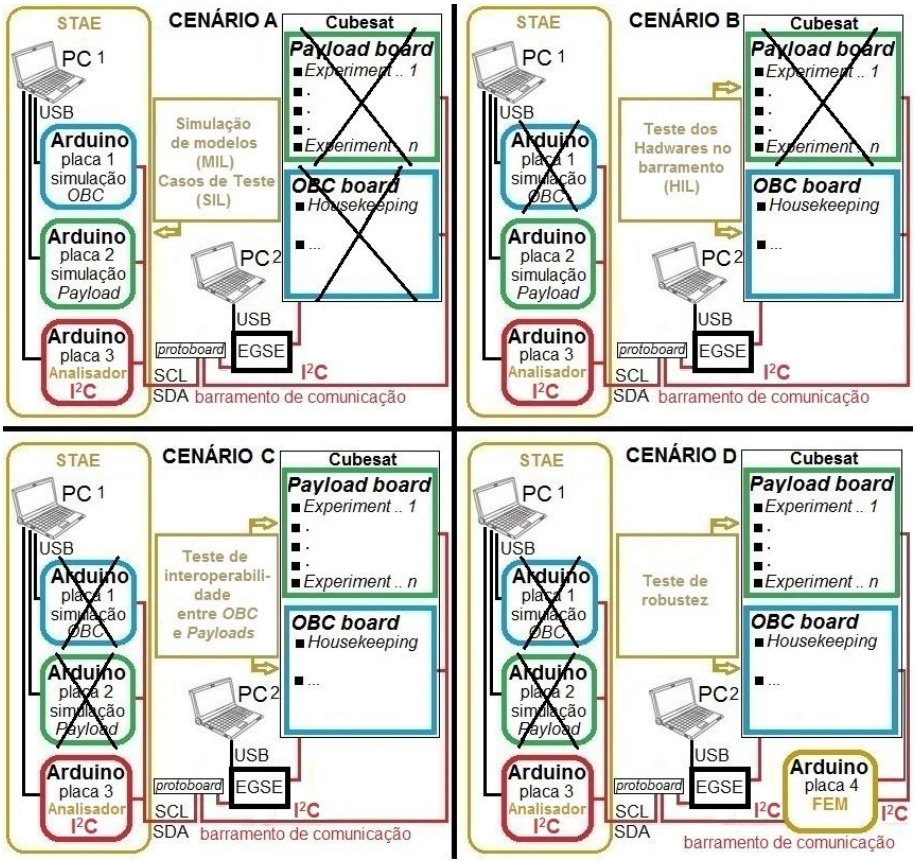
O STAE comporta testar os requisitos lógicos entre os subsistemas computador de bordo e cargas úteis do Cubesat.

3.2 Cenários do STAE

A abordagem desenvolvida utiliza quatro cenários objetivando a reutilização do STAE de uma forma incremental implementadas por modelos. Dentro do processo de V&V durante as fases de desenvolvimento cada cenário desempenha respectivamente as seguintes funções: a) verificar os requisitos de interoperabilidade de subsistemas intensivos em software que se comunica por meio de modelos de estado que representam o comportamento nominal dos subsistemas comunicantes OBC e carga útil com foco nas interações (MIL) (módulos 1), verificar a robustez dos subsistemas comunicantes por meio de códigos simulados e uso do barramento real de comunicação (SIL) (módulo 2) e validar os casos de teste abstratos gerados por meio de estímulos utilizando a técnica MBT (módulo3), b) validar a implementação dos subsistemas reais no loop (HIL) separadamente (módulo 4), c) validar a interoperabilidade dos

subsistemas reais no loop (módulo 5), e d) injetar falhas no canal de comunicação para testar os requisitos de robustez especificados entre os subsistemas reais comunicantes (módulo 6). As placas computacionais programáveis utilizadas para simular o comportamento dos subsistemas em interação são Arduino (McRoberts, 2011). A Figura 3.3 apresentas esses cenários.

Figura 3.3 - Cenários de teste baseados no barramento de comunicação I2C.



Esses cenários tem como foco a capacidade de testar os requisitos de interoperabilidade e segurança dos subsistemas OBC e cargas úteis.

4 RECURSOS UTILIZADOS NO STAE

Os recursos utilizados para a realização de testes utilizando o STAE são:

- Uma placa Arduino Mega (simular Computador de Bordo).
- Uma placa Arduino UNO (simular uma carga útil).

- Uma placa Arduino UNO (executar analisador lógico para verificar sinais no barramento I2C).
- Uma placa Arduino UNO (executar o Mecanismo Emulador de Falhas - FEM).
- Uma placa Ethernet (conectar a placa Arduino MEGA cuja finalidade é armazenar informações em base de dados).
- Um EGSE (Equipamentos de Suporte em Solo para Teste Elétrico do NanosatC-BR2).
- Um computador pessoal (PC) (executar o STAE).
- Um computador pessoal (PC) (inicializar e verificar o EGSE).
- Um *protoboard* (conectar as placas ao barramento I2C).
- Um concentrador (*hub*) (conectar a placa ethernet acoplada no Arduino ao computador executando o STAE).
- Ambiente de modelagem Uppaal versão 4.1.19 (técnica MBT).
- Ambiente de desenvolvimento Arduino versão 1.6.5-r5.
- Ambiente de modelagem Eclipse versão Neon.1 release 4.6.1 (técnica MDE).
- Ferramenta Yakindu SCT 2.9.3 (gerar os códigos computacionais de forma automatiza para placas Arduino).
- Aplicativo para geração de códigos computacionais de forma automatizada dentro do Yakindu (make.exe).
- Analisador Lógico denominado "LogicSniffer - Logic Analyser Client" versão "logic_analyzer-agla_v0_10" (capturar as informações no barramento de comunicação por meio do aplicativo ols-0.9.7.2).
- Aplicativo Xampp versão 3.2.2 (executar o ambiente de banco de dados MySQL).

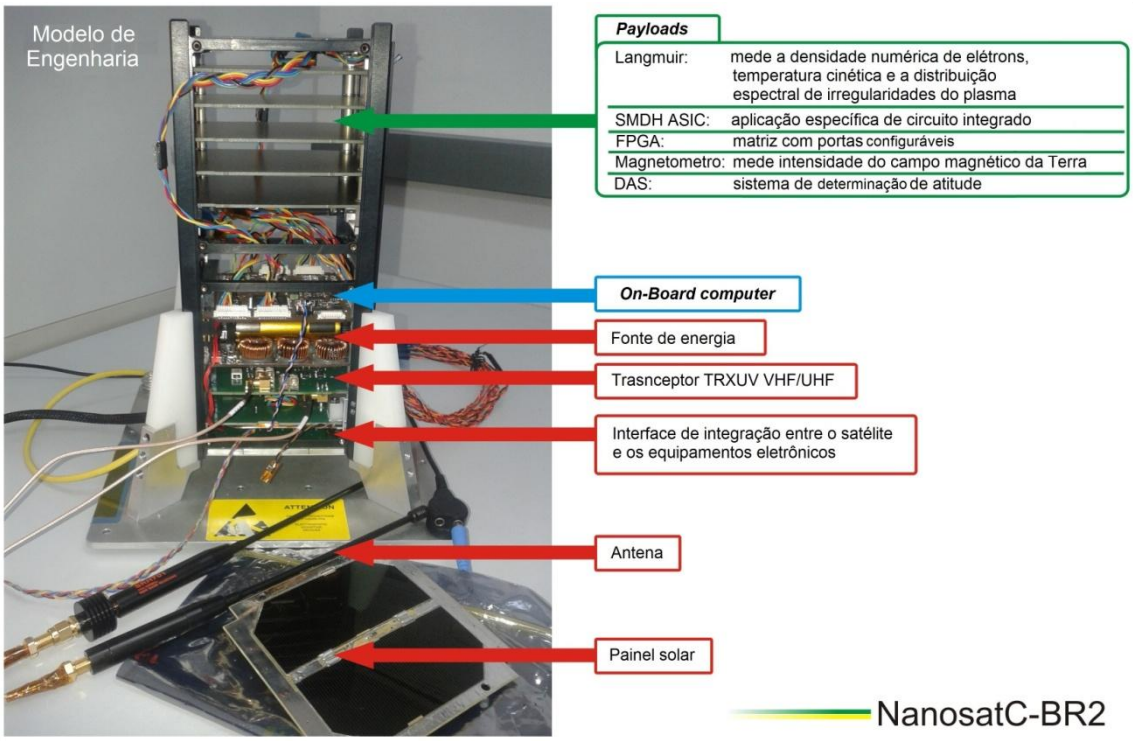
Uma planilha de dependabilidade e tabela FMEA foram desenvolvidas para a abordagem apresentada, cujo objetivo é realizar o tratamento de falhas nos subsistemas comunicantes do nanosatélite identificando e evitando falhas. Essa planilha e tabela foram desenvolvidas na planilha Excel da Microsoft e não são apresentadas neste manual por não necessitarem de instalação e configurações específicas. A planilha foi desenvolvidas com base nos conceitos de dependabilidade da Taxonomia de Avizienis et al. (2004). Essa planilha é utilizada para classificar as falhas identificadas durante os testes e relacionar as mitigações que devem ser realizadas para evitar essas falhas. A tabela é utilizada para analisar causa-efeito e relacionar as possíveis falhas identificadas com os mecanismos e procedimentos para realizar as mitigações necessárias para evitar essas falhas.

5 ESTUDO DE CASO NANOSATC-BR2

O Instituto Nacional de Pesquisas Espaciais (INPE) vem desenvolvendo pesquisas em plataformas de nanosatélites buscando adquirir conhecimento e desenvolver novas tecnologias para atender necessidades de âmbito nacional. Atualmente o INPE, juntamente com parcerias, desenvolve um Cubesat 2U, denominado NanosatC-BR2 (Schuch et al., 2017), cuja missão é coletar dados de campo magnético da Terra e testar em voo a resistência de circuitos à radiação entre outras novas tecnologias projetadas no Brasil.

A Figura 5.1 apresenta a arquitetura do NanosatC-BR2, sendo que suas carga úteis (*payload* - PL) não estão integradas no satélite.

Figura 5.1 – Arquitetura do NanosatC-BR2.



A interoperabilidade entre os subsistemas do satélite incluindo as cargas úteis é feito por meio do barramento de comunicação I2C.

5.1 Arquitetura de testes

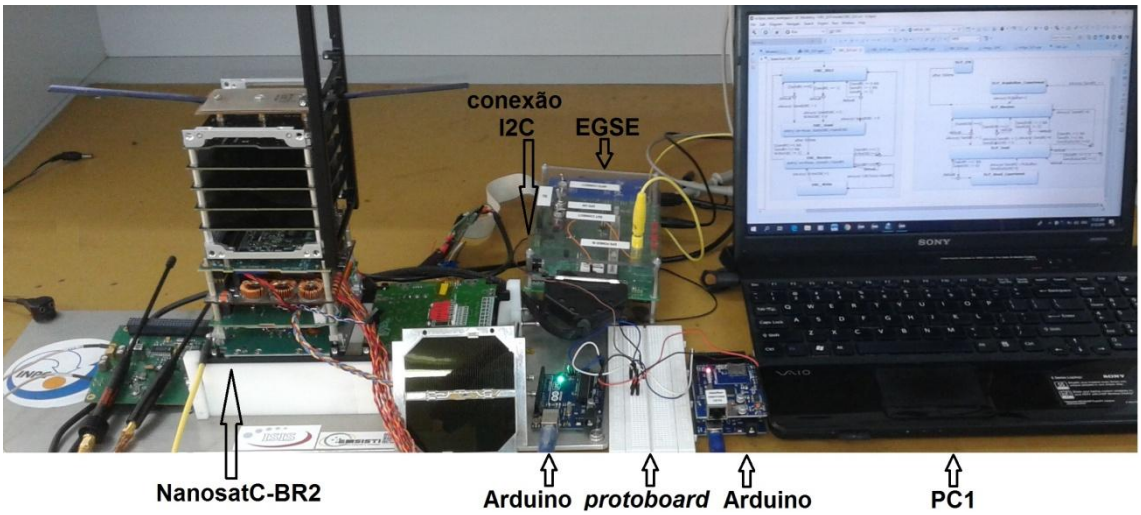
A arquitetura de testes evolui da seguinte forma:

- Inicialmente as placas computacionais Arduino, contendo os códigos computacionais simulando o comportamento dos subsistemas reais em interação do satélite, são conectadas por meio de uma placa denominada *protoboard*.
- Posteriormente as placas dos subsistemas reais são conectadas as placas Arduino por meio da *protoboard* antes de haver a integração no satélite.
- Havendo a validação dos subsistemas reais, esses são integrados ao satélite.
- O EGSE é conectado ao barramento de comunicação do satélite.

- As placas Arduino são conectadas ao barramento do satélite por meio do EGSE.

A Figura 5.2 apresenta a arquitetura de testes contendo as placas Arduino conectadas ao satélite por meio do EGSE. O satélite está parcialmente integrado.

Figura 5.2 – Arquitetura de teste.



Esta arquitetura permite executar a evolução dos testes para a abordagem desenvolvida.

5.2 Especificação dos requisitos

A sistemática é desenvolvida a partir dos requisitos dos subsistemas comunicantes sendo um o Computador de Bordo (*On-Board computer* - OBC) e o outro uma Carga Útil (*Payload* - PL). Os requisitos básicos especificados desses subsistemas para a realização da modelagem de comportamento nominal foram:

- O subsistema OBC assume o papel de mestre e seu comportamento nominal na interação com a carga útil PL deve ser representado pelos seguintes estados: 1) estar em operação avaliando informações recebidas pela PL (estado *IDLE*), 2) enviar comandos (*Command*) para a PL (estado *Send*), 3) receber informações (Information) da PL para definir a próxima atividade (estado *Receive*), e 5) gravar na área de memória os dados lidos do experimento da PL (estado *Write*);

- O subsistema carga útil PL assume o papel de escravo e seu comportamento nominal na interação com o subsistema OBC deve ser representado pelos seguintes estados: 1) estar em operação (estado *ON*) aguardando início da comunicação com o OBC, 2) receber comandos (*Command*) do OBC (estado *Receive*) para avaliar a próxima atividade a ser realizada, 3) enviar informações (*Information*) solicitadas pelo OBC (estado *Send*), 4) realizar aquisição dos dados do experimento e armazenar no buffer (estado *Acquisition_Experiment*), e 5) ler os dados do experimento (estado *Read_Experiment*) e enviar para o OBC (estado *Send*).

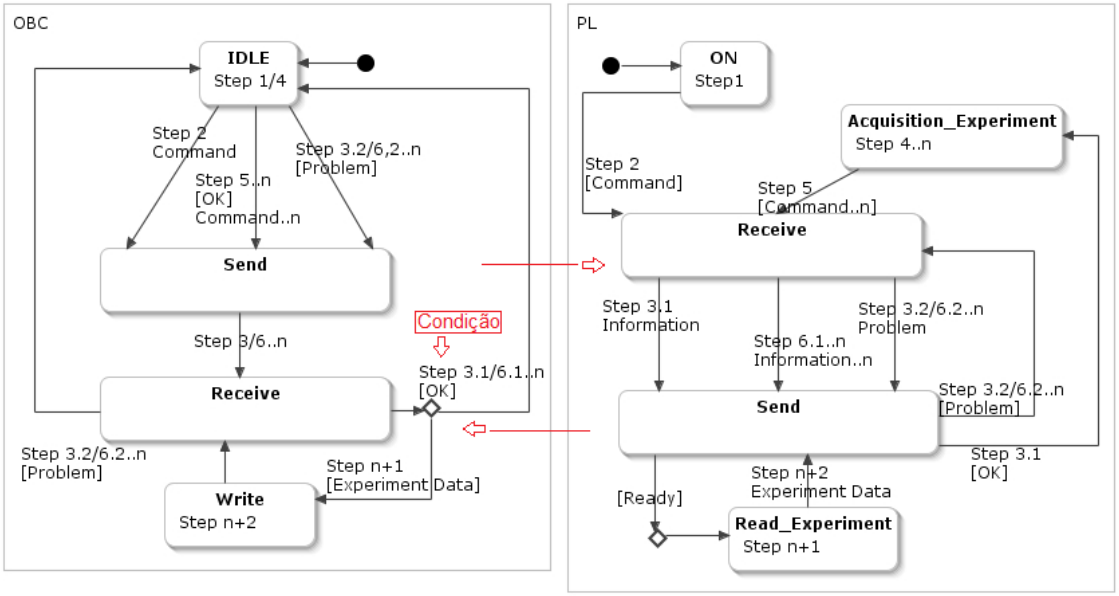
Outras informações identificadas e selecionadas para apresentação da abordagem foram às condições excepcionais entre os subsistemas. As condições selecionadas foram: 1) no caso do OBC receber uma informação não reconhecida deve informar que houve problema e deve reenviar o primeiro comando, 2) no caso da PL receber um comando não reconhecido deve informar que houve problema e deve ler novamente o primeiro comando reenviado pelo OBC, 3) no caso da PL reconhecer o comando recebido deve informar que está certo, 4) quando a PL terminar a aquisição do experimento deve informar que está pronta para ler novos comandos e deve enviar os dados para o OBC, e 5) quando a PL terminar de enviar os dados para o OBC deve informar que está tudo certo e deve retornar a aquisição de novos dados do experimento.

5.3 Diagrama de atividades

O diagrama de atividades (Pimentel, 2015) é utilizado para a representação das atividades dos subsistemas em interação de forma gráfica, o qual possibilita uma visualização e identificação das etapas realizadas de cada subsistema assim como a interação entre eles.

Tendo como foco a interação entre o OBC e uma PL, os diagramas podem ser representados conforme representado na Figura 5.3.

Figura 5.3 – Diagramas de atividades.

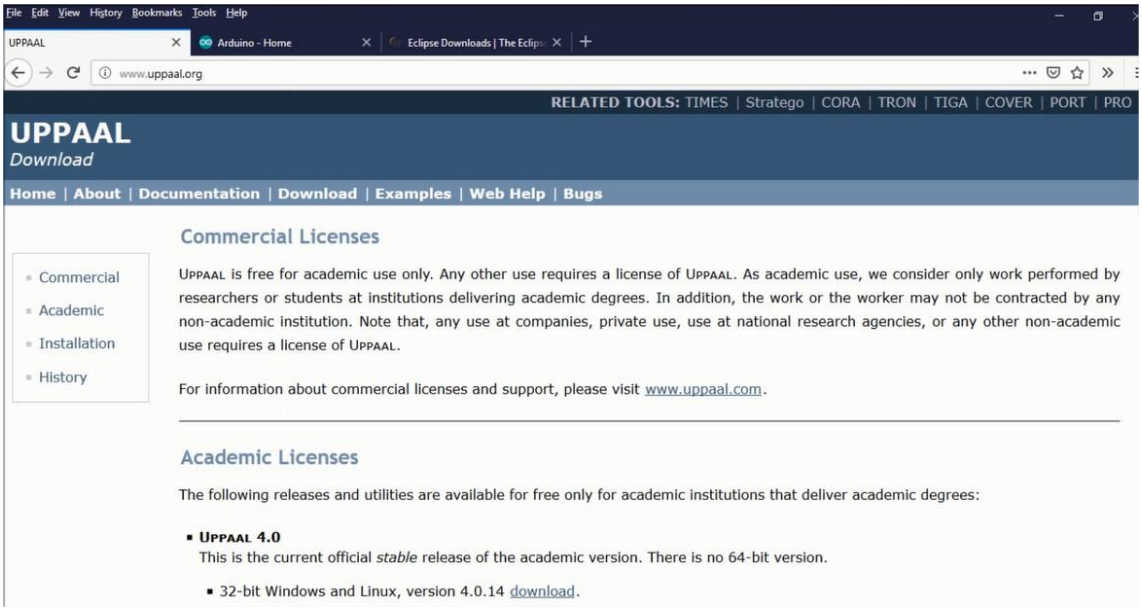


Os diagramas são utilizados como base para a realização das modelagens dos subsistemas em interação.

5.4 Modelagem e simulação do comportamento nominal

Diversas ferramentas de modelagem permitem simulação do comportamento na interação entre os subsistemas submetidos a testes no início do desenvolvimento do projeto. A ferramenta de modelagem selecionada para a implementação da MBT na sistematização dos testes foi a Uppaal versão 4.1.19 - rev. 5648 (UPPAAL, 2009). A Figura 4.4 apresenta o site para abaixar esse aplicativo.

Figura 4.4 – Site da Uppaal (www.uppaal.org).



Dentro do ambiente Uppaal é realizada a modelagem da Máquina de Estado Finita de cada subsistema a ser testado em interação.

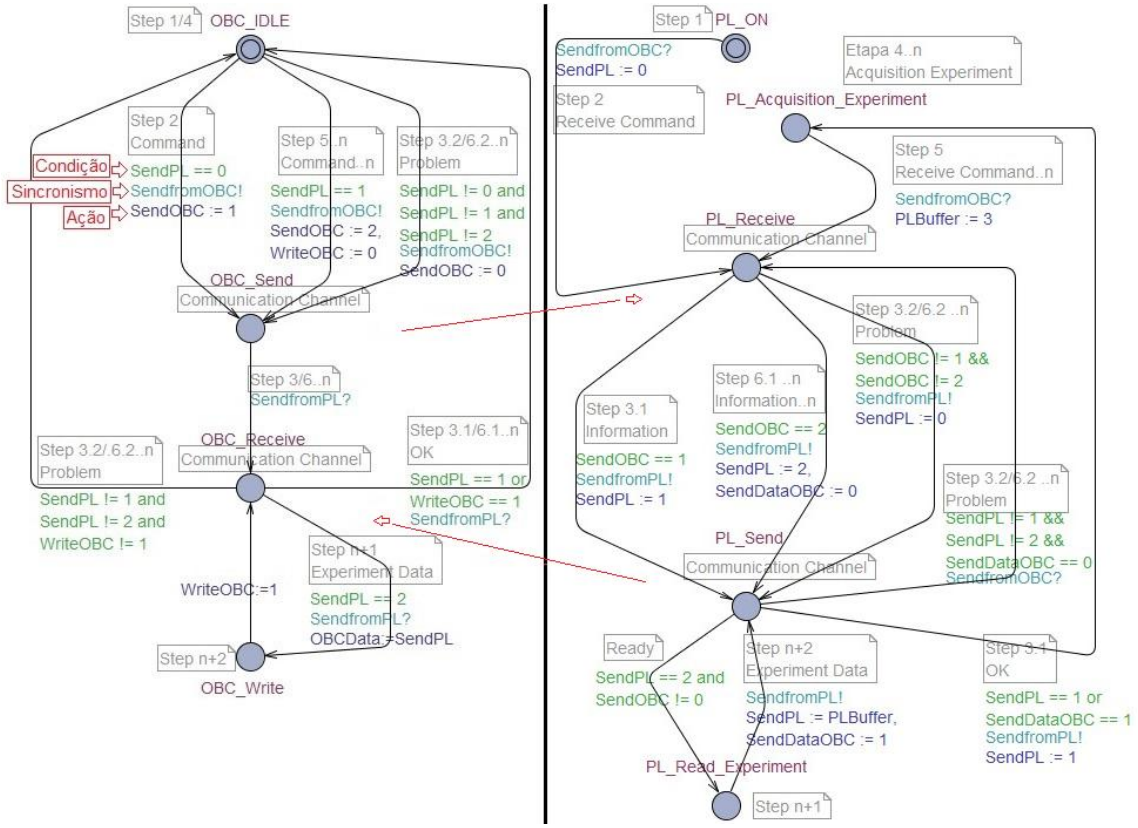
Na sistematização dos testes da abordagem desenvolvida a modelagem dos subsistemas OBC em interação com a PL o sincronismo é realizado por meio dos identificadores SendfromOBC! para o OBC enviar comandos à SLP e SendfromPL? para receber informações. Assim como, SendfromPL! para a PL enviar informações e SendfromOBC? para receber.

Nas modelagens a serem apresentadas os comandos e informações são por opções alteradas para representações numéricas. Essas alterações são apenas para haver comparações numéricas e não alfanuméricas. Sendo assim, no modelo OBC: i) *Command*=1, ii) *Command..n*=2, e iii) *Problem*=0. Na modelagem da PL: i) *Information*=*OK*=1, ii) *Information..n*=*Ready*=2, iii) *Problem*=0, e iv) *Experiment Data*=3.

Algumas variáveis também foram definidas para armazenamento de informações durante a transferência de informações entre os subsistemas em interação.

Na ferramenta UPPALL as condições são identificadas na cor verde clara, o sincronismo na cor azul claro e as ações na cor azul escuro, conforme identificado com setas na Figura 5.5.

Figura 5.5 – Modelagem das máquinas de estado do OBC (esquerda) e PL (direita).



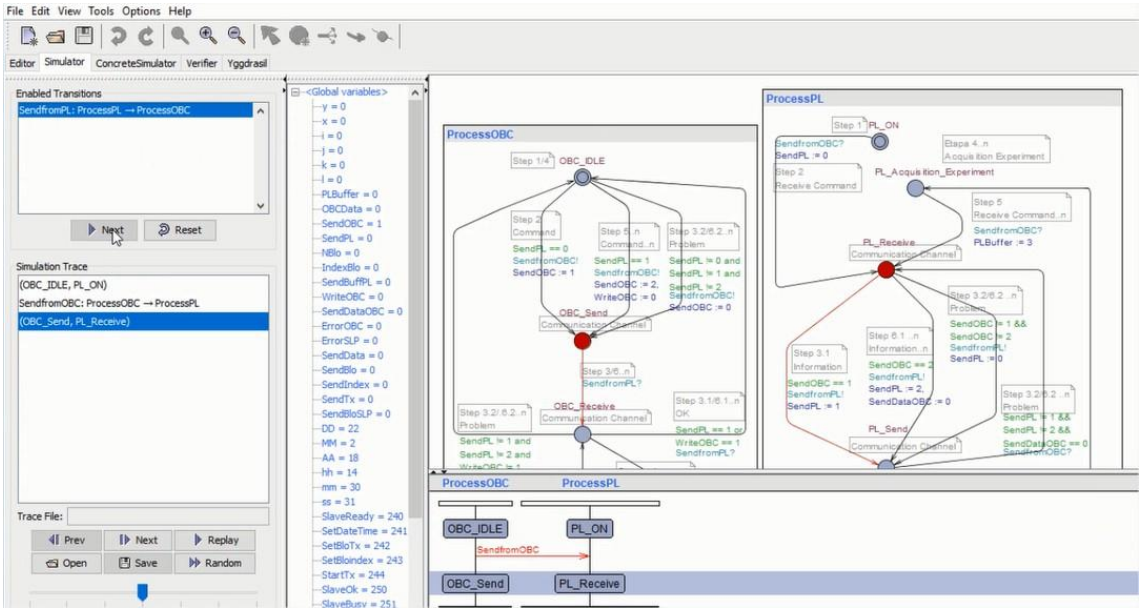
Durante a modelagem foram incluídos nas transições, dentro da opção *Edit Edge*, os *Test Code*. Na geração de casos de teste de forma automatizada os caminhos de teste gerados são identificados com os *Test Code* definidos. Um exemplo de *Test Code* definido foi *OBC_IDEL->OBC_Send*. Esse *Test Code* representa a transição quando o OBC passa do estado *IDLE* para o estado *Send*.

Algumas declarações no UPPAAL foram: const int N=15; typedef int [0,N-1] id_t; chan SendfromPL; chan Sendfrom OBC; int _reach_=0; int _single_=0, entre outras.

Algumas declarações dos sistemas foram: ProcessOBC = OBC(); ProcessPL = PL(); system ProcessOBC, ProcessPL;.

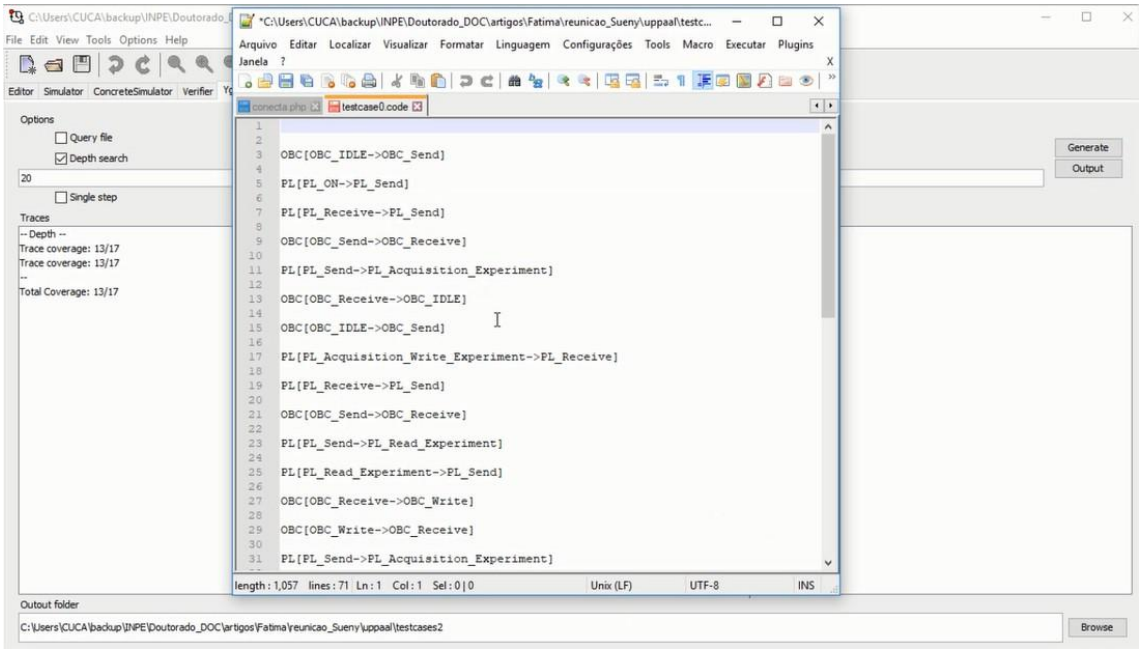
Selecionar o *submenu Simulator* para realizar a simulação das máquinas de estado em interação. A simulação apresenta os estados de cada máquina durante a interação nos momentos de sincronismo, os valores das variáveis e um diagrama de sequência dos sincronismos ocorridos. A figura 5.6 apresenta a tela de simulação.

Figura 5.6 – Simulação de sincronismo das Máquinas de Estado.



Selecionar o *submenu Yggdrasil* para gerar os casos de teste de forma automatizada. Os procedimentos para geras os casos de teste são: i) B*rowse*: selecionar a pasta aonde será gerado os casos de teste, ii) *Options*: selecionar o número de casos de teste a ser gerado, iii) *Generate*: gerar os casos de teste sendo apresentado a quantidade em *Traces*, e iv) *Output*: enviar os casos de teste para a pasta selecionada. A Figura 5.7 apresenta um caso de teste gerado.

Figura 5.7 – Caso de tese.

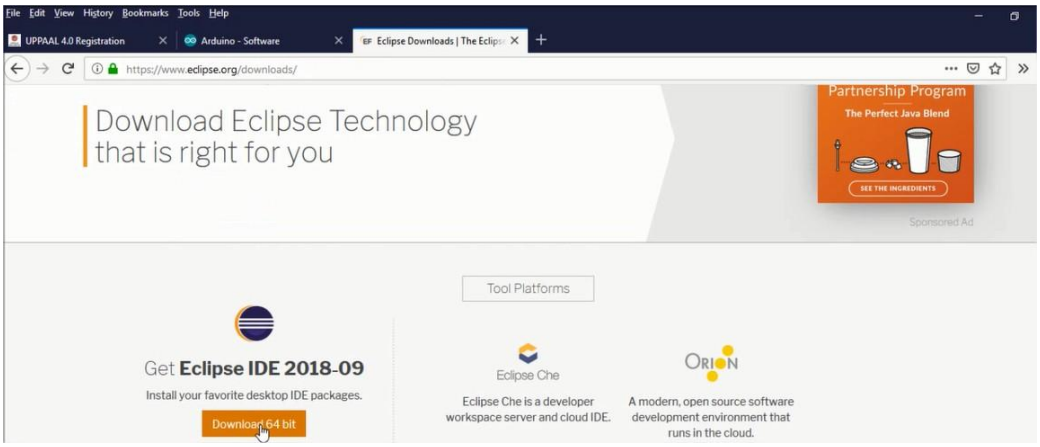


Foi utilizado um editor de programas denominado Notepad para apresentar o caso de teste gerado.

5.5 Ambiente de desenvolvimento Eclipse

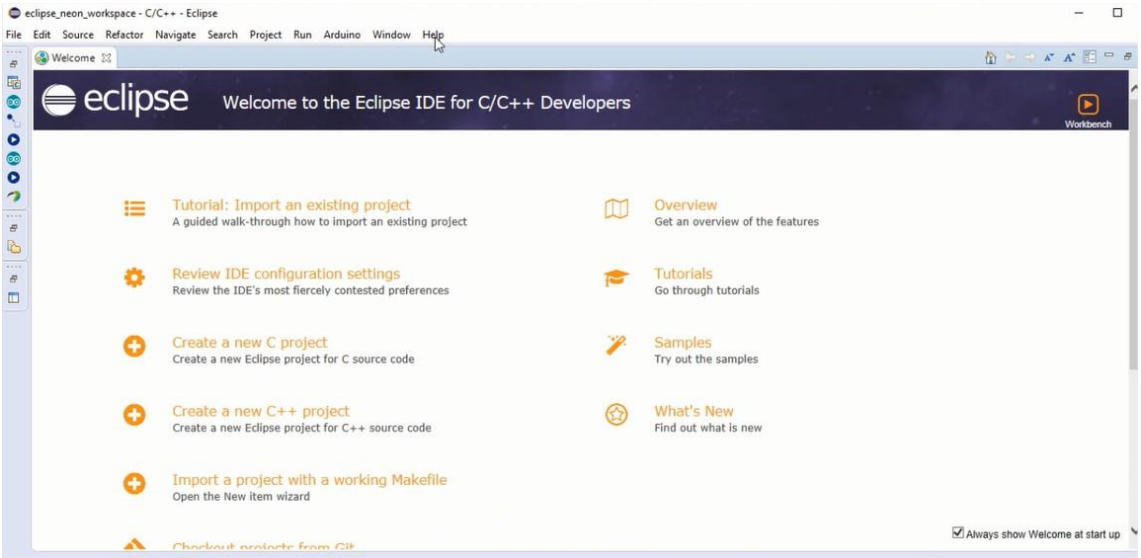
A versão Eclipse instalada para realizar a transformação de modelo é Eclipse IDE for C/C++ Developers, Version: Neon.1 Release (4.6.1) (arquivo: eclipse-cpp-neon-1a-win32-x86_64.zip). O arquivo .zip foi descompactado e instalado dentro da pasta C:\software\eclipse. Dentro dessa pasta fica instalado o eclipse.exe. No diretório C:\ foi criado uma pasta denominada eclipse_neon_workspace aonde são gravados os projetos desenvolvido nos Eclipse. Também foi criado outra pasta de trabalho dentro do diretório C:\ denominada eclipse_neon_worspace2 aonde é criado um segundo projeto. Dessa forma é permitido embarcar os códigos de cada projeto em placas computacionais programáveis distintas. A Figura 5.8 apresenta o site para baixar o ambiente Eclipse.

Figura 5.8 – Site do Eclipse (www.eclipse.org/downloads).



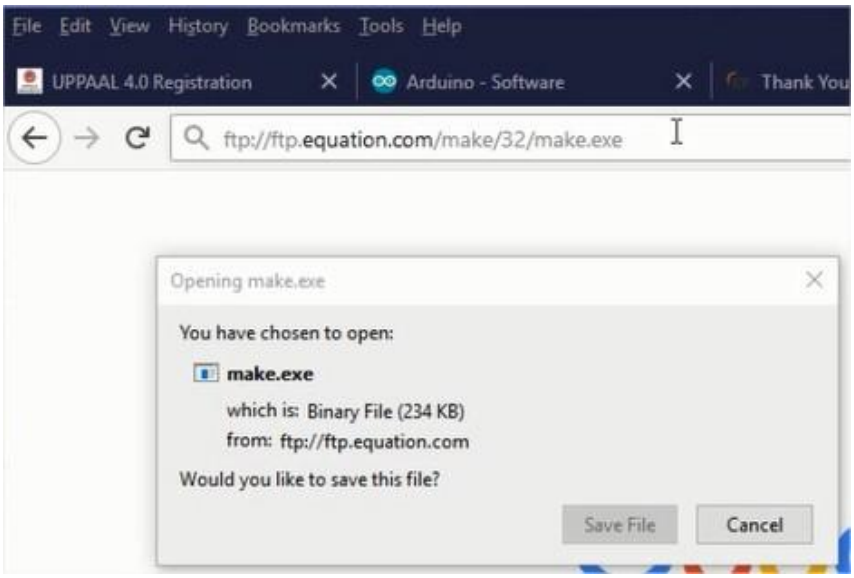
A Figura 5.9 apresenta o ambiente de desenvolvimento Eclipse.

Figura 5.9 – Ambiente de desenvolvimento Eclipse.



Realizar os seguintes procedimentos para a geração dos códigos computacionais de forma automatizada dentro do ambiente Eclipse: i) baixar um arquivo executável denominado make.exe, e ii) instala-lo na pasta C:\Windows. A Figura 5.10 apresenta o site para baixar esse arquivo.

Figura 5.10 – Site do arquivo make.exe (ftp://ftp.equation.com/make/32/make.exe).



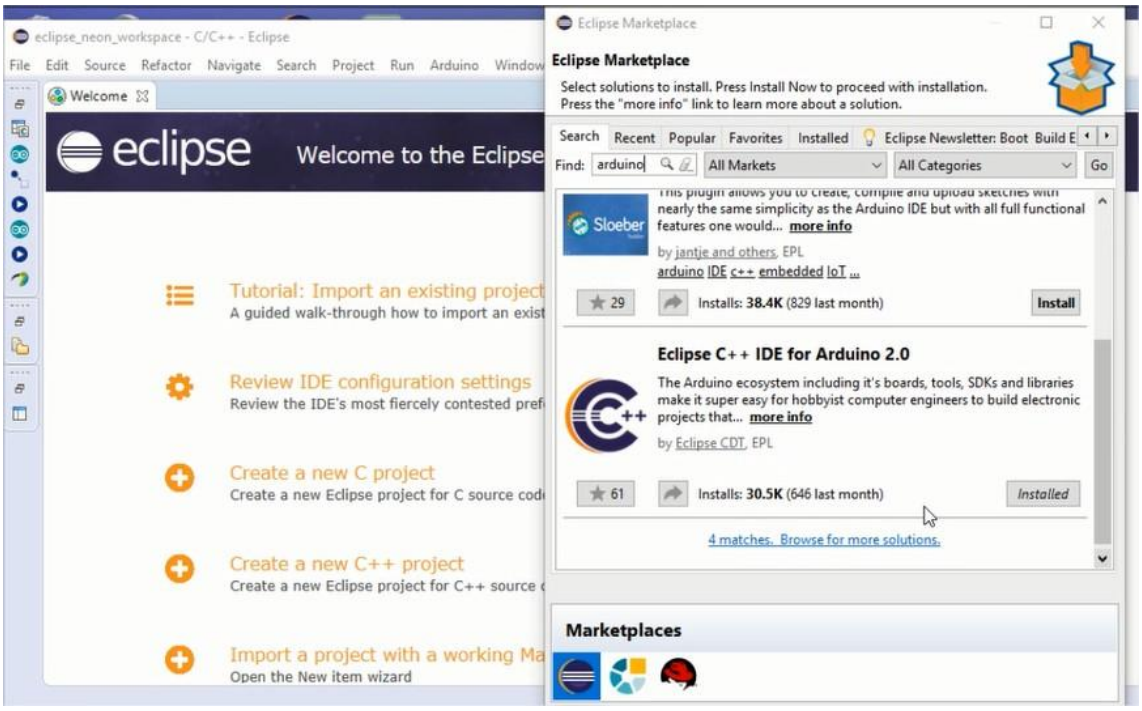
Dentro da plataforma Eclipse foram instaladas as seguintes ferramentas computacionais para atenderem ao desenvolvimento do STAE: i) *Eclipse C++ IDE for Arduino 2.0*, ii) *Arduino AVR Boards 1.6.15*, iii) *CDT releases neon 9.0*, iv) *C++ Development Tools SDK 9.1.0, versão V2*, v) *Yakindu SCT 2.9.3*, e vi) *Yakindu Statechart Tools for Arduino 0.3.0*.

5.6 Ambiente de teste Arduino

O STAE utiliza placas computacionais Arduino para simular o comportamento funcional dos subsistemas do nanosatélite em interação. O ambiente de desenvolvimento Arduino utilizado foi a versão 1.6.5-r5. Essa versão é configurada no ambiente de desenvolvimento Eclipse (Eclipse, 2016) permitindo que os códigos gerados de forma automatizado na ferramenta Yakindu (2017) sejam embarcados nas placas Arduino. O Arduino é instalado dentro da pasta C:\Program File (x86)\Arduino, sendo baixado no site www.arduino.cc.

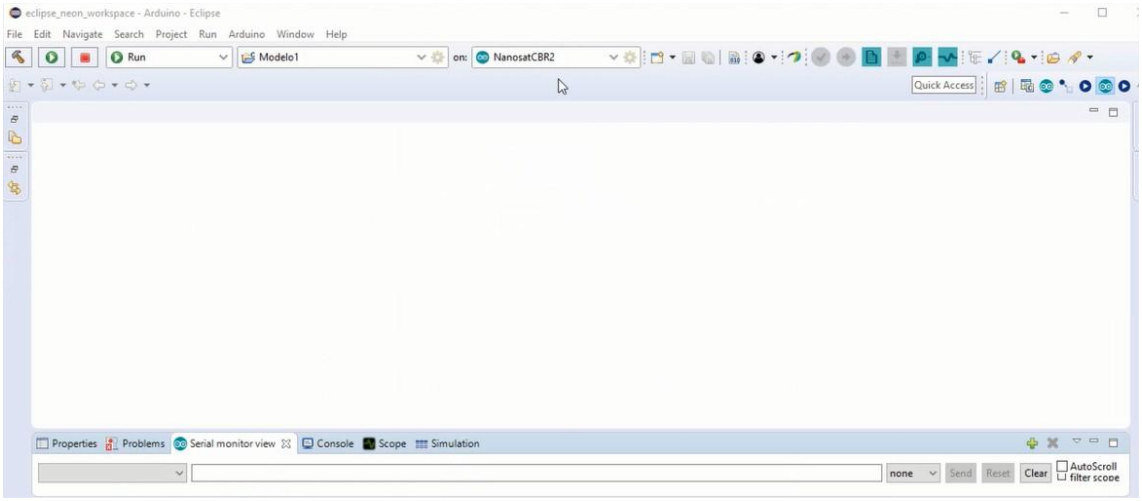
Realizar os seguintes procedimentos para instalar no ambiente Eclipse o ambiente Arduino: no ambiente Eclipse selecionar o *menu Help*, *submenu Eclipse Marketplace*, em *Find*: digitar Arduino e instalar Eclipse C++ IDE for Arduino 2.0. A Figura 5.11 apresenta o ambiente de desenvolvimento do Arduino a ser instalado.

Figura 5.11 – Ambiente de desenvolvimento do Arduino para Eclipse.



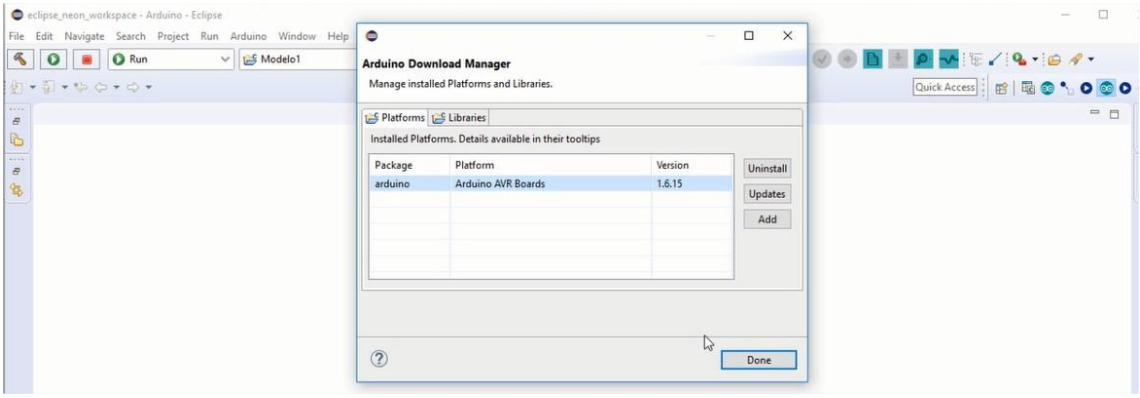
Após essa instalação o ambiente Eclipse irá apresentar ícones e janelas para definição do ambiente Arduino conforme apresentado na Figura 5.12.

Figura 5.12 – Ambiente Arduino dentro do Eclipse.



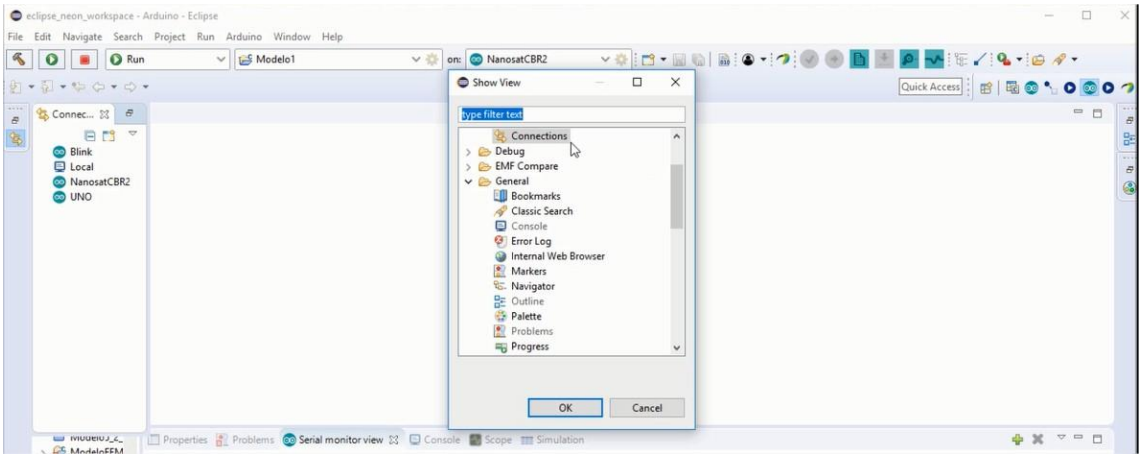
Dentro do Eclipse configurar as placas do Arduino realizando os seguintes procedimentos: no ambiente Eclipse selecionar o *menu Help*, *submenu Arduino Downloads Manager*, selecionar *Platforms*, *Add*, *Package* arduino, *Platform Arduino AVR Boards*, *version* 1.6.15 conforme apresentado na Figura 5.13.

Figura 5.13 – Configuração das placas Arduino no ambiente Eclipse.



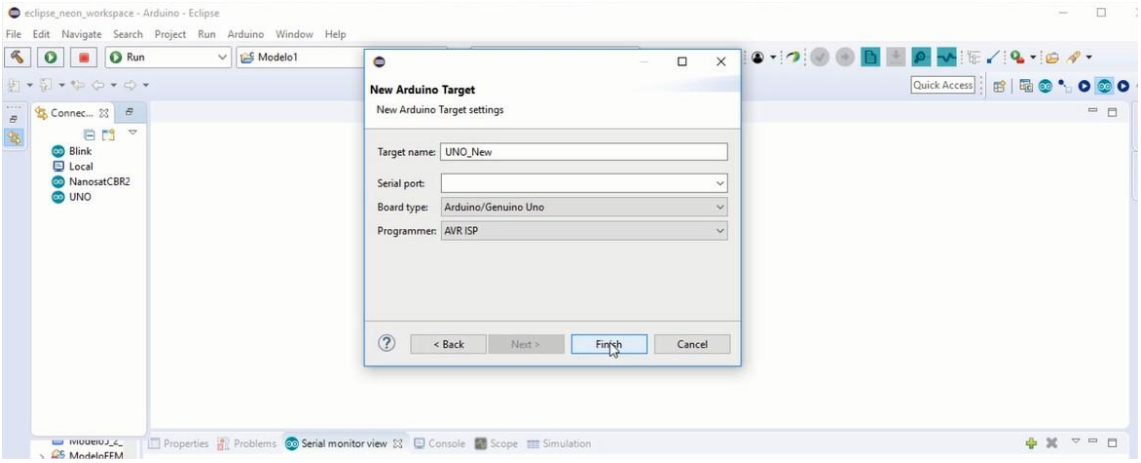
Realizar os seguintes procedimentos para apresentar as conexões com as placas Arduino: no ambiente Eclipse selecionar o *menu Windows*, *submenu show view*, selecionar *others* e *connections*. As conexões são exibidas conforme apresentado na Figura 5.14.

Figura 5.14 – Conexões Arduino.



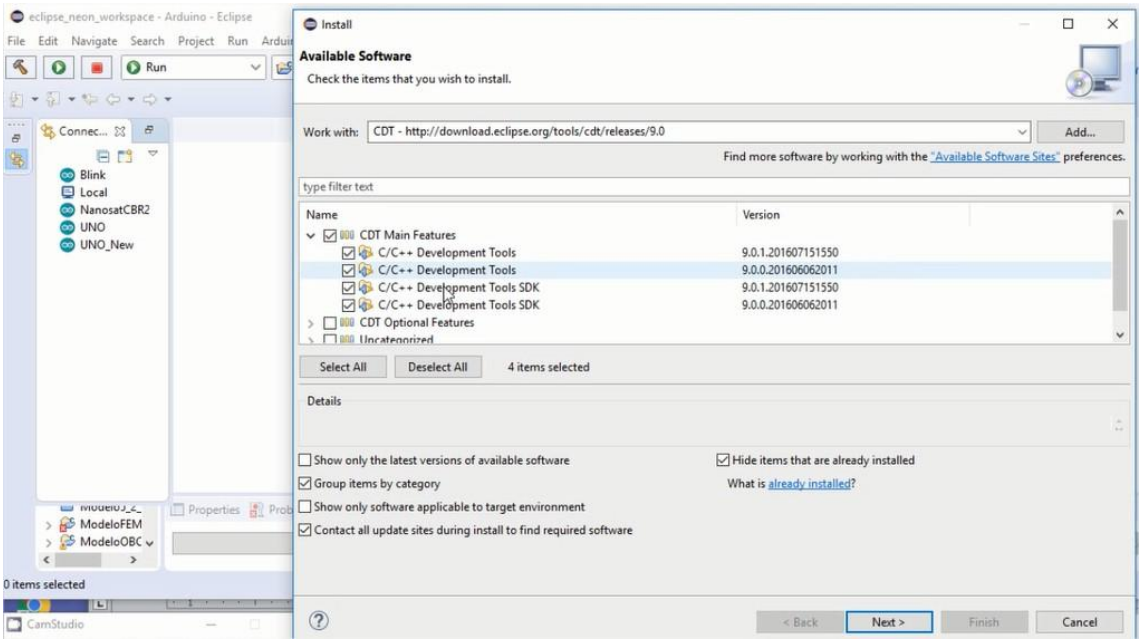
Realizar os seguintes procedimentos para configurar uma placa Arduino: selecionar *New connection* (ícone com um ponto amarelo em *Connections*), selecionar Arduino, definir em *Target name* a placa Arduino a ser utilizada (opção inicial por UNO), selecionar o número da *Serial port* a qual a placa Arduino está conectada, selecionar *Board type* (no caso foi selecionado Arduino/Genuino Uno e selecionar *Programmer* como AVR ISP, conforme apresentado na Figura 5.15.

Figura 5.15 – Configuração da placa Arduino a ser utilizada no Eclipse.



Realizar os seguintes procedimentos para instalar *releases*: no ambiente Eclipse selecionar o *menu help*, *submenu Install New Software*, selecionar *Add*, em *Location* digitar http://download.eclipse.org/tools/cdt/releases/9.0, selecionar *CDT Main Features*, selecionar *C/C++ Development Tools* e *C/C++ Development Tools SDK 9.0.1* e instalar conforme apresentado na Figura 5.16.

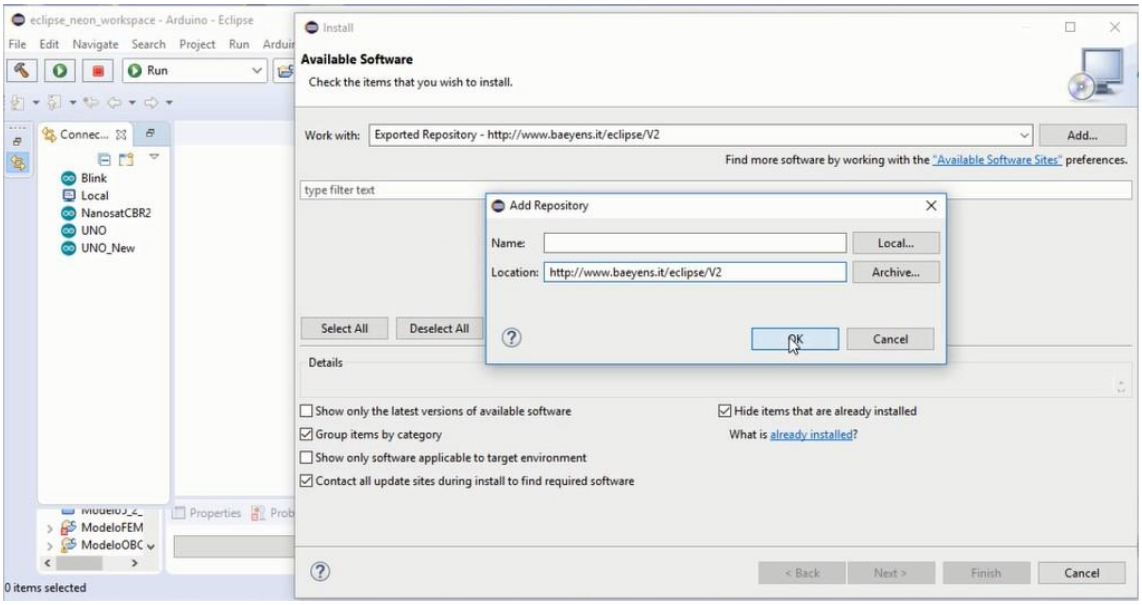
Figura 5.16 – Instalação de *releases* no Eclipse.



Realizar os seguintes procedimentos para instalar os ícones do ambiente Arduino: no ambiente Eclipse selecionar o *menu Help*, *submenu Install New Software*, selecionar *Add*, em *Location* digitar http://www.baeyens.it/eclipse/V2,

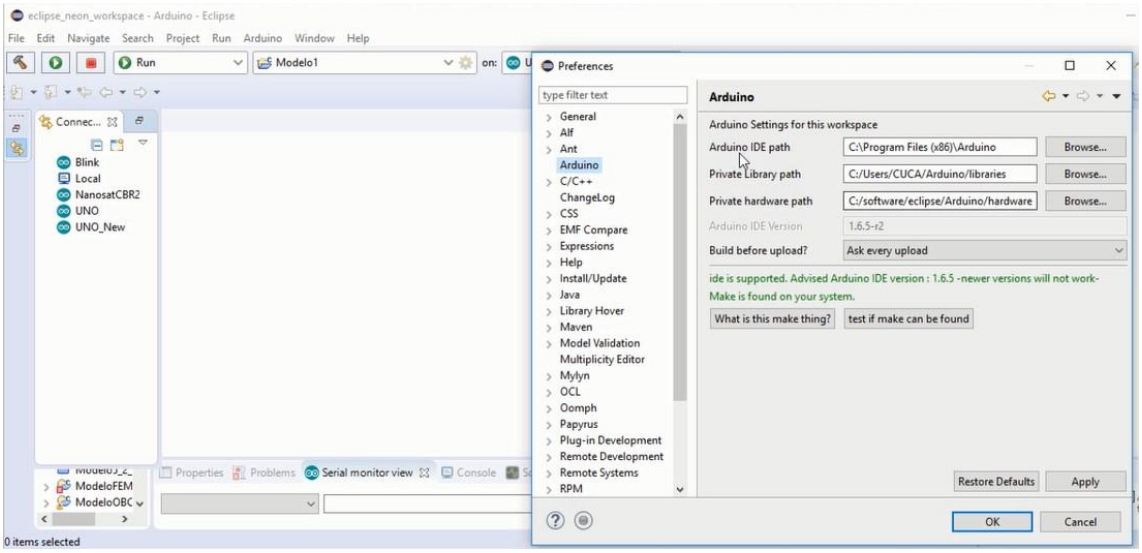
conforme apresentado na Figura 5.17. Para apresentar as versões não deve estar selecionado a opção *Group item by category*.

Figura 5.17 –Versão do Arduino.



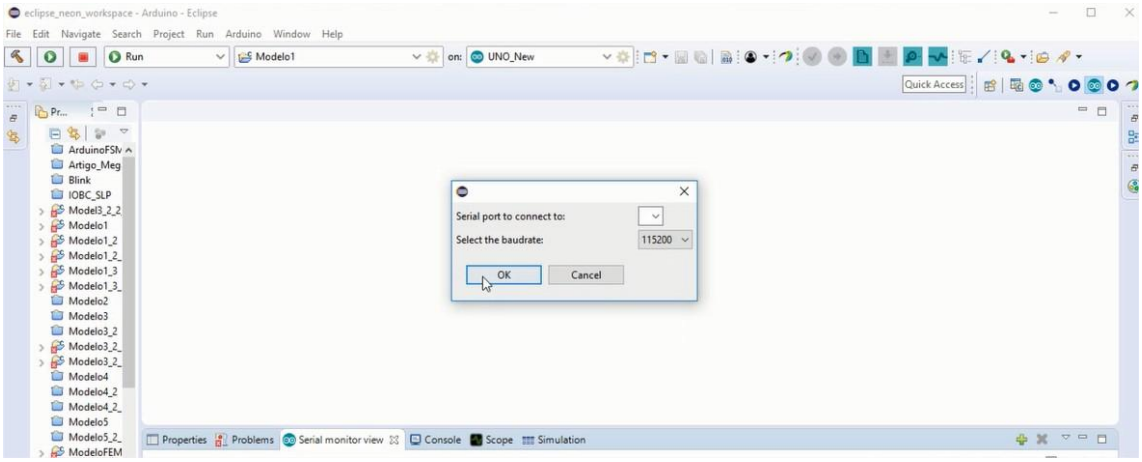
Realizar os seguintes procedimentos para verificar se a versão instalada do Arduino e a localização das bibliotecas estão sendo reconhecidas no ambiente Eclipse: Selecionar o *menu Windows*, *submenu Preferences*, realizar o teste por meio da opção *test id make can be found*. Verificar *Arduino IDE path*, *Private Libary path*, *Private Hardware path e versão do Arduino*, conforme apresentado na Figura 5.18.

Figura 5.18 – Localização dos *paths* do Arduino.



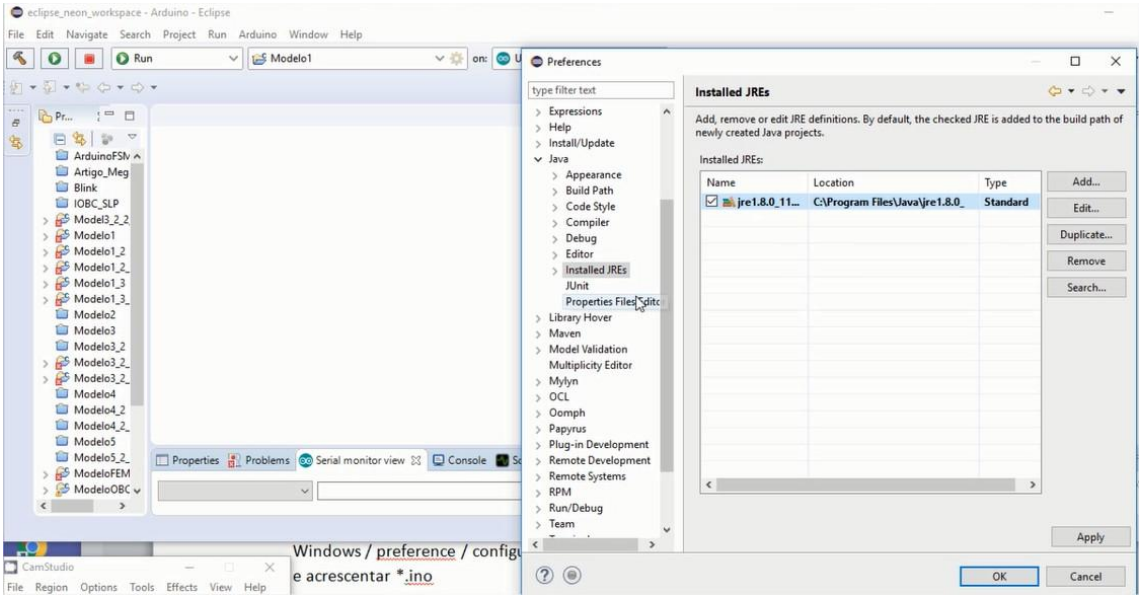
Realizar os seguintes procedimentos para definir a porta e velocidade de transmissão da placa Arduino: selecionar na console *Serial monitor view*, selecionar o sinal **+**, definir a porta serial a qual a placa Arduino está conectada e a velocidade de transmissão definir como 115200, conforme apresentado na Figura 5.19. O sinal **-** desconecta a serial.

Figura 5.19 – Configuração da porta e velocidade de transmissão da placa Arduino.



Verificar a versão do Java instalado seguindo os seguintes procedimentos: no *menu Windows*, *submenu Preferences*, selecionar Java, selecionar *Installed JREs* e verificar a versão do Java. No STAE foi selecionado a versão jdk1.8.0_111, conforme apresentado na Figura 5.20.

Figura 5.20 – Versão do Java.



Realizar os seguintes procedimentos para salvar com frequência os projetos desenvolvidos: selecionar o *menu Windows*, *submenu Preferences*, selecionar *General* e posteriormente *Workspace* e selecionar a opção *Save automatically before build*. Nessa janela apresentam as opções iniciais *Default (Cp1252)* e *Default (Windows)*, sendo mantidas para a abordagem apresentada.

Realizar os seguintes procedimentos para realizar *updates* automaticamente no Eclipse: selecionar o *menu Windows*, *submenu Preferences*, selecionar, *Install/Update*, selecionar *Automatic Updates*, selecionar as opções *Look for update on the following schedule*, *Search for update and notify me when they are available* e *Notify me once about updates*.

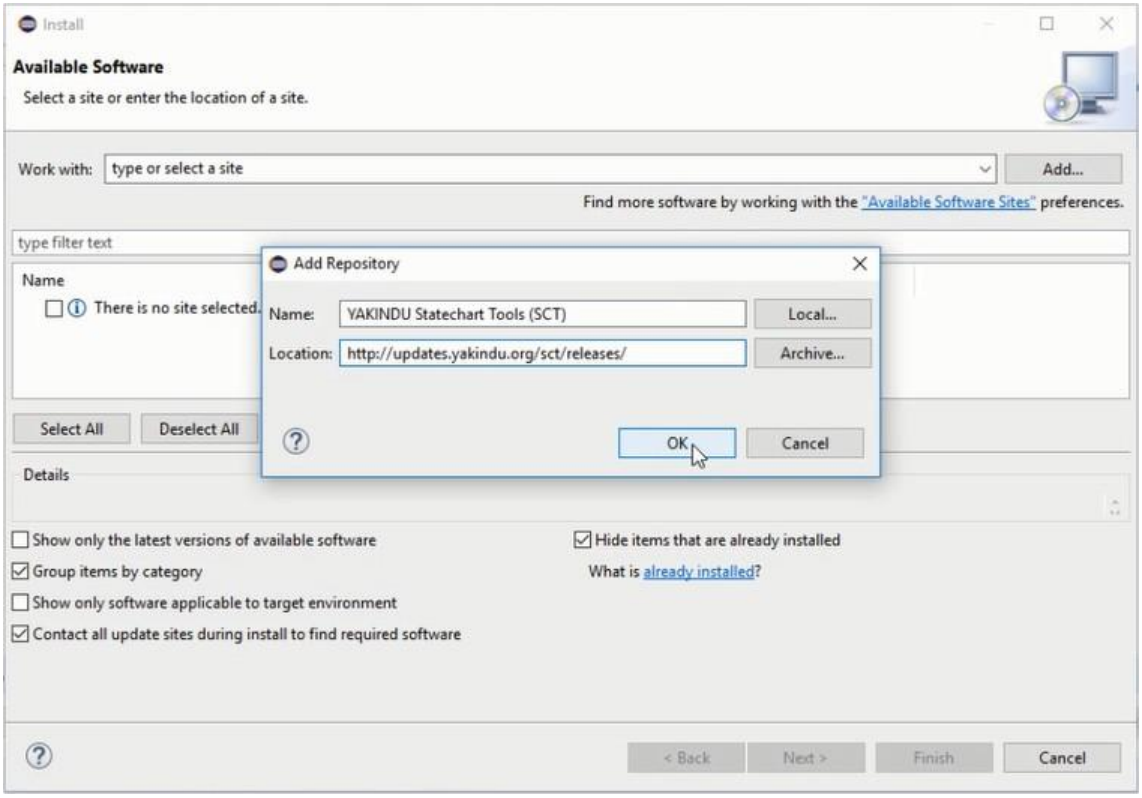
Para acrescentar arquivos com extensão .ino do Arduino ir ao *menu Windows*, *submenu Preferences*, selecionar *C++*, selecionar *File type*, selecionar *Add*, acrescentar **.ino* em *filename* e *C++ Source File* em *Description*.

5.7 Ferramenta Yakindu

A ferramenta Yakindu instalada dentro do ambiente Eclipse permite gerar códigos computacionais de forma automatizada, a partir das modelagens, e embarcar os códigos em placas Arduino.

Realizar os seguintes procedimentos para instalar a ferramenta Yakindu no ambiente: selecionar o *menu Help*, *submenu Install New Software*, selecionar *Add*, em *Name* digitar *Yakindu Statechart Tools (SCT)* e em *Location* digitar *http://updates.yakindu.org/sct/releases/*, conforme apresentado na Figura 5.21.

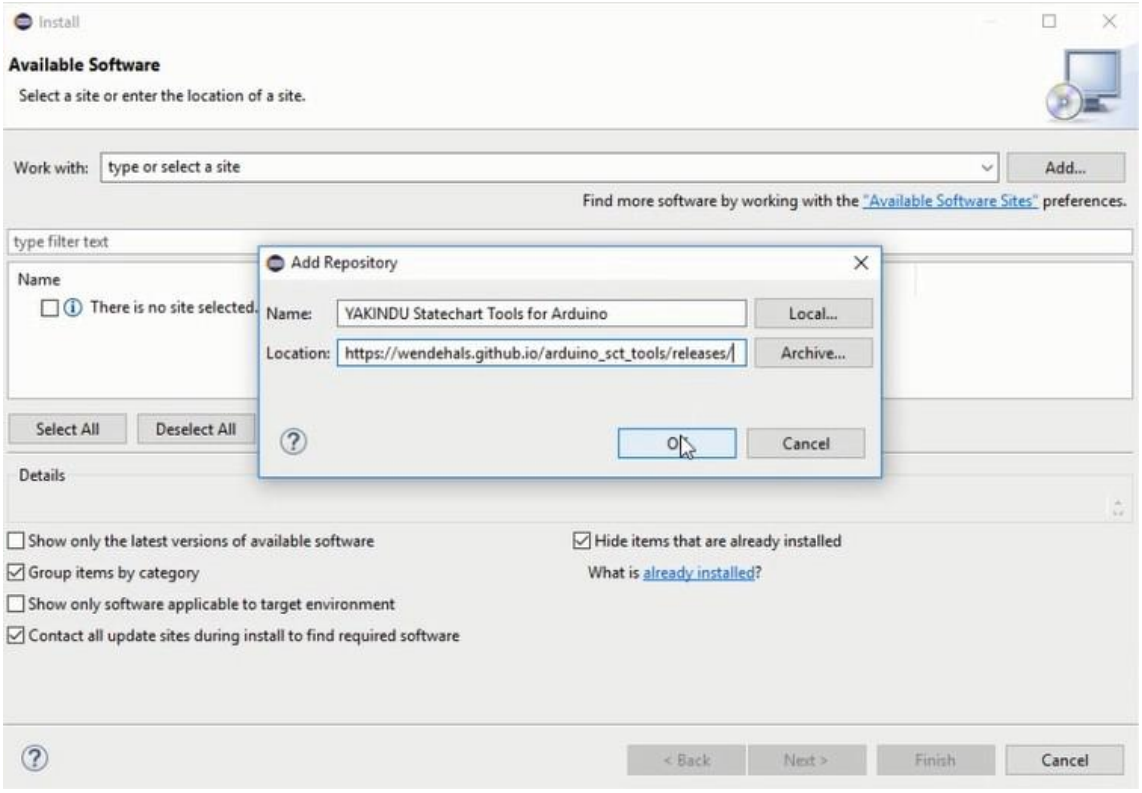
Figura 5.21 – Instalação da ferramenta computacional Yakindu no Eclipse.



Após clicar em *OK* selecionar *Yakindu SCT*. Para o STAE foi selecionado a versão 2.9.3.

Realizar os seguintes procedimentos para a Yakindu gerar códigos para as placas Arduino: selecionar o *menu Help*, *submenu, Install New Software*, selecionar *Add*, em *Name* digitar *Yakindu Statechart Tools for Arduino*, em *Location* digitar *https://wendehals.github.io/arduino_sct_tools/releases/*, conforme apresentado na Figura 5.22.

Figura 5.22 – Instalação da ferramenta computacional Yakindu para Arduino.



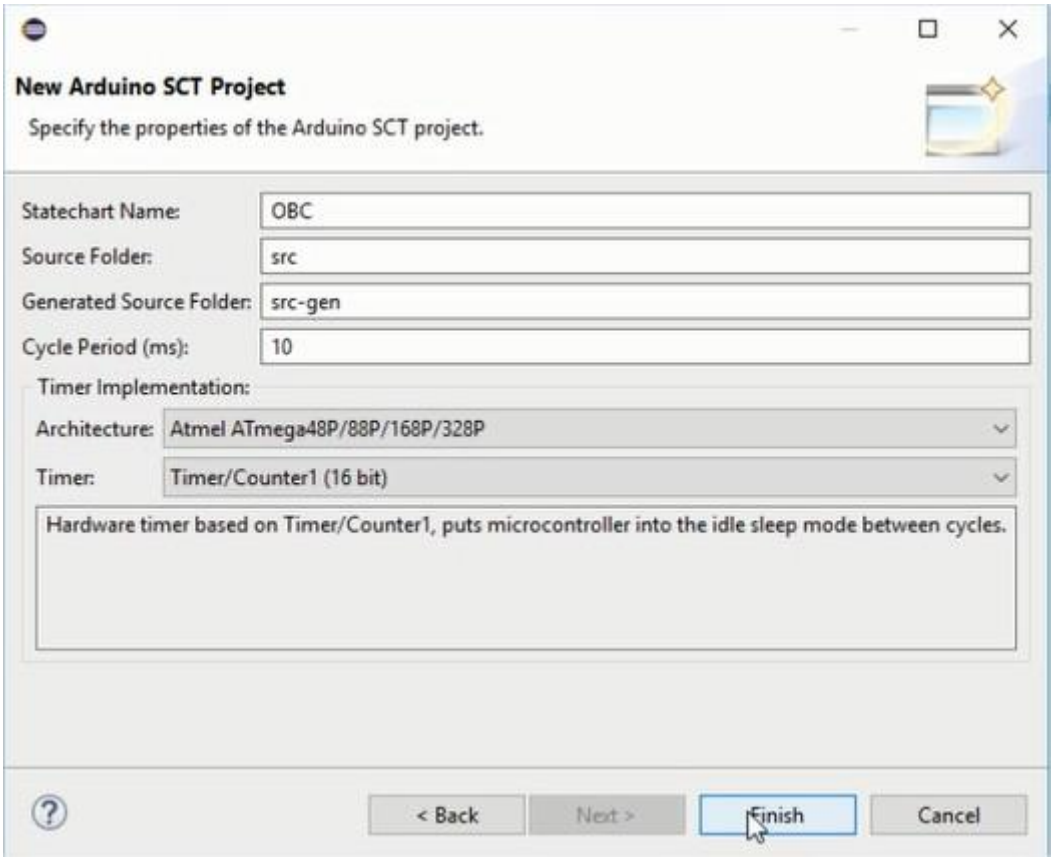
Após clicar em *OK* selecionar Yakindu *Statechart Tools for Arduino* e selecionar a versão. Para a abordagem apresentada foi instalado a versão 0.3.0.

5.8 Transformação de modelos

Realizar os seguintes procedimentos para gerar a modelagem dos subsistemas sob teste utilizando a ferramenta Yakindu: i) conectar a placa Arduino a porta USB do computador contendo o ambiente de desenvolvimento Yakindu, ii) no *menu* superior desse ambiente selecionar *New Launch Target*, iii) definir nome para a placa Arduino, iv) selecionar porta de conexão com a placa, v) selecionar o modelo da placa, vi) selecionar *AVR ISP*, vii) no *menu* superior do ambiente Eclipse selecionar *File*, viii) *submenu Project*, selecionar *Yakindu Statechart Tools for Arduino,* ix) posteriormente *Arduino Project with Yakindu Statechart*, e x) definir o *Nome do Projeto* em *Project name.* Na sequência será apresentado uma tela para definir as seguintes configurações: *Statechart Name*: "nome da máquina de estado", *Source Folder: src*, *Generated Source Folder: src-gen*, *Cycle Period (ms): 10*, *Arquitecture*: "definir placa Arduino

utilizada", *Timer: Timer/Counter1 (16 bit)*. Essas configurações foram definidas no STAE conforme apresentado na Figura 5.23.

Figura 5.23 – Definição da placa Arduino no Yakindu.



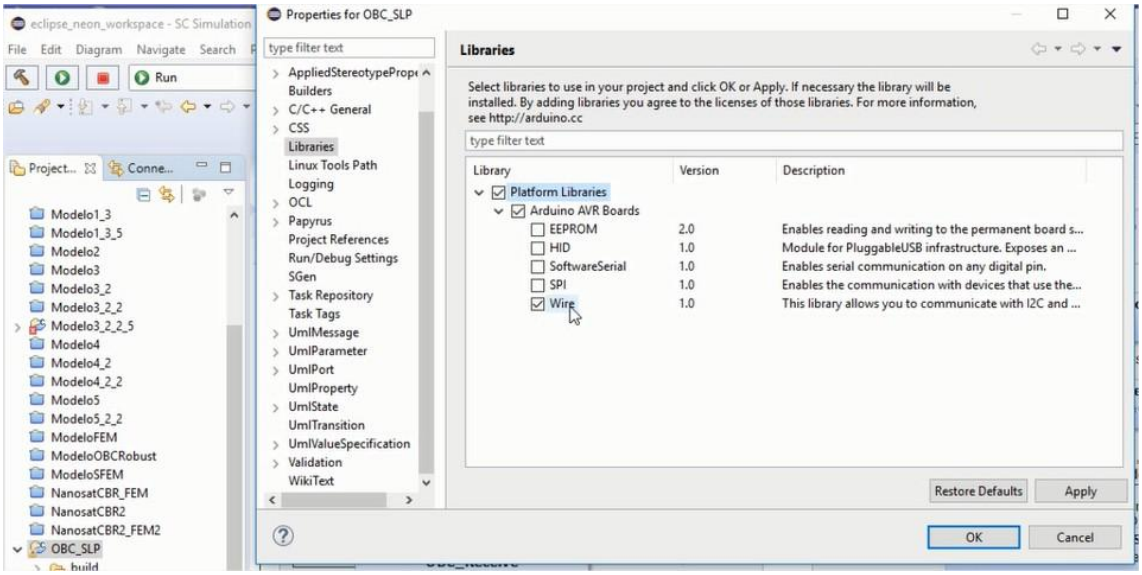
Na sequência é questionado se o projeto é para ser convertido em Xtest. A opção é confirmada.

A partir desses procedimentos é gerada uma pasta dentro do projeto identificada com o nome *model* e dentro dessa pasta são criados dois arquivos. Esses arquivos são: i) um arquivo com extensão .sct, o qual é realizado a modelagem a partir da modelagem na UPPALL, e ii) um arquivo com extensão .sgen, o qual a partir dele são gerados os códigos do modelo de forma automatizada selecionando a opção *Generate Code Artifacts.*

Realizar os seguintes procedimentos para incluir as bibliotecas de comunicação (protocolo de comunicação I2C): clicar com o botão do lado direito do *mouse* sobre o "*Nome do Projeto"* em desenvolvimento e selecionar a opção *Properties*, selecionar *Libraries*, selecionar *Platform Libraries*,

selecionar *Arduino AVR Boards* e selecionar *Wire*, conforme apresentado na Figura 5.24.

Figura 5.24 – Configuração da biblioteca I2C.

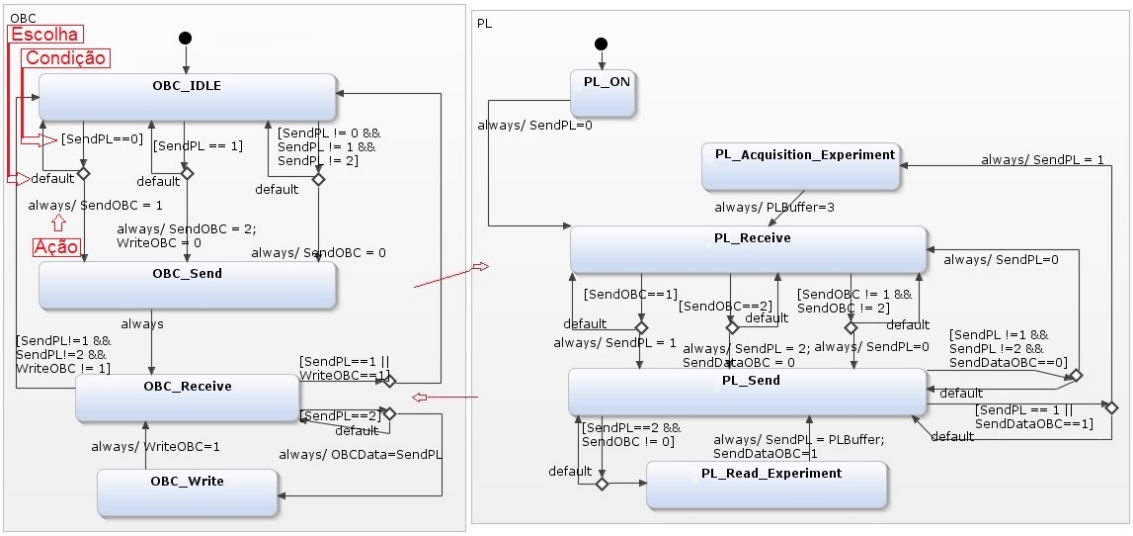


Na transformação de modelos na Yakindu são utilizadas as representações dos estados, transições, ações e condições representadas na Uppaal para realizar as modelagens do OBC e da PL na Yakindu.

As alterações realizadas na modelagem na Yakindu comparada a Uppaal são: i) retirada do sincronismo definido na Uppaal, ii) utilização do recurso escolha (*choice*), iii) a condição está representada entre colchetes [ ], e iv) a ação está após o comando *always*.

A Figura 5.25 apresenta a máquina de estado do OBC à esquerda modelada na Yakindu. Os eventos recebidos pelos estados são por meio da variável *SendOBC* (comando a ser enviado pelo OBC) ou *SendPL* (informação recebida pela PL). A máquina de estado da PL está à direita modelada na Yakindu. Os eventos recebidos pelos estados são por meio da variável *SendPL* (informação a ser enviada pela PL) ou *SendOBC* (informação recebida pelo OBC).

Figura 5.25 – Máquinas de estado do OBC (esquerda) e da PL (direita).



Algumas variáveis definidas em interface são: var SendOBC:integer = 0, var SendPL:integer = 0, var on:boolean, var stateOBC: string, var statePL: string, entre outras.

Realizar a simulação clicando com o botão do lado direito do *mouse* sobre o arquivo .sct, selecionar *Runs As* e optar por *Statechart Simulation*. Os valores das variáveis aparecem na janela de simulação.

5.9 Geração dos cõdigos computacionais

Realizar os seguintes procedimentos para a geração dos códigos separadamente dos subsistemas modelados no ambiente de desenvolvimento Eclipse/Yakindu: i) conectar as placas Arduino ao computador, contendo o ambiente de desenvolvimento Eclipse/Yakindu, por meio dos cabos específicos dessas placas utilizando as portas de comunicação apropriadas, ii) abrir dois ambientes de desenvolvimento Yakindu, iii) utilizar um ambiente para configurar uma placa conectada e o outro ambiente para a outra placa, vi) configurar um ambiente com uma modelagem de um subsistema a ser testado e o outro ambiente com a outra modelagem, v) gerar os códigos computacionais de forma automatizadas de cada modelagem em seu respectivo ambiente selecionado a opção *Generate Code Artifacts* no arquivo com extensão .sgen criado a partir da definição do projeto do modelo do

subsistema a ser testado, vi) implementar manualmente nos códigos gerados de cada modelagem as bibliotecas necessárias para haver interoperabilidade entre os subsistemas comunicantes (protocolo de comunicação), e vii) implementar manualmente outras bibliotecas necessárias (banco de dados, entre outras).

A Yakindu gera no projeto da modelagem em desenvolvimento uma pasta denominada src e outra pasta src-gen. Na pasta src é gerado um arquivo denominado *NomeProjetoConnector.cpp*. Nesse arquivo permite ser incluído comandos para definição da porta serial do Arduino e também controles de entrada e saída. Na pasta src-gen é gerado um arquivo denominado *NomeProjeto.cpp*. Nesse arquivo permite ser implementado de forma manual as bibliotecas e alguns códigos do protocolo de comunicação. A Tabela 5.1 apresenta as linhas de comandos do arquivo OBCConnector.cpp. Após duas barras (//) estão os comentários de algumas linhas de comando.

Tabela 5.1 - Arquivo OBCConnector.cpp.

Arquivo OBCConnector.cpp

```
/* Generated by YAKINDU Statechart Tools for Arduino v0.3.0 */
#include "OBCConnector.h"
#include "Wire.h" // protocolo I2C
#include <string.h>
#include "../src-gen/OBC.h"
int saida_local; // variável indicando leitura ou gravação de experimento
OBCConnector::OBCConnector(OBC* statemachine) {
        this->statemachine = statemachine;
}
void OBCConnector::init() {
          pinMode(LED_BUILTIN, OUTPUT);
          Serial.begin(115200); // velocidade da porta serial
}
void OBCConnector::runCycle() {
          digitalWrite(LED_BUILTIN, statemachine->get_on()); // verifica variável on
          saida_local = statemachine->get_saida(); // transfere informação de saida para saida_local
          Serial.println (" ");
          Serial.print (" set_saida : ");
          Serial.println (statemachine->get_saida());
           if (statemachine->get_on() == true)  { // no caso de on estar TRUE
             Serial.println("Reading Experiment PL");  // OBC está lendo experimento
             Serial.print(" Saida local ...: ");
               Serial.print (saida_local);  // apresenta valor lido
           }
           else  {  // no caso de on estar FALSE
             Serial.println("Writing Buffer OBC");  // OBC está gravando experimento
             Serial.print(" Saida : ");
             Serial.print (saida_local);  // apresenta valor gravado
           }
        }
```

A Tabela 5.2 apresenta algumas linhas de comandos referentes ao protocolo I2C implementadas manualmente nos arquivos OBC.cpp e PL.cpp. Cada linha possui um comentário após duas barras (//) descrevendo o que será realizado.

Tabela 5.2 - Arquivo OBC.cpp (esquerda) Arquivo PL.cpp (direita) contendo comandos do protocolo I2C.

Arquivo OBC.cpp	Arquivo PL.cpp

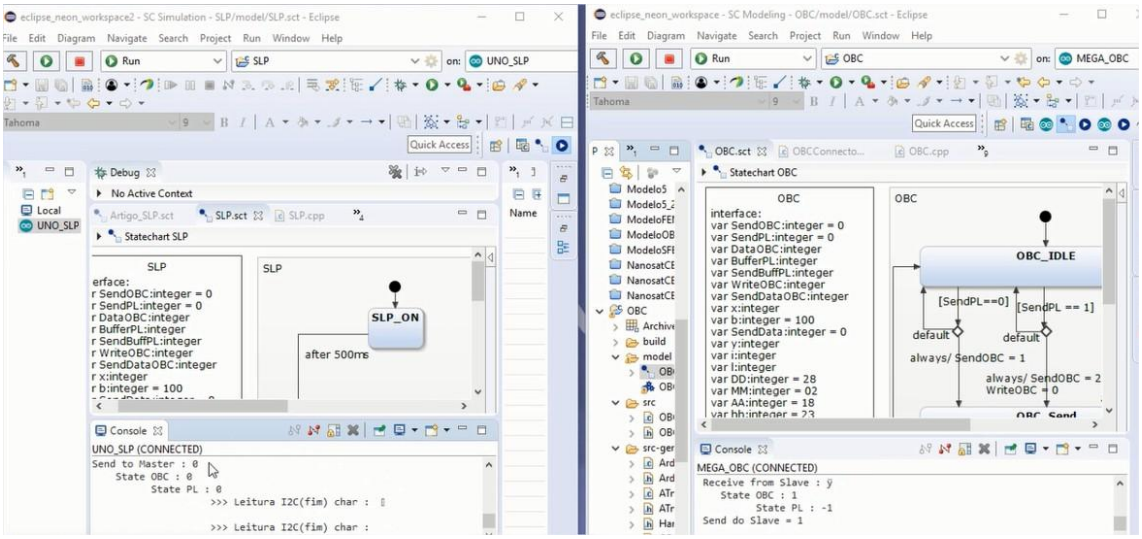
#include "Wire.h" // inclui biblioteca I2C
char c; // define variável c tipo caractere
char read_I2C; // define variável real_I2C tipo caractere
void OBC::init() // inicia função
{
Wire.begin(0x3C); // inicia comunicação com endereço
do escravo (PL)
}
void Artigo_OBC::runCycle() // executa função
{
Wire.beginTransmission(0x3C); // inicia transmissão
Wire.write(ifaceOBC.TC);// envia informação
Wire.endTransmission();// termina transmissão
Wire.begin(0x3C); // inicia transmissão
Wire.requestFrom(0x3C,5); // solicita leitura de informação
while (Wire.available()) // avalia comunicação
{
char c = Wire.read(); // lê informação
read_I2C=c; // armazena informação lida na variável c
}
Wire.endTransmission(); // termino da transmissão
}
void OBC::enact_OBC_I2C_Connection_r1_Read_from_Slave() // executa função
{
ifaceOBC.Read = read_I2C; // transfere a informação lida para variável de leitura do OBC
}
	#include "Wire.h" // inclui biblioteca I2C
	char c; // define variável c tipo caractere
	char x; // define variável x tipo caractere
	char read_I2C; // define variável read_I2C tipo caractere
	void receiveEvent (int howMany) // inicia função para recebimento de informação
	{
	while (Wire.available()) // avalia comunicação
	{
	char c = Wire.read (); // lê informação
	read_I2C = c; // armazena a informação lida na variável read_I2C
	}
	} // término da função de recebimento de informação
	void PL::init() // executa função
	{
	Wire.begin (0x3C); // inicia comunicação I2C
	Wire.onReceive (receiveEvent); // executa função de recebimento de informação
	x=ifacePL.Send; // transfere a informação da variável lida pela PL para a variável x
	Wire.endTransmission(); // termino da transmissão
	}
	void PL::enact_PL_I2C_Connection_r1_Read_from_Master() // executa função
	{
	ifacePL.Read = read_I2C; // transfere a informação lida
	para a variável de leitura da PL
	}

5.10 Execução dos códigos computacionais

Realizar os seguintes procedimentos para embarcar e simular os códigos dos modelos dos subsistemas nas placas computacionais programáveis utilizando o ambiente Eclipse/Yakindu: i) utilizar os ambientes de desenvolvimento Yakindu, um ambiente com o modelo do OBC e o outro ambiente com o modelo da PL, para embarcar e simular os códigos gerados de cada modelo para as respectivas placas computacionais de cada ambiente, ii) interconectar as placas por meio de um barramento de comunicação conforme o protocolo de comunicação do Cubesat, iii) utilizar os ambientes de desenvolvimento para executarem os códigos de cada respectiva placa, iv) na janela *Connections* de

cada ambiente selecionar o ícone *New Connection* (ícone com um ponto amarelo), v) selecionar a conexão da placa Arduino já previamente configurada, vi) selecionar com o botão do lado direito do *mouse* a opção *Open Command Shell*, vii) após aparecer na Console que a conexão está realizada selecionar no *menu* superior de cada plataforma de desenvolvimento, na janela superior do lado esquerdo de *on:*, a pasta do arquivo com extensão .sct criado a partir do modelo, viii) selecionar no menu superior de cada ambiente, na janela do lado direito de *on:,* o console da placa a ser embarcado os códigos do modelo a ser testado, e ix) selecionar no *menu* superior de ambos ambientes *Launch in RUN* para embarcar e executar os códigos nas respectivas placas, e x) utilizar os consoles conectados dos ambientes para verificar a execução dos códigos, conforme apresentado na Figura 5.26.

Figura 5.26 – Execução dos códigos do modelos.

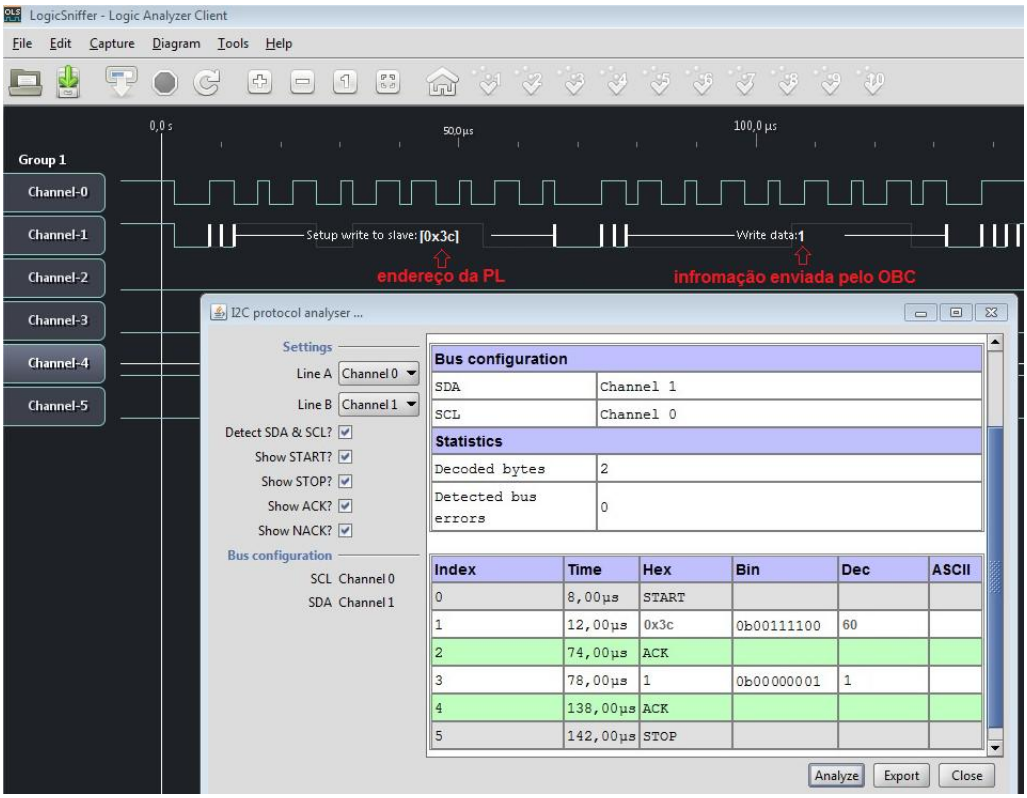


5.11 Analisador lógico

Na arquitetura do STAE é utilizado um analisador lógico para capturar e analisar as informações que trafegam no barramento de comunicação I2C. O analisador é conectado ao barramento e identifica os sinais SDA (*data*) e SCL (*clock*) do protocolo de comunicação. Conforme o padrão I2C é identificado o endereçamento da mensagem, os sinais de leitura ou gravação, o pacote de mensagem e o término de cada pacote. Em destaque na parte inferior da Figura 5.27 apresenta as seguintes informações: i) início da mensagem

(*START*), ii) endereçamento do envio da informação pelo OBC (*0x3c* = endereço da PL), iii) reconhecimento da informação (*ACK*), iv) informação enviada pelo OBC (*1*), v) reconhecimento da informação (*ACK*), vi) término do envio da informação (*STOP*), e vii) outras informações como tempo de envio do pacote, representação binária e decimal dessas informações.

Figura 5.27 – Analisador lógico I2C.



O analisador lógico utilizado foi o "*LogicSniffer - Logic Analyzer Client*" versão "logic_analyzer-agla_v0_10", o qual captura as informações por meio do aplicativo ols-0.9.7.2 (LXTREME, 2017). Este analisador está disponibilizado gratuitamente na internet

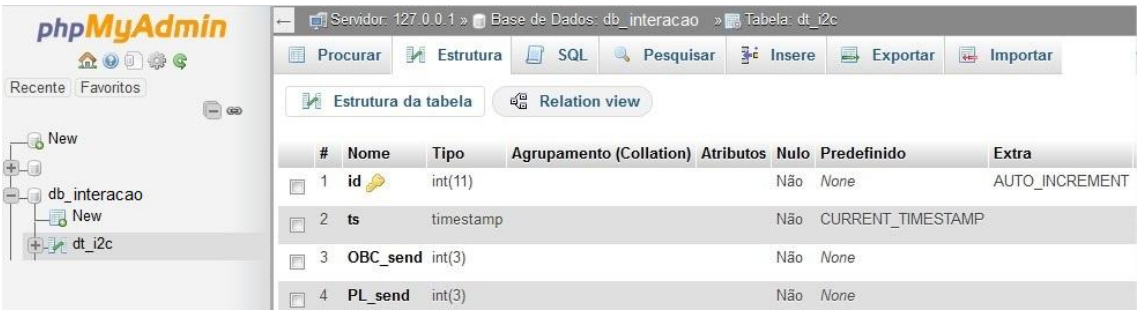
5.12 Base de dados Interação

O desenvolvimento do STAE utiliza um aplicativo computacional denominado Meu Administrador Pessoal de Páginas Caseiro (*Personal Home Page My Admin* – PhpMyAdmin), o qual possui recursos que facilitam o gerenciamento de base de dados por meio de uma linguagem denominada Minha Linguagem de Consulta (*My Structured Query Language* – MYSQL). A plataforma utilizada

neste trabalho para utilizar esses aplicativos é *XAMPP Control Painel versão 3.2.2*.

A base de dados que armazena as informações da interação entre os subsistemas comunicantes contém os seguintes dados: i) *id* (número sequencial de registro), ii) *timestamp* (data e hora de armazenamento da informação), iii) *OBC_send* (informação enviada pelo OBC, e iv) *PL_send* (informação enviada pela PL). A Figura 5.28 apresenta esta base de dados com uma tabela de dados denominada *dt_i2c*.

Figura 5.28 – Base de dados Interação.



As bibliotecas e comandos necessários para os subsistemas comunicantes acessarem a base de dados são implementados manualmente nos códigos gerados dos modelos desses subsistemas.

A Tabela 5.3 apresenta algumas linhas de comandos implementadas manualmente referentes ao protocolo I2C, ethernet e acesso a base de dados. Cada linha possui um comentário após duas barras (//) descrevendo o que será realizado.

Tabela 5.3 - Comandos para gravar dados na base MySQL.

```
// bibliotecas ethernet e I2C
#include <SPI.h>
#include <String.h>
#include <Ethernet.h>
#include <Wire.h>
const byte MY_ADDRESS = 0x3C; // endereço da PL
// Configuração para conexão com Arduino
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x9B, 0x36 };
// Endereço do Arduino
byte ip[] = { 192, 168, 1, 100 };
// Endereço do computador - IP
byte servidor[] = { 192, 168, 1, 101 };
// Porta do Arquivo como servidor 8090
// O bando de dados MySQL está na porta 80
EthernetServer server(8090);
EthernetClient cliente;
// Base de Dados db_nanosatc_br
// Tabela dt_i2c
// Comandos para interagir com a base de dados
String readString = String(50);
// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0; // will store last time LED was updated
// constants won't change :
const long interval = 5000;// interval at which to blink (milliseconds)
  Ethernet.begin(mac, ip); // inicializa acesso via ethernet
  Serial.begin(9600); // define velocidade da porta serial
void () {
  EthernetClient client = server.available(); // inicializa ethernet com cliente
}
void () {
//MYSQL
  EthernetClient client = server.available(); // acesso da ethernet com cliente
    if (cliente.connect(servidor, 80)) { // verifica conexão com a porta 80
      Serial.println("Conected Data Base"); // aviso de conexão com base de dados
      // arquivo php com comandos para gravar dados na base de dados
      cliente.print("GET /arduino/salvardadosArtigo2.php?");
      cliente.print("OBC_send=");
      cliente.print(ifaceOBC.TC); // grava informação enviada pelo OBC
      cliente.print("&PL_send=");
      cliente.print(read_I2C); // grava informação enviada pela SLP
      cliente.stop(); // termino da conexão
    }
    else {
      Serial.println("Connection fail - Data base"); // falha na conexão
      cliente.stop(); // termino da conexão
    }
} //FIM MYSQL
```

As linhas de comandos da Tabela 5.3 são implementadas no arquivo OBC.cpp.

A Tabela 5.4 apresenta as linhas de comandos de dois arquivos php: i) arquivo salvardados.php, o qual grava dados na base de dados, e ii) arquivo conecta.php, o qual realiza a conexão com a base de dados. Nas linhas de comandos está descrito após duas barras (//) o que será realizado.

Tabela 5.4 – Arquivo salvardados.php (esquerda) e conecta.php (direita).

salvardados.php

```
<?php
	include("conecta.php"); // acessa arquivo que realiza conexão com a base de dados
	$OBC_send = $_GET['OBC_send']; // lê conteúdo da variável OBC_send
	$PL_send = $_GET['PL_send']; // lê conteúdo da variável PL_send

	$sql_insert = "insert into dt_i2c (OBC_send,PL_send) values ('$OBC_send','$PL_send')"; // grava na tabela dt_i2c informações lidas

	mysql_query($sql_insert);
	if($sql_insert) // verifica se ocorreu erro
{
		echo "Salvo com sucesso";
	} else {
		echo "Ocorreu um erro";
	}
?>
```

conecta.php

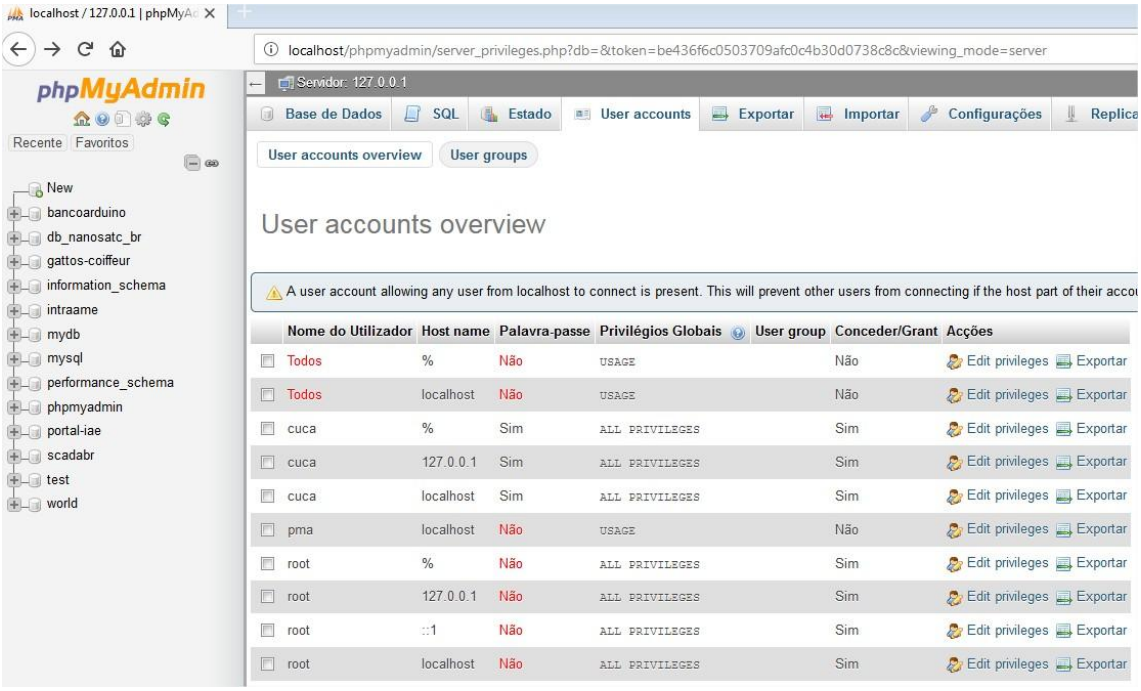
```
<?php
	$usuario = "root"; // verifica usuário e senha
	$senha = "";
	$host = "localhost";
	$conexao = mysql_connect($host,$usuario,$senha);
	$selecionabd = mysql_select_db('db_interacao', $conexao); // seleciona a base de dados db_interacao
	if($conexao) // verifica conexão
	{
		echo "Conectou com sucesso";
	} else {
		echo "Ocorreu um erro";
	}
?>
```

Na abordagem desenvolvida são utilizados esses arquivos configurados em php para acessar as bases de dados MySQL utilizando placas Arduino. Esses arquivos foram gravados na pasta C:\xampp\htdocs\arduino, sendo que C:\xampp é a pasta onde está instalada a plataforma XAMPP.

A plataforma XAMPP foi configurada da seguinte forma: abrir o *Control Painel*, selecionar *Config* do modulo Apache, selecionar o arquivo Apache (httpd.conf), configura a porta do apache na linha do comando *Listen* (no caso do STAE esse comando está na linha 58 e foi configurado *Listen* 80). No *Control Painel* do XAMPP está configurado a porta do Apache como 80,443 e do MySQL como 3306.

Por meio do *Control Panel* do XAMPP selecionar *Admin* do MySlq para executar o ambiente de desenvolvimento phpMyAdmin. Dentro do ambiente phpMyAdmin realizar uma alteração de privilégio do *root* para evitar bloqueio por permissão. Selecionar *New*, no *menu* superior selecionar *User accounts*, na conta *root* contento *Host name %* selecionar *Edit privileges* e selecionar *All PRIVILEGES* conforme apresentado na Figura 5.29.

Figura 5.29 – Alteração de privilégios de *root* no MySQL.



No *menu* superior dentro de *Edit privilegis* selecionar Base de Dados, selecionar Edit privilegis e selecionar *ALL PRIVILEGES*.

As execuções realizadas para acessar as bases de dados utilizando as placas Arduino são: i) conectar duas placas Arduino conforme padrão do protocolo de comunicação, ii) em uma das placas, conectar uma placa com acesso a *ethernet*, iii) conectar um equipamento que permita a conexão da placa *ethernet* do Arduino com a placa *ethernet* do computador ao qual as placas Arduino são conectadas (concentrador), e iv) identificar os endereços de Protocolo da Internet (*Internet Protocol – IP*) da placa Arduino e da placa de rede *ethernet* do computador. Esses endereços *IP*s são configurados no arquivo da Tabela 4.3 apresentado anteriormente.

Utilizar a *sketch WebClient* para identificar se está havendo conexão com a placa Ethernet. Essa *sketch* está disponível dentro do ambiente de desenvolvimento IDE do Arduino selecionando Arquivo/Exemplos/Ethernet/WebClient. Para identificar o número IP da placa com acesso a *Ethernet* conectada a placa Arduino foi utilizado a *sketch DhcpAddressPrinter.ino*.

5.13 Base de dados Estímulo

Os estímulos identificados são armazenados em uma base de dados denominada BD Estímulo. A tabela contendo esses dados é denominada *dt_est* sendo estruturada da seguinte forma: i) *id* (número sequencial de registro), ii) *timestamp* (data e hora da operação), iii) *nsub* (número do subsistema que envia o estímulo de falha [1 a 100 = OBC, 201 a 200 = uma determinada PL, 201 a 300 = outra determinada PL ...]), e iv) *est* (informação contendo o estímulo de falha). A Figura 5.30 apresenta essa base de dados.

Figura 5.30 – Base de dados Estímulo.

O objetivo é utilizar as informações desta base de dados para efetuar estímulos nos subsistemas sob teste e gerar casos de teste para observar se os resultados estão conforme as especificações dos requisitos de interação entre os subsistemas comunicantes.

5.14 Mecanismo Emulado de Falhas

O Mecanismo Emulador de Falhas (FEM) é uma placa Arduino configurada com o protocolo de comunicação do Cubesat configurada manualmente da seguinte forma: i) duas portas (SDA e SCL) da placa são configuradas para estarem conectadas a um subsistema, e ii) outras duas portas (outras SDA e SCL) estão configuradas para estarem conectadas ao outro subsistema. Desta forma o FEM é o interconector entre os subsistemas. A Tabela 5.5 apresenta uma codificação utilizada no FEM.

Tabela 5.5 - Codificação do FEM.

```
#include <SoftI2CMaster.h> // biblioteca para definir porta SCL e SDA
#include <Wire.h> // biblioteca I2C
#define SCL 6 // segunda porta SCL
#define SDA 7 // segunda porta SDA
// SoftI2CMaster é a biblioteca chamada para definir as portas SCL e SDA
SoftI2CMaster i2c = SoftI2CMaster(SCL,SDA,0);
#define DATA_SIZE 32 // tamanho da variável
const uint8_t SLV_ADDR = 0x3C; // endereço do esrcravo
const uint8_t MST_ADDR = 0x03; // endereço do escravo
byte x[DATA_SIZE],y[DATA_SIZE]; // tamanho da variável
unsigned int mst_msg=0,slv_resp=0,fault;
// injecao de bitflip
 byte bitflip(byte who, int where){
   //Serial.print("\n");
   //Serial.println(who,BIN);
   bitWrite(who,where,!bitRead(who,where));
   //Serial.println(who,BIN);
   return who;
 }
// provision faults
 byte provision(byte who){
   return who;
 }
// time related faults
 void timeout(int who){
   delay(who);
 }
void setup() {
  Serial.begin(9600); // velocidade da porta serial
  Wire.begin(SLV_ADDR); // inicia I2C
  i2c.begin();
  Wire.onReceive(callReceive); // chama função
  Wire.onRequest(callRequest); // chama função
  randomSeed(analogRead(0));
}
void loop(){
//  fault = random(0x7F);
}
```

```
void callReceive(int numBytes){
  mst_msg++;
  i2c.beginTransmission(SLV_ADDR); // inicia transmissão I2C
  for(int i=0;i<numBytes;i++){
    x[i]=Wire.read(); // lê I2C
    if(mst_msg==6){x[i]=bitflip(x[i],0);} // bitflip
     i2c.write(x[i]); // escreve I2C
   }
  i2c.endTransmission(); // fim transmissão I2C
  Serial.print("\n"); // envia informação para serial
  Serial.print("Write");
  Serial.print("\t");
  Serial.print(mst_msg);
  Serial.print("\t");
  Serial.print("Read");
  Serial.print("\t");
  Serial.print(slv_resp);
  if(mst_msg==6){
    Serial.print("\t");
    Serial.print("Bit-Flip on Writing from Master");
  }
}
void callRequest(){
  slv_resp++;
  i2c.requestFrom(SLV_ADDR,DATA_SIZE); // solicita informação do escravo
  delay(100000);
  for(int i=0;i<DATA_SIZE;i++){
     x[i]=i2c.read();
    if(slv_resp==2){x[i]=bitflip(x[i],7);} // bitflip
    Wire.write(x[i]); // escreve no I2C
  }
  Serial.print("\n"); // envia informação para serial
  Serial.print("Write");
  Serial.print("\t");
  Serial.print(mst_msg);
  Serial.print("\t");
  Serial.print("Read");
  Serial.print("\t");
  Serial.print(slv_resp);
  if(slv_resp==2){
    Serial.print("\t");
    Serial.print("Bit-Flip on Reading from Slave");
  }
}
```

Fonte: Batista et al. (2018).

Nas linhas de comandos está descrito o que é realizado após duas barras (//).

6 CONCLUSÃO

A abordagem desenvolvida utiliza um Sistema de Teste, o qual permite sua reutilização no processo de V&V de um missão espacial fazendo implementações nas fases de teste de validação, adicionando ou removendo os subsistemas do satélite no *loop*. Também existe a possibilidade do mesmo sistema de teste ser usado em diferentes satélites da mesma família. Os recursos propostos para implementar o protótipo deste sistema são *COTS*, eficazes e flexíveis o suficiente para testar diversas cargas úteis em missão nanosatélite que adotam o padrão CubeSat.

REFERÊNCIAS BIBLIOGRÁFICAS

AVIZIENIS, A., LAPRIE, J. C., RANDELL, B., LANDWEHR, C., TECHNICAL RESEARCH REPORT, Basic Concepts and Taxonomy of Dependable and Secure Computing, 2004.

BATISTA, C. L. G., MARTINS, E., MATTIELLO-FRANSCISCO, M. F., On the use of a Failure Emulator Mechanism at Nanosatellite Subsystems Integration Tests, 2018.

CONCEICAO, C., A., MONTECCHI, L., MATTIELLO-FRANCISCO, F., Método para remoção de falhas para sistemas intensivos em software a bordo de nanosatélites, 2018.

ECLIPSE, 2016, https://eclipse.org/home/index.php.

MCROBERTS, M., Arduino Básico, Novatec Editora Ltda. 2011, http://www.fema.com.br/arduino/wp-content/uploads/2014/08/arduino.pdf.

MYSQL, MySQL 5.1 Reference Manual. Disponível em: http://dev.mysql.com/doc/refman/5.1/en/index.html, 2013.

PHILIPS SEMICONDUCTORS: AN10216-01, I2C MANUAL, 2003.

PHPMYADMIN, Documentação do PhPMyAdmin, versão 3.2.0.1, 2009. Disponível em: http://localhost/phpmyadmin/Documentation.html.

PIMENTEL, A. R., Projeto de Software Usando a UML, Apostila para Curso de Projeto de Sistemas Orientado a Objetos Usando a UML, 2015.

SCHUCH, N., DURÃO, O., SILVA, M., MATTIELLO-FRANCISCO, F., silva, A. and NanosatC-BR Team, 'NANOSATC-BRSTATUS - A JOINT CUBESAT-BASED PROGRAM DEVELOPED BY INPE AND UFSM, 2017.

UPPAAL 4.0: Small Tutorial, 2009, http://www.uppaal.org/.

YAKINDU, 2017. Disponível em: https://www.itemis.com/en/yakindu/state-machine/documentat