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

sid.inpe.br/mtc-m21c/2019/01.16.14.41-TDI

INJEÇÃO DE FALHAS EM TESTES DE INTEGRAÇÃO DE SUSBSISTEMAS APLICADA A NANOSSATÉLITES

Carlos Leandro Gomes Batista

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pela Dra. Maria de Fátima Mattiello Francisco, aprovada em 30 de janeiro de 2019.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3SJ7TLP>

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/2019/01.16.14.41-TDI

INJEÇÃO DE FALHAS EM TESTES DE INTEGRAÇÃO DE SUSBSISTEMAS APLICADA A NANOSSATÉLITES

Carlos Leandro Gomes Batista

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Engenharia e Gerenciamento de Sistemas Espaciais, orientada pela Dra. Maria de Fátima Mattiello Francisco, aprovada em 30 de janeiro de 2019.

URL do documento original:
<http://urlib.net/8JMKD3MGP3W34R/3SJ7TLP>

INPE
São José dos Campos
2019

Dados Internacionais de Catalogação na Publicação (CIP)

Batista, Carlos Leandro Gomes.

Ba32i Injeção de falhas em testes de integração de susbsistemas aplicada a nanossatélites / Carlos Leandro Gomes Batista. – São José dos Campos : INPE, 2019.

xxii + 64 p. ; (sid.inpe.br/mtc-m21c/2019/01.16.14.41-TDI)

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

Orientadora : Dra. Maria de Fátima Mattiello Francisco.

1. Verificação e validação. 2. Nanossatélite. 3. CubeSat. 4. Robustez. 5. Injeção de falhas. I.Título.

CDU 629.783-022.532

Aluno (a): *Carlos Leandro Gomes Batista*

Título: "INJEÇÃO DE FALHAS EM TESTES DE INTEGRAÇÃO DE SUBSISTEMAS APLICADA A NANOSSATÉLITES"

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

(X) Aprovado ( ) Reprovado

Dr. Ronaldo Arias

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

( ) Participação por Video - Conferência

(X) Aprovado ( ) Reprovado

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 janeiro de 2019

"Para voltar do mundo à face clara

Nessa vereda escusa penetramos:

De nós nenhum de repousar cuidara.

Virgílio e eu, logo após, nos elevamos

Té que do ledo céu as coisas belas

Por circular aberta divisamos:

Saindo a ver tornamos as estrelas."

Dante Alighieri
"A Divina Comédia: Inferno, Canto XXXIV:132-139".

Aos meus pais, ***Batista*** *e* ***Regina****, e ao meu irmão* ***Leonardo****.*

AGRADECIMENTOS

Agradeço à Professora Maria de Fátima Mattiello-Francisco que, não apenas como orientadora, me guiava e manteve firme a confiança em mim a todo momento.

Ao Sr. Manoel Jozeanne Mafra de Carvalho e toda a equipe do Centro Regional Nordeste, eternamente minha segunda casa.

À Professora Eliane Martins e ao Anderson Weller, ambos da UniCamp, os quais, sem eles, não teria sido possível a publicação do artigo derivado deste trabalho.

À minha família, o porto seguro mesmo que longe.

À minha companheira, Mainara, a vela que ilumina este mundo assombrado por demônios.

À minha amiga, Williane, que fora a primeira pessoa a me apoiar nesta empreitada.

Ao São José Rugby Clube, mesmo a revelia de alguns, que me acolheram de braços abertos ao chegar em São José dos Campos.

A CAPES e ao INPE pelo financiamento sem o qual estes anos de aprendizado seriam impossíveis.

E a todos aqueles que de alguma forma contribuíram para a conclusão deste trabalho.

Meu muito obrigado!

RESUMO

O tempo curto de desenvolvimento e o baixo custo das missões de nanossatélites com o uso do padrão CubeSat tem motivado o crescimento do número de lançamentos desses objetos ao espaço na última década ao redor do mundo. No entanto, rapidez e pequenos orçamentos não são garantia de sucesso de missão espacial. Deficiências na adoção das boas práticas de projeto, montagem e testes têm sido apontadas como uma das grandes causas para as ativas falhas nas missões com nanossatélites. Esforços no uso de técnicas eficientes de verificação e validação são necessárias. Dado a crescente aplicação de missões de nanossatélites para qualificação de novas tecnologias em órbita, comportamentos errôneos desses subsistemas são esperados. Porém, tal mal funcionamento não pode representar um risco para a missão como um todo. Robustez é uma propriedade importante de um sistema crítico reativo e não propriamente explorado no padrão CubeSats. Aspectos de comportamento entre os subsistemas comunicantes devem ser verificados. A sistematização de testes para nanossatélites baseados no padrão CubeSat deve ser apoiada por ferramentas compatíveis de modo a reduzir o tempo de desenvolvimento no que diz respeito ao tempo consumido nas atividades de verificação e validação. Neste trabalho é apresentado o *framework* de um Mecanismo Emulador de Defeitos, FEM, para testes de robustez de subsistemas intensivos em software interoperantes a bordo de um nanossatélite. O FEM atua no canal de comunicação sendo parte da bancada de testes de integração em duas fases do projeto do nanossatélite: (i) especificação dos requisitos de robustez utilizando *model in the loop*, MIL e (ii) validação da robustez utilizando *hardware in the loop*, HIL. Os aspectos arquiteturais do FEM apoiam sua instanciação para qualquer canal de comunicação presente no padrão CubeSat. Como exemplo, o FEM foi instanciado para o canal de comunicação I2C para apoiar os testes realizados no NanoSatC-BR2. O NanoSatC-BR2 é uma missão científica baseada em um CubeSat em desenvolvimento e integração no Instituto Nacional de Pesquisas Espaciais, INPE, que utiliza o canal de comunicação I^2C para a interação entre as cargas úteis e o subsistema do computador de bordo (OBC). Este protótipo do FEM foi utilizado para apoiar aos testes de integração do OBC com cada carga útil em um cenário MIL visando a antecipação de requisitos de robustez durante o ciclo de desenvolvimento do nanossatélite.

Palavras-chave: Verificação e Validação. Nanossatélite. CubeSat. Robustez. Injeção de Falhas. Testes.

FAULT INJECTION IN SUBSYSTEMS INTEGRATION TESTS APPLIED TO NANOSATELLITES

ABSTRACT

The short development time and low cost of nanosatellite missions using CubeSat standard has motivated the increasing number of launches of these objects to the space during the last decade around the world. But, velocity and low budgets do not represent success on space missions. Lack on good design, assembly and tests practices have been pointed as one of the great causes for active failures on nanosatellite missions. Efforts on the use of efficient verification and validation techniques are needed. Faulty beahavior of these systems are expected as the nanosatellite missions applications grows to qualify new technologies in orbit. Such malfuction shall not represent a risk to the whole mission. Robustness is an important propriety of reactive critical systems and not properly exploited in CubeSat standardization. Behavior aspects of the communicating subsystems on the use of their interfaces shall be verified. The test systematization of CubeSat-based nanosatellites supported by proper tools is necessary to reduce the mission development cycle in terms of the time consumed by the verification and validation activities. In this work we present a failure emulator mechanism framework, named FEM, for robustness testing of interoperable software-intensive subsystems onboard nanosatellite. FEM acts in the communication channel being part of the integration test workbench in two phases of nanosatellite design: (i) robustness requirement specification using model in the loop (MIL) and (ii) robustness validation using hardware in the loop (HIL). The architectural aspects of the proposed FEM framework support its instantiation to any communication channel of the CubeSat standard. As an example, FEM prototype was instantiated to I^2C communication channel to support NanosatC-BR2 testing. NanosatC-BR2 is a Cubesat based scientific mission, under development and integration at Brazilian Institute for Space Research (INPE), which uses I^2C communication channel for its payloads interactions with the On-Board Data Handling computer subsystem (OBC). FEM prototype was used to support OBC integration testing with each payload subsystem at MIL scenario aiming at anticipating the robustness requirement verification on the development lifecycle.

Keywords: Verification and Validation. Nanosatellite. CubeSat. Robustness. Fault Injection. Testing.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

LISTA DE ABREVIATURAS E SIGLAS

EEVL	–	*Evolved Expendable Launch Vehicle*
ESPA	–	*Evolved Secondary Payload Adapter*
PPOD	–	*Poly-Picosatellite Orbital Deployer*
V&V	–	Verificação & Validação
COTS	–	*Commercial-of-the-shelf*
API	–	*Application programming interface*
DSR	–	*Design Science Research*
FEM	–	*Failure Emulator Mechanism*
INCOSE	–	*International Council on Systems Engineering*
SSWG	–	*Space Systems Work Group*
MarCo	–	Missão *Mars Cube One*
SUT	–	*System Under Test*
SEE	–	*Single Event Effects*
SEU	–	*Single Event Upset*
SEL	–	*Single Event Latchup*
SEB	–	*Sngle Event Burnout*
SEGB	–	*Single Event Gate Burnout*
FPGA	–	*Field Programmable Gate Array*
VHDL	–	*VHSIC Hardware Description Language*
SFI	–	*Software Fault Injection*
SWFI	–	*Software-Implemented Fault Injection*
MIL	–	*Model-in-the-Loop*
HIL	–	*Hardware-in-the-Loop*
MBT	–	*Model-Based Testing*
SiS	–	Sistemas Intensivos em Software
PCO	–	Ponto de Controlabilidade e Observabilidade
TS	–	*Test System*
ECSS	–	*European Cooperation for Space Standardization*
UFSM	–	Universidade Federal de Santa Maria
ASIC	–	*Application-Specific Integrated Circuit*
OBDH	–	*On Board Data Handling*
OBC	–	*On Board Computer*
ISIS	–	*Innovative Solutions In Space*
I^2C	–	*Inter-Integrated Circuits*
SDA	–	*Serial Data Bus*
SCL	–	*Serial Clock Bus*
TWI	–	*Two Wire Interface*
PLD	–	*Payload*
MOSFET	–	*Metal-Oxide-Semiconductor Field Effect Transistor*
SLP	–	Sonda de Langmuir

SUMÁRIO

Pág.

1 INTRODUÇÃO

Nos últimos vinte anos, um aumento significativo no número de nanossatélites desenvolvidos e lançados em órbita tem sido constatado. A última década mostrou ao mundo as capacidades que a indústria aeroespacial pode alcançar com os nanossatélites (POGHOSYAN; GOLKAR, 2016; NATIONAL ACADEMIES OF SCIENCES ENGINEERING AND MEDICINE et al., 2016; SELVA; KREJCI, 2012; MARIÑO et al., 2017; MOTA, 2017). Em especial, o padrão CubeSat, idealizado no início dos anos 2000, pelos professores Jordi Puig-Suari, da *California Polytech State University*, e Bob Twigs, da *Stanford University*, viabilizou a inserção da academia e de empresas emergentes no setor com acesso ao espaço a baixo custo (TWIGGS et al., 2001; MEHRPARVAR et al., 2014).

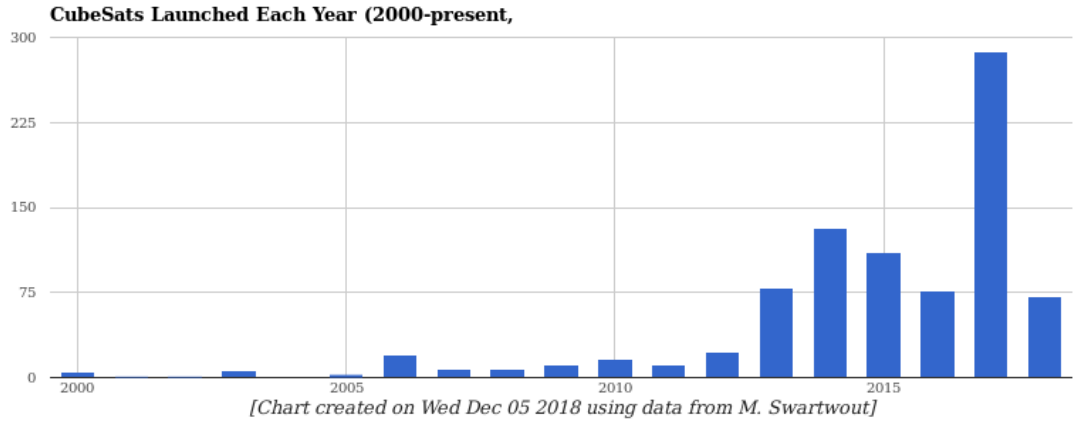
A miniaturização da eletrônica, frequentes e baratas oportunidades de lançamentos por empresas emergentes, oportunidades de carona em foguetes tradicionais usando EEVL (*Evolved Expendable Launch Vehicle*) e ESPA (*Evolved Secondary Payload Adapter*), mecanismos de *deployment* para CubeSats usando PPOD (*Poly-Picosatellite Orbital Deployer*) ou os *NanoRacks*, na Estação Espacial Internacional, além do aumento na disponibilidade de estações de solo de recepção são alguns fatores que corroboram ao sucesso destes minúsculos satélites (NAG, 2017).

De acordo com uma estimativa do site *Nanosatellite & CubeSat Database*, mais de quinhentos desses satélites com até 10 kg já estão programados para lançamentos até o ano de 2020, sendo, até o momento, quase mil já lançados e quase noventa por cento destes são do padrão em cubos (KULU, 2018).

A Figura 1.1 mostra os lançamentos já realizados de acordo com a base de dados do professor Swartwout (2018a). O que se pode observar desse gráfico é o aumento significativo de missões de nanossatélites baseadas em CubeSats ao redor do mundo, principalmente nos últimos anos.

Na América Latina, em especial, no Brasil, não foi diferente, o boom dos projetos de nanossatélites é uma realidade. Um grande número de missões de nanossatélites emergiu nos últimos anos: CONASAT (CARVALHO et al., 2013), Serpens (FIGUEIRÓ, 2014), AESP-14 (BÜRGER et al., 2013), NanoSatC-BR (SCHUCH et al., 2017), ITASAT (CARRARA et al., 2017), Suchai (DIAZ et al., 2016), UbatubaSat (SANTOS et al., 2011) entre outros. Até mesmo o plano diretor do INPE contempla o lançamento de, ao menos dez, experimentos científicos e tecnológicos em plataformas baseadas em nanossatélites até o ano de 2019 (INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS,

Figura 1.1 - Lançamentos de Nanossatélites até o ano de 2018.



FONTE: Swartwout (2018a)

2016).

Desenvolvimento rápido, baixo custo, curto tempo de vida, tudo isso contribui para o sucesso promocional desses satélites. No entanto, o que está sendo sacrificado para que essas missões sejam tão rápidas, tão baratas? Seria possível desenvolver missões baseadas no padrão CubeSats com tempo de vida maior? Ou mesmo com propósito operacional ou comercial de prestação de serviços?

Essas questões foram abordadas em estudos recentes (SWARTWOUT, 2016; SWARTWOUT, 2018b) demonstrando, de forma geral, que missões de nanossatélites que utilizam o padrão CubeSat possuem uma taxa de sucesso da ordem de cinquenta por cento, com melhores números, cerca de noventa por cento de sucesso em suas missões, por agências espaciais e empresas maiores e mais tradicionais no ramo. No outro lado da moeda, desenvolvedores menores, entusiastas do ramo espacial e pequenos programas espaciais alcançam taxas de sucesso baixas, na casa dos quarenta por cento. Sua conclusão associa esses dados a uma deficiência de bons processos de Verificação & Validação - V&V - durante as fases de desenvolvimento, testes e integração desses pequenos projetos espaciais.

Similar ao que aconteceu com a filosofia "*faster, cheaper and better*"(mais rápido, mais barato e melhor) nos anos 1990, projetos envolvendo nanossatélites foca no uso de componentes e sistemas de prateleira (*COTS – Commercial off-the-shelf*). E os resultados demonstram que sem um processo válido e compreensivo, "mais rápido e mais barato"não garante automaticamente a vertente do "melhor". No entanto, um processo construído inteligentemente pode elevar as taxas de sucesso nesses projetos

de forma rápida, barata e melhor, efetivamente (FORSBERG; CO-PRINCIPALS, 1999).

Processos customizados são comumente utilizados em projetos envolvendo nanossatélites (KASLOW et al., 2018; KASLOW et al., 2017; KASLOW et al., 2015) sem os quais a filosofia do "rápido e barato"torna-se inviável. Mesmo assim, para atingir o ramo do "melhor"nesses projetos, um processo bem definido de verificação e validação é necessário, uma vez que erros identificados mais cedo são mais baratos de se corrigir prevenindo muito do retrabalho (ANDO et al., 2015).

Devido ao seu curto tempo de desenvolvimento, missões baseadas em CubeSats usualmente focam em requisitos e restrições associadas ao lançamento reduzindo o processo de V&V a apenas testes de aceitação. Testes de integração são uma importante atividade do processo de V&V, onde se objetiva identificar problemas na interoperabilidade entre os sistemas comunicantes. Além de perda de dados, falhas na interação entre dois subsistemas comunicantes intensivos em software também podem levar a perda da missão. Por exemplo, a vulnerabilidade de um subsistema ao mal funcionamento do outro no uso de recursos compartilhados. Robustez, segundo IEEE (1990), é uma propriedade crítica em sistemas reativos que permite avaliar a vulnerabilidade do sistema a entradas inesperadas. Com o objetivo de aumentar a confiança do sistema como um todo, requisitos de robustez devem ser postos em pauta no projeto de uma missão.

Neste contexto, a lógica de injeção de falhas é comumente usada nos chamados testes de estresse e considerada de importante contribuição no desenvolvimento de *software* robusto. Testes de robustez com injeção de falhas são usados para verificar vulnerabilidades em interfaces de comunicação como protocolos, parâmetros de linhas de comando e APIs (VOAS, 1997; KAKSONEN, 2001).

O desafio é como especificar e testar esses requisitos de robustez.

1.1 Objetivo

O objetivo desta dissertação é apresentar um *framework* para um injetor de falhas que busque auxiliar as questões referentes à verificação de requisitos de robustez e de interoperabilidade em testes de integração de subsistemas de nanossatélites, sua proposição, implementação e aplicação. O *framework* do injetor irá interagir com os subsistemas sob teste via interface de comunicação de forma a interceptar as mensagens trocadas e atuar, ou não, emulando possíveis defeitos através de modificações (falhas) nos pacotes de mensagens.

O injetor, então, visa apoiar os testes de integração viabilizando a realização de casos de teste que vislumbram a presença de falhas especificadas no âmbito da interação entre dois subsistemas comunicantes.

1.2 Metodologia

A metodologia adotada nesta dissertação busca responder a duas perguntas de pesquisa:

a) Como verificar o cumprimento dos requisitos de interoperabilidade e robustez em sistemas reais embarcados em nanossatélites?

b) Como associar uma solução rápida e barata para melhorar o processos de desenvolvimento, integração e testes em nanossatélites?

Assim, pensando a cerca destas perguntas, adotou-se uma abordagem que, para Hevner et al. (2008), é fundamentalmente um paradigma para a solução de problemas, a *Design Science Research – DSR*.

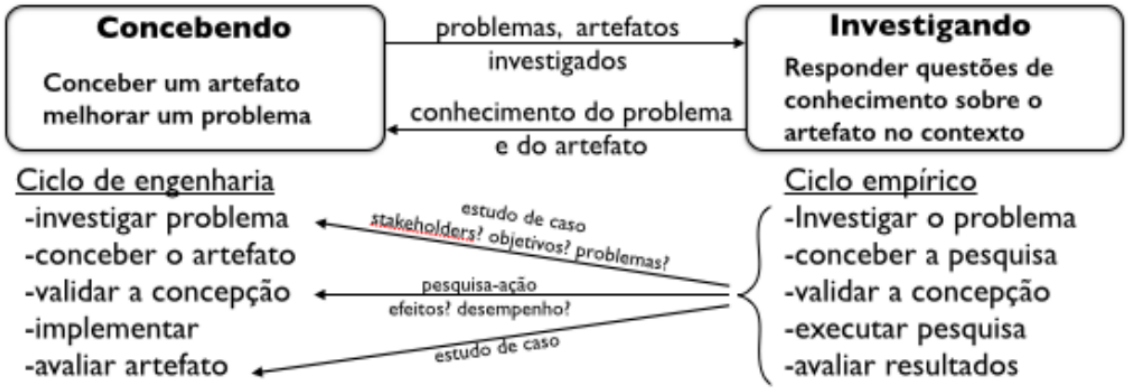
A DSR se vale da criação de artefatos – construtores, modelos, métodos ou instanciações – que são aplicados, a partir do conhecimento e compreensão do problema a ser resolvido, verificando-os, validando-os e avaliando-os na medida que são construídos e utilizados (BAX, 2013).

As diretivas da DSR, segundo Hevner (2007) são: (i) o projeto de pesquisa produz artefatos; (ii) problemas devem ser potencialmente relevantes e são dependentes do contexto; (iii) propostas de solução devem ser validadas; (iv) propostas devem produzir teorias que melhorem as práticas; (v) devem ser validadas com métodos científicos; (vi) a pesquisa deve iterar entre concepção e validação e; (vii) os resultados devem ser comunicados/publicados.

Wieringa e Heerkens (2006), Wieringa (2009) identifica dois ciclos a serem executados, enfatizando a conexão entre conhecimento e prática (Figura 1.2). Um ciclo de engenharia que investiga o problema, concebe a solução, verifica, implementa e avalia. Um ciclo empírico, ou de conhecimento, que investiga o problema, concebe a pesquisa, valida, executa e avalia os resultados.

Isto posto, as questões de pesquisa supracitadas norteiam a investigação do problema e concebem a pesquisa ao mesmo tempo que validam a concepção do injetor de falhas (artefato), avaliando os resultados obtidos a partir da implementação do protótipo

Figura 1.2 - Ciclos da DSR



FONTE: Bax (2013)

instanciado, neste trabalho, para os testes de integração das cargas úteis do programa NanoSatC-BR2.

E, como prediz uma das diretrizes da DSR, os resultados desta dissertação resultaram em uma publicação revisada por pares (BATISTA et al., 2019).

1.3 Estrutura do Texto

O texto desta dissertação está estruturado da seguinte forma:

- Capítulo 2: Fundamentação Teórica, uma breve revisão de alguns conceitos abordados neste trabalho que discorrem quanto às questões levantadas no Capítulo 1.
- Capítulo 3: Injeção de Falhas, conceitos sobre falha, erro e defeito, o que significa injetar falhas e como essa abordagem pode ser utilizada.
- Capítulo 4: Mecanismo Emulador de Defeitos, FEM, descrição do *framework* para a injeção de falhas e apoio aos testes de integração de subsistemas.
- Capítulo 5: Caso de Estudo, NanoSatC-BR2, descrição da missão NanoSatC-BR2 utilizada para validar o *framework* instanciando o FEM para a interface de comunicação da plataforma.
- Capítulo 6: Uso do FEM, em cenário de *Model-in-the-loop*, descreve a geração dos modelos de interoperabilidade dos subsistemas sob teste assim

como apresenta os resultados dos casos de teste executados e discussões dos resultados.

- Capítulo 7: Conclusões, fechamento do trabalho com considerações acerca do uso do *framework* assim como possibilidades de melhoria no projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Os parágrafos a seguir buscam explanar os fundamentos teóricos obtidos no estudo bibliográfico guiado pelas questões de pesquisa mencionadas no Capítulo 1.

2.1 Nanossatélites

Dentro da categoria de pequenos satélites (massa menor que $500kg$), encontram-se os nanossatélites (massa de $1kg$ a $10kg$). Segundo Konecny (2004) a classificação de acordo com a massa segue a distribuição da Tabela 2.1.

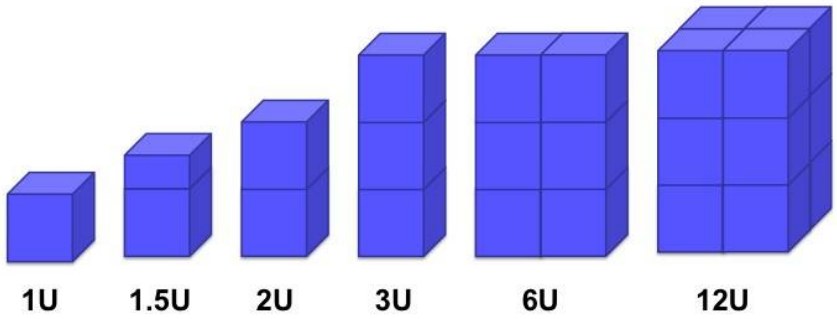
Quando falamos em nanossatélites não há como não nos depararmos com os CubeSats. Como já mencionado no Capítulo 1, o CubeSat é um padrão desenvolvido pelos professores Jordi Puig-Suari e Robert Twigs na qual se uniformiza o tamanho do que passa a ser chamado de Unidade, U. Uma unidade, ou convencionado 1U, equivale a um cubo de $10cm$ de aresta, ou $10 \times 10 \times 10cm^3$ de volume.

A Figura 2.1 apresenta algumas das configurações mais comuns utilizadas para CubeSats. Vale notar que cada Unidade de um CubeSat pode possuir entre $750g$ a $1,5kg$ a depender dos equipamentos embarcados .Então, estruturas com 12U po-

Tabela 2.1 - Classificação de Pequenos Satélites segundo sua massa

Categoria	Massa (em kg)
Minissatélite	$100 - 500$
Microssatélite	$10 - 100$
Nanossatélite	$1 - 10$
Picossatélite	$0.1 - 1$
Fentomsatélite	$< 0,1$

Figura 2.1 - Configurações de CubeSats



FONTE: Mabrouk (2015)

deriam ser classificadas como microssatélites, levando a conclusão que nem todo CubeSat é um nanossatélite.

Diversas iniciativas vem ocorrendo ao redor do mundo no interesse de melhor aproveitar o que os nano/CubeSats têm a oferecer. Iniciativas da INCOSE (*International Council on Systems Enginnering*) com o seu *Space Systems Work Group – SSWG* – buscam elaborar um modelo de referência para o desenvolvimento de CubeSats (`https://bit.ly/2QEpKVT`) (KASLOW et al., 2018), assim fomentando parcerias com órgãos como a OMG (*Object Management Group*), responsável por padronizações como UML e SysML, a fim de buscar avanços na engenharia de sistemas de CubeSats (`https://bit.ly/2A2E4x9`).

Uma iniciativa que chama a atenção é a empresa PlanetLabs (`https://www.planet.com/`) que se utilizando de micro e nanossatélites (uma constelação de mais de cento e cinquenta CubeSats de 3U) vem atuando no mercado de imageamento terrestre fornecendo dados de observação terrestre 24h por dia, sete dias por semana.

No entanto, nada se compara em termos de capacidade desses pequenos cubos espaciais à recente missão MarCO, *Mars Cube One*, do Laboratório de Propulsão a Jato da NASA (KLESH; KRAJEWSKI, 2018; KLESH; KRAJEWSKI, 2015).

Os nanossatélites, MarCO-A ("Eve") e MarCO-B ("Wall-E") foram os primeiros de seu tipo a ir ao chamado *deep space*, espaço profundo, ou seja, fora da órbita do planeta Terra (CALTECH, 2018). Quando do seu lançamento e abertura de suas antenas (HODGES et al., 2017), em maio de 2018, já foram capazes de registrar fotos tanto de Marte (GOOD, 2018a) como uma singela homenagem à Voyager-1 em sua foto do pálido ponto azul (GOOD, 2018b). Cabe o detalhe que a Voyager-1 possui mais de oitocentos quilos, e essas novas fotos foram tiradas por "Wall-E", um nanossatélite de pouco mais de sete quilos.

Missões como a MarCO e iniciativas comerciais como a da PlanetLabs, requerem confiabilidade e robustez das partes comunicantes da missão que, antes, ao se imaginar satélites de cunho universitário, não se esperava deles.

Como podemos melhorar a qualidade, mantendo rapidez e baixo custo, tão intrínsecos aos projetos de nanossatélites, sabendo-se que as funcionalidades do satélites realizadas por software embarcado são crescentes?

2.2 Interoperabilidade e Robustez

Considerando o crescimento no uso das missões de nanossatélites para a qualificação de novas tecnologias em órbita e exploração espacial (KLESH; KRAJEWSKI, 2018; ZUFFADA et al., 2018), a presença de falhas nas novas tecnologias embarcadas nessas missões deve ser considerada. Porém, tal comportamento defeituoso não pode representar um risco para toda a missão.

Teste é uma das técnicas mais conhecidas de verificar, ou mesmo, validar um produto/sistema, quer em bancada, por meio de simulações e até mesmo em ambiente real de uso.

Práticas rigorosas de V&V normalmente consomem grandes quantias em dinheiro, esforço e tempo (LEWIS, 1992), tudo que corre na contra mão da filosofia de desenvolvimento de Nano/CubeSats.

Assim sendo, fica clara a deficiência existente em projetos envolvendo nanossatélites, principalmente, quando realizados por equipes pequenas ou de baixa *expertise* no ramo, no que tange aos testes de integração dos subsistemas comunicantes do satélite visto que incertezas na qualidade das partes são inerentes a projetos de desenvolvimento descentralizado.

Nesse contexto, **interoperabilidade** – a capacidade de duas implementações comunicarem-se, ou interagir, corretamente, provendo os serviços como especificado durante essa interação (DESMOULIN; VIHO, 2007) – e **robustez** – o nível com o qual um sistema/subsistema ou componente pode operar corretamente na presença de entradas inválidas ou condições danosas do ambiente (IEEE, 1990) – são dois conceitos fundamentais.

Na engenharia de software, aspectos de interoperabilidade e robustez são requisitos não funcionais inerentes a sistemas críticos reativos. O objetivo dos testes de interoperabilidade é verificar se dois sistemas comunicantes, cada um referido a partir de agora como Sistema Sob Teste (SUT), satisfatoriamente provêm os serviços descritos em suas especificações. Sob o ponto de vista da interoperabilidade de um sistema intensivo em software, a integração de um sistema (SUT1) envolve validar o comportamento de um software embarcado integrado com o hardware alvo e suas interações com outro sistema (SUT2). Usualmente, os subsistemas comunicantes operam segundo a lógica de mestre-escravo na qual os SUT são estimulados pelo sistema de testes (TS) via interface unilateral associada ao mestre (DESMOULIN; VIHO, 2007).

No tocante à robustez, a integração de um SUT visa validar seu comportamento de acordo ao especificado mediante condições adversas, sejam internas, do ambiente ou de outros subsistemas interoperantes com este (MATTIELLO-FRANCISCO, 2009).

A análise desses dois aspectos, interoperabilidade e robustez, é crucial na integração dos subsistemas de um satélite, visto que o mesmo é um sistema complexo que abarca um número significativo de funções implementadas por software, referidos nessa dissertação por subsistemas intensivos em software, que operam em um ambiente ativamente perigoso, e com requisitos de confiabilidade extremamente contundentes.

3 INJEÇÃO DE FALHAS

A técnica de injeção de falhas nasce nos anos 90, como um elemento chave da análise de dependabilidade (ARLAT et al., 1990) e parte fundamental do chamado ciclo *fault-error-failure*, que discutiremos mais a frente.

Os primeiros experimentos de injeção de falhas envolviam exclusivamente *hardware* e compreendiam nada mais que curto-circuitos em placas e observação de seus efeitos no sistema. Testando, primeiramente, a dependabilidade dos sistemas de *hardware*. Mais tarde sistemas de teste especializados foram desenvolvidos para estender essa técnica, tais como dispositivos capazes de bombardear áreas específicas de circuitos com radiação pesada. Mais a frente, técnicas utilizando *software* para a indução de falhas foram introduzidas e aspectos da injeção de falhas puderam ser úteis na análise e avaliação de robustez de sistemas intensivos em *software* (HSUEH et al., 1997; CARREIRA et al., 1999).

Neste escopo, a injeção de falhas é uma abordagem que se encaixa bem para a identificação e compreensão de eventos defeituosos e útil para observar o comportamento do sistema na presença de falhas nas fases de prototipagem, integração e operação (CLARK; PRADHAN, 1995). Adicionamos, ainda, a aplicabilidade desse método em testes de interoperabilidade e robustez, como colocado por Mattiello-Francisco (2009) e referido em trabalhos subsequentes, visando o atendimento de requisitos (MATTIELLO-FRANCISCO et al., 2012; WELLER et al., 2015).

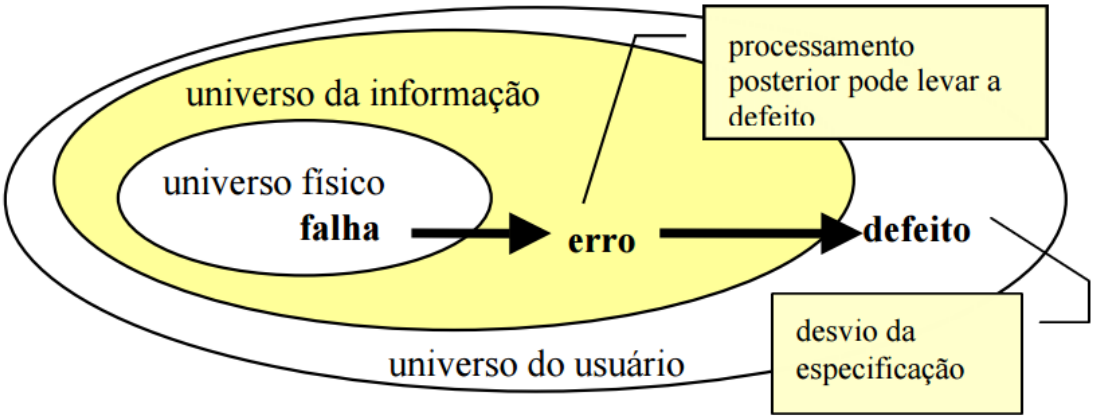
Injeção de Falhas na inserção de falhas em um sistema com o propósito de emular possíveis falhas (ARLAT et al., 1990), permitindo testar as capacidades do sistema na detecção, isolação, reconfiguração e recuperação de falhas/erros de um sistema (HSUEH et al., 1997).

Nota-se aqui que alguns conceitos podem ser objeto de confusão quando traduzidos da língua inglesa – *fault, error and failure*.

Serão consideradas as seguintes definições: uma falha (*fault*) causa um erro (*error*), um estado interno não esperado/especificado, o que leva a um defeito (*failure*), *crash* do sistema ou possível degradação dos serviços ao usuário, como definido por Avizienis et al. (2004) e apresentado na Figura 3.1.

Falhas estão associadas ao universo unitário, erros ao universo da informação e defeitos ao universo do usuário. Assim um conversor A/D, que apresenta uma falha devido a um *Single Event Upset* em um de seus bits (falha no universo físico),

Figura 3.1 - Definições para Falha, Erro e Defeito



FONTE: Weber (2003)

pode provocar uma interpretação errada da informação transferida ao subsistema de controle de temperatura (erro no universo da informação) e como resultado o sistema pode negar o aquecimento das baterias causando degradação do serviço do satélite (defeito no universo do usuário) (WEBER, 2003).

Assim, falhas podem ser divididas em duas diferentes categorias: falhas de *software* e falhas de *hardware*.

- Falhas de *Hardware*: falhas apresentadas em caráter físico no sistema. Por exemplo, um *bit-flip* causado por excessiva exposição à radiação. Não há estímulo para provocar esse tipo de falha em um laboratório pelo fato de, normalmente, possuir caráter destrutivo do equipamento testado.

- Falhas de *Software*: levam em consideração problemas de codificação e interpretação de sinais por parte de um programa ou código embarcado. *A priori*, são de fácil implementação visto que partem do pressuposto que podem ser viabilizadas pelo próprio desenvolvedor alterando linhas de código.

Em projetos de nanosatélites identifica-se que a injeção de falhas tanto de hardware quanto de software em nível de teste de conformidade, aplicada no ambiente do desenvolvedor do subsistema, é praticamente inviável devido os baixos orçamentos e recursos laboratoriais de uma missão baseada em CubeSat. Apesar de os subsistemas serem desenvolvidos de forma distribuída, é comum o próprio desenvolvedor estar

encarregado de testá-lo e, muitas vezes não ter tempo hábil para tal feito.

Soma-se às limitações descritas, o fato de não serem permitidas alterações nos subsistemas depois da sua integração na plataforma do satélite.

Assim, a mitigação de defeitos e a geração de contra-medidas se tornam cada vez mais difíceis de serem alcançadas no ambiente de desenvolvimento de subsistemas embarcados em nanosatélites, uma vez que restrições como *time-to-market*, reuso constante e o uso de componentes COTS são comuns nestes projetos, mesmo em sistemas críticos da missão.

Entretanto, injetar falhas é uma técnica efetiva considerada em diferentes metodologias e processos de V&V, especialmente para avaliação de robustez. Durante o desenvolvimento de *safety-critical software* é recomendado a injeção de falhas quando o sistema adota contra-medidas com o objetivo de detectar condições anormais e de forma a manter um estado normal de operação. Algumas normas de *safety* (e.g. ISO 26262 e NASA *Software Safety Guidebook*) mencionam a injeção de falhas entre os métodos para testes de integração e para verificar o comportamento do sistema quando colocado junto a *software* desenvolvido por terceiros (O'CONNOR, 2004; COTRONEO; NATELLA, 2013).

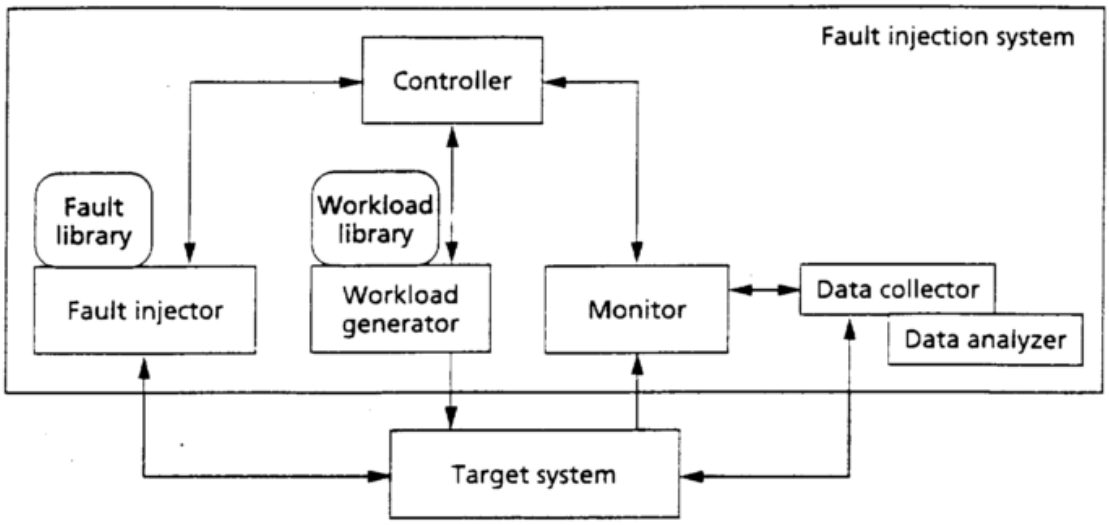
Fundamentado na uso da técnica de injeção de falhas para testes de integração de subsistemas intensivos em software comunicantes, o presente estudo foi desenvolvido.

A Figura 3.2 mostra o ambiente genérico para a injeção de falhas, normalmente composto por um alvo (*target system*), injetor (*fault injector*) e biblioteca (Ifault library) de falhas, biblioteca e gerador de carga (*workload library* e *workload generator*), monitor, coletor e analizador de dados (*data collector* e *data analyzer*), além do próprio controlador da injeção (*controller*).

O método e implementação da injeção são definidos pela necessidade do teste. Falhas do tipo *stuck-at* ou SEE podem ser melhor representadas por falhas de *hardware*. No entanto, se o interesse do teste é corrupção de dados, um método de injeção de falhas de *software* tem melhor resposta. Algumas falhas, como *bit-blip* em posições de memória, podem ser representados pelos dois métodos, ficando a cargo do desenvolvedor/testador a avaliação de certas medida (custo, tempo, repetibilidade, etc) para a escolha do mais apropriado.

Mesmo sendo consideradas em duas categorias diferentes, falhas de *software* e *hardware* podem se propagar, sendo possível uma falha em meio físico (*hardware faults*)

Figura 3.2 - Ambiente de Injeção de Falhas



FONTE:Hsueh et al. (1997)

se propagar para o universo de programa *software* (NATELLA et al., 2016), demonstrado na Figura 3.3.

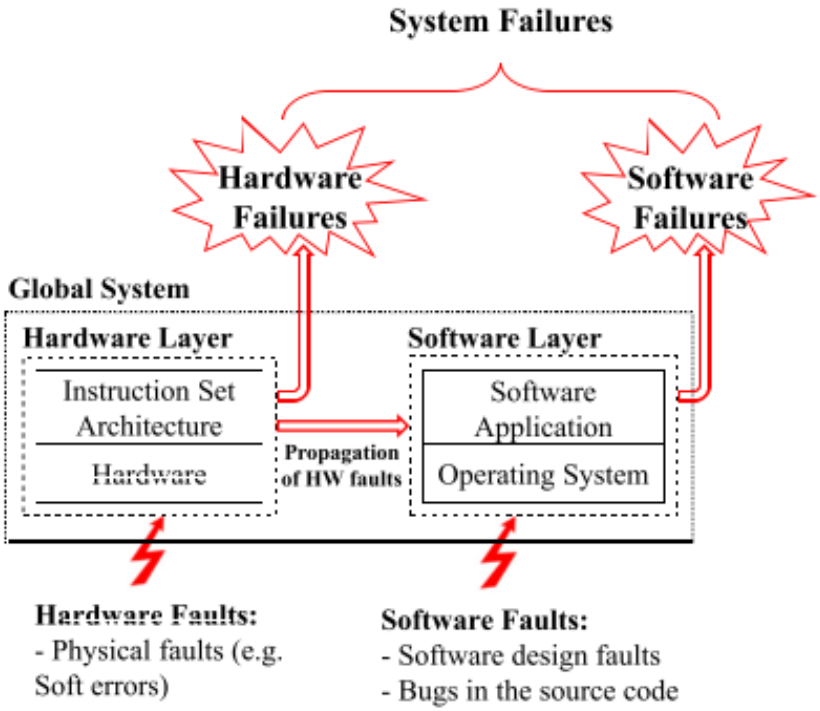
3.1 Injeção de Falhas de *Hardware*

Implementações nesta categoria se utilizam de *hardware* externo para introduzir de falhas no alvo. Normalmente, estas técnicas permitem alcançar regiões que as injeções por *software* não pode chegar, no entanto tendem a ser caras e dispendiosas em tempo e execução. Dependendo da sua implementação as falhas de *hardware* podem ser divididas em duas categorias:

- falhas com contato: incluem falhas com contato físico direto com o alvo. Sobretensão ou sobrecorrente podem ser gerados com métodos usando *probes* ou *sockets* para direta intromissão nas características elétricas do alvo.
- falhas sem contato: nesta categoria incluem-se injetores que criam fenômenos físicos externos ao alvo. Canhão de íons pesados, geradores de interferência eletromagnética e o uso de laser são exemplos de injetores sem contato.

Kooli et al. (2017) ainda coloca como possíveis técnicas de injeção de falhas de *hardware* as simulações em VHDL e Verilog e as baseadas em modelos de falhas implementados em FPGA, que usam reconfiguração e instrumentação (STERPONE;

Figura 3.3 - Propagação de Falhas



FONTE: Kooli et al. (2017)

VIOLANTE, 2007; ZARANDI et al., 2003).

3.2 Estado da Arte em Injeções de Falhas de *hardware*

Messaline (ARLAT et al., 1990) é uma ferramenta desenvolvida no LAAS-CNRS, França, que utiliza tanto *probes* como *sockets* para introdução de falhas em nível de pinos. Algumas falhas que podem se injetadas pelo Messaline são *stucked-at* e falhas lógicas complexas todas com possível controle de duração e frequencia. Seu uso vem desde a década de 1990 em redes de trem automatizadas até recentes validações de estudos sobre injeção de falhas (JEONG et al., 2016).

FIST (GUNNEFLO et al., 1989) - *Fault Injection System for Study of Transient Fault Effect* - é uma ferramenta desenvolvida pela Universidade Chalmers de Tecnologia, na Suíça, que faz uso de íons pesados para a injeção de falhas transitórias em locais aleatórios de *Systems-on-Chip*. A fonte de radiação montada é uma câmara de vácuo com dois sistemas de processadores. Um diretamente bobardeado pela radição e outro como grupo de controle para referência de onde e quando acontecem as falhas (HSUEH et al., 1997). Além dos efeitos de radição, o FIST pode injetar falhas no meio de alimentção elétrica dos circuitos. Os resultados experimentais demontram

que ambos os métodos resultam em efeitos similares. No entanto, a radição tem mais efeito sobre barramentos de comunicação enquanto que falhas de alimentação afetam mais os sinais de controle (GUNNEFLO et al., 1989).

MARS - Maintainable Real-time System - (KARLSSON et al., 1998) é uma arquitetura tolerante à falhas, distribuída, concebida na Universidade de Viena. Além dos efeitos de radiação por íons pesados, o MARS utilizou-se de campos eletromagnéticos para a injeção de falhas sem contato direto. Pesquisadores utilizaram e compararam os três métodos (radiação, campos EM e *pin-level*) para exercitar os mecanismos de detecção de falhas do MARS (KARLSSON et al., 1998), onde esses se mostraram complementares no que diz respeito à geração de diferentes tipos de erros.

3.3 Injeção de Falhas de *Software*

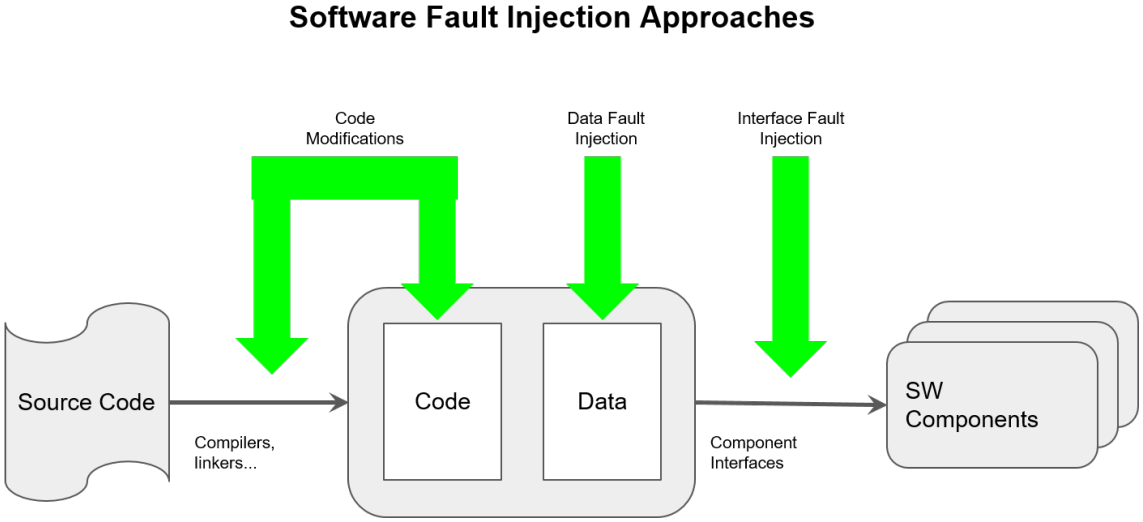
Com o passar dos anos, dada a evolução dos sistemas e rápido desenvolvimento de novas tecnologias, falhas relacionadas a *software* tem sido a causa de grandes perdas econômicas e levado, também, a perda de vidas humanas, e.g. 1996 Ariane 5 *Software Failure* (DOWSON, 1997). Com isso, injeção de falhas vem se tornando objeto de mais trabalhos, mudando o foco das pesquisas de *hardware* em direção a falhas de *software* (NATELLA et al., 2016).

A emulação de falhas de *software* tem sido perseguida ao longo dos anos, técnicas de ferramentas vem sido desenvolvidas, o que nos leva a identificação de três abordagens de injeção de falhas 3.4:

- Injeção de Erros de Dados (*Data Fault Injection*): corrupção de registradores e regiões de memória de maneira similar à injeção de falhas de *hardware*.
- Injeção de Falhas de Interface (Interface Fault Injection): corrupção de dados de entrada e saída nas interfaces dos componentes.
- Injeção de Modificações de Código (Code Modifications): emular falhas de *software* introduzindo códigos defeituosos a fim de mimetizar a maioria dos *bugs* de programação.

As primeiras abordagens de injeção de falhas de *software*, SFI, cresceram no contexto de estudos em *Software-Implemented Fault Injection*, SWFI, para falhas de *hardware*. O SWFI tem por objetivo reproduzir os efeitos de falhas de *hardware* (falhas de CPU, barramento e memória) perturbando os estados de regiões de memória ou registradores através de *software* (NATELLA et al., 2016).

Figura 3.4 - Visão Geral da Injeção de Falhas em *Software*



FONTE: Adaptado de Natella et al. (2016)

Estudos empíricos demonstram que a injeção de mudanças de código podem emular realisticamente falhas de *software* de forma que mudanças no código podem resultar em defeitos reais na execução de um programa. Alterações de variáveis de inicialização, mudanças na ordem de execução de tarefas, omissão de funções, são exemplos de falhas a serem inseridas com essa abordagem (NATELLA et al., 2016).

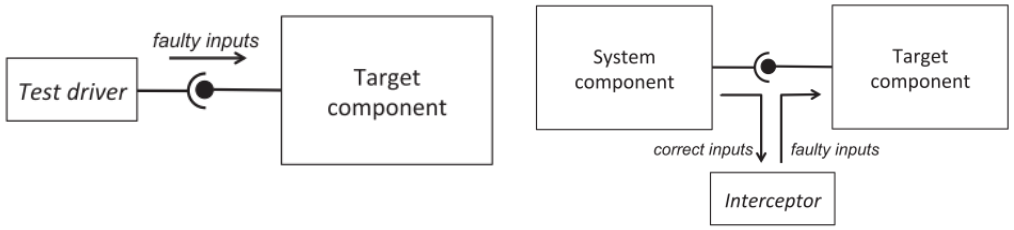
Injeção de erros de interface é comumente adotada para testes de robustez. Essa abordagem utiliza dois métodos distintos de implementação, conforme observado na Figura 3.5: a) um *drive* de testes conectado ao componente alvo (*Target component*) com o objetivo de emular o comportamento e injetar falhas; e, b) interceptação e corrupção das interações entre o alvo (*Target component*) e o resto do sistema (*System component*) por meio de um elemento interceptador (*Interceptor*) (NATELLA et al., 2016).

Implementações do tipo injetor *drive* são usadas em contextos HIL de testes de aceitação para emular o ambiente externo ao SUT, por exemplo teste em câmara de termo-vácuo.

Por outro lado a implementação de um injetor do tipo interceptador possibilita a integração no contexto de testes caixa-preta dado que a implementação realizada no SUT não precisa ser, necessariamente, conhecida. O foco passa a ser a interface comum e como as falhas, erros e defeitos podem ser emuladas.

O critério adotado para os erros de interface fora imaginado por Johansson e Suri

Figura 3.5 - Abordagens de Injeção de Erros de Interface



(a) *Drive* de Testes (b) Interceptador

FONTE: Natella et al. (2016)

(2005) e Carreira et al. (1999)

- O QUE injetar: um valor inválido, *bit-flip* ou qualquer outro tipo de falha baseada em tipo de dado.
- ONDE injetar: identificado pela arquitetura do sistemas, falhas podem ser injetadas em entradas ou saídas da interface de um *software*.
- QUANDO injetar: a injeção pode ser ativada quando um serviço do alvo é requerido, número de serviços ou tempo decorrido.

SWFI tornou-se, então, um caminho possível para atuar na melhoria da qualidade de missões baseadas em CubeSats por meio dos testes de integração dos subsistemas comunicantes, uma vez que suas características de desenvolvimento distribuído revelam que as interfaces entre os subsistemas comunicantes são um acesso estratégico para medir, avaliar e, até mesmo, interferir na requerida robustez do subsistema.

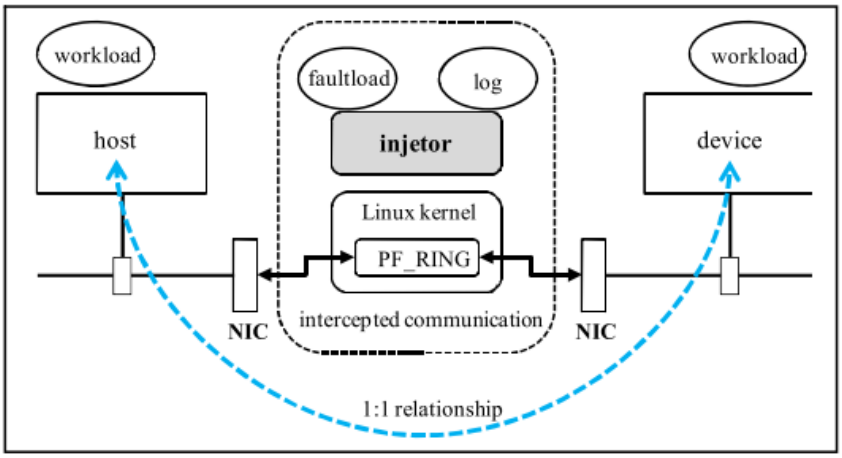
3.4 Estado da Arte em Injeções de Falhas de *Software*

O FITT - *Fault Injector Test Tool* -, desenvolvido pela UFRGS, (DOBLER et al., 2016), é uma ferramenta de software específica para validar implementações PROFIsafe. A ferramenta é completamente independente do *host* e do *device*, que não enxergam o injetor, Figura 3.6.

O ambiente é baseado em um computador convencional e duas interfaces de rede (NIC - *network interface communication*).

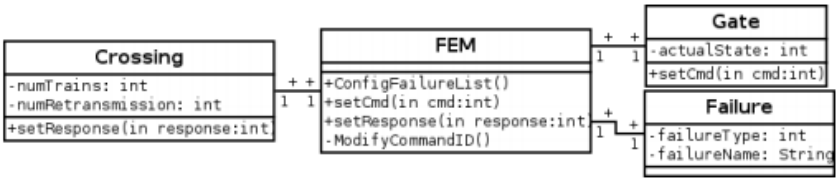
Com tal arquitetura, é possível a injeção de erros em camada de interface com uma ferramenta reutilizável não intrusiva no código dos dispositivos alvo em teste.

Figura 3.6 - Ambiente de Injeção de Falhas do FITT



FONTE: Dobler et al. (2016)

Figura 3.7 - Diagrama de classe da arquitetura com uso do FEM



FONTE: Weller et al. (2015)

O FITT roda em ambiente Linux, capturando as mensagens entre o *host* e o *device* causando a mínima intereferência temporal como necessária nos requisitos do PROFIsafe (DOBLER et al., 2016).

Outro exemplo de injeção de falhas em caráter de interface é o proposto pela metodologia InRob (MATTIELLO-FRANCISCO, 2009), que visa testes de interoperabilidade e robustez entre sistemas comunicantes. A metodologia prevê o uso de um FEM (*Failure Emulator Mechanism*), Figura 3.7, para emular defeitos em um canal de comunicação representando um canal de comunicação que representam falhas, no contexto de de robustez, dos subsistemas comunicantes nesse canal. Para o estudo de caso, fora idealizado a passagem de um trem em um cruzamento, sua comunicação entre um sensor de passagem do veículo (*Crossing*) e a cancela (*Gate*) do cruzamento (WELLER et al., 2015).

A inclusão do modelo de um injetor de falhas (FEM) no modelo das IUTs interoperantes facilita a sua reutilização em outras aplicações além de permitir sincronismo

entre *faultload* e *workload* na realização dos testes de robustez (WELLER et al., 2015).

Os próximos capítulos desta dissertação apresentam o uso da técnica de injeção de falhas nos testes de interoperabilidade de subsistemas de nanossatélites com foco nos aspectos de robustez, validando o comportamento, interoperabilidade de subsistemas na presença de falhas de comunicação (interface) entre SUT1 e SUT2. A função de injetor de falhas é feita por uma ferramenta do tipo interceptador, o mecanismo emulador de defeitos, FEM. Este implementa uma interface, com falhas, entre dois SUT objetivando emular possíveis defeitos no canal de comunicação através de implementações em *software* e/ou *hardware*.

4 MECANISMO EMULADOR DE DEFEITOS – FEM

O conceito de um mecanismo emulador de defeitos, FEM, voltado aos testes de interoperabilidade e robustez no campo das missões espaciais advém de trabalhos anteriores (MATTIELLO-FRANCISCO, 2009; CONCEICAO et al., 2016) onde a metodologia fora proposta para testes de integração de subsistemas. O FEM é parte de um sistema de testes (TS) responsável por injetar falhas nas mensagens trocadas entre dois subsistemas intensivos em *software* (SiS) através do canal de comunicação, como um injetor de falhas interceptador.

O mecanismo foi originalmente proposto para auxiliar apenas a execução de um conjunto de testes gerado automaticamente usando abordagem de testes baseados em modelos (MBT).

Pouca atenção havia sido dada ao FEM e seu potencial uso em fase inicial de desenvolvimento dos subsistemas.

O objetivo deste emulador é interceptar as mensagens em um canal de comunicação específico e inserir, ou não, falhas que visem emular defeitos e erros no SUT composto por dois SiS comunicantes. Como citado anteriormente, o FEM atua como injetor de falhas de interface, do tipo interceptador, evitando, assim, modificações diretas no código embarcado nos SiS habilitando mimetizar falhas a partir de erros controlados na interface do barramento de comunicação.

Considerando a evolução das abordagens de engenharia orientada a modelos e ferramentas disponíveis para geração automática de código a partir, também, de modelos, onde pode-se facilmente embarcá-lo no *hardware* alvo para simulações comportamentais, o conceito do FEM foi estendido.

FEM pode ser útil em diferentes etapas dos processos de desenvolvimento, integração e testes. Em nível de modelos, MIL – *Model in the loop*, antecipar no ciclo de desenvolvimento, problemas que viriam a se apresentar apenas durante a fase de integração. Tais requisitos podem ser verificados usando MIL, o que pode aumentar a confiança nos requisitos especificados. E, quando a implementação final de um SUT se encontrar disponível, o mesmo FEM pode ser usado para testes de conformidade, aceitação e integração, agora em um cenário HIL, *Hardware in the loop* (NIBERT et al., 2012; SHOKRY; HINCHEY, 2009; VULI et al., 2010).

Concebido para testes de subsistemas em nanossatélites, o FEM proposto atua no suporte aos testes de integração de subsistemas comunicantes aplicável em diferentes

missões baseadas em CubeSats. O *framework*, aqui idealizado, para o FEM é uma aboradagem idependente de plataforma a qual trabalha como um guia para futuras implementações em diferentes interfaces e diferentes SUT compostos de no mínimo dois SiS comunicantes a bordo de nanossatélites.

4.1 Arquitetura

A arquitetura do FEM segue a abordagem de um interceptador para a injeção de falhas em nível de interface fazendo uso da ISO 9646 (TRETMANS et al., 1991) como referência para testes embarcados.

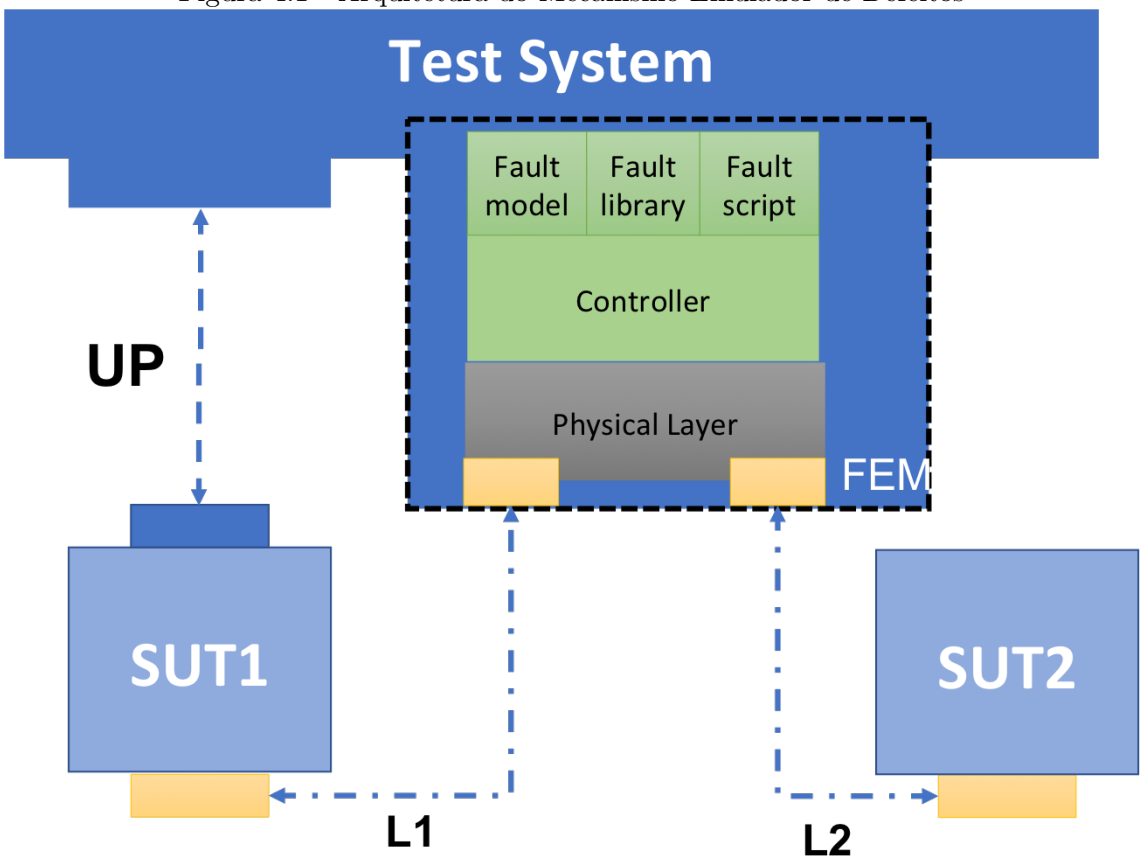
A Figura 4.1 apresenta o FEM como parte de um TS (*Test System*) projetado para apoiar na integração de dois SiS comunicantes embarcados em dois subsistemas diferentes do nanossatélite, referidos como SUT1 e SUT2. Pode-se observar que os acessos do TS aos SUTs são feitos por meio dos pontos de observabilidade e controlabilidade (PCOs), representados pelas linhas tracejadas: UP, referida como interface superior e; L1 e L2, referidos por interface inferior dos SUTs. Dada que a arquitetura do TS pressupõe uma relação Mestre-Escravo entre, respectivamente, o SUT1 e o SUT2. UP refere-se à interface direta com o testador, onde o PCO (ponto de controlabilidade e observabilidade do caso de teste) atua no dispositivo mestre.

O FEM permite um acesso individual à interface inferior (LOW) de ambos os SUT, L1 e L2, respectivamente, criando um novo ponto de observabilidade, na medida que monitora as mensagens trocadas no canal de comunicação, e controlabilidade, na medida que injeta as falhas. Sendo um mecanismo intrusivo no canal de comunicação, o FEM pode atuar em L1 em dado momento, emulando um mal-funcionamento do SUT2 e vice-versa.

A vantagem deste tipo de injetor de falhas para testes de integração é evitar mecanismos intrusivos em cada SUT para testar propriedades de robustez. Por outro lado, o preço a ser pago é o atraso inserido no canal de comunicação pelas operações do FEM, que são representados pela camada física na arquitetura.

A arquitetura proposta para o FEM ainda conta com mais duas camadas. A primeira é responsável por implementar as funcionalidades da injeção propriamente especificada para o canal. Essa camada define a plataforma onde todo o FEM será embarcado. A segunda camada, nomeada Controlador, é responsável por auxiliar a execução do FEM através de três elementos do *faultload*: (i) o modelo de falhas - *fault model*, que lida com as falhas que serão modeladas em alto nível, (ii) a bibli-

Figura 4.1 - Arquitetura do Mecanismo Emulador de Defeitos



FONTE: Próprio Autor

oteca de falhas - *fault library*, lida com as falhas a serem implementadas e (iii) o roteiro de falhas - *fault script*, que controla passa-a-passo a seqüência de falhas a serem injetadas. Com o *faultload* definido, o controlador é capaz de operar a execução da injeção de falhas baseado no modelo, implementado pela biblioteca e guiado pelo roteiro.

Para que o FEM possa atuar de forma correta, a implementação depende de definições da plataforma em termos de: interface (L1 e L2), *faultload* e injetor de falhas, como a seguir.

4.2 Interface

A interface deve ser bem definida em termos das características lógica e elétrica. Questões como: Qual a camada física? Qual a variação dos sinais presentes? Há um protocolo com troca de mensagens pre-estabelecidas entre os SUT por meio das interfaces L1 e L2? O protocolo é dependente do tempo? A comunicação é serial?

Paralela? A conexão mecânica, existe? Pinos? Estes são exemplo de questões a serem levantadas para definir e especificar a interface entre os subsistemas comunicantes e iniciar a implementação do FEM.

A definição da interface também impacta a escolha do injetor de falhas a ser utilizado, uma vez que este deve estar em conformidade com a especificação da interface.

Quando falamos de sistemas espaciais, essas informações podem ser encontradas na documentação da plataforma orbital. Utilizando das normas da ECSS (*European Cooperation for Space Standardization*), diferentes documentos devem ser produzidos durante o desenvolvimento e ciclo de vida de um produto espacial. Entretanto, ainda há deficiência na produção desses documentos em um grande número de projetos de nanossatélites. Muito devido a filosofia de curto tempo de desenvolvimento, especificações e documentações são deixadas à margem a fim de reduzir tempo.

4.3 Faultload

O *Faultload* determina as falhas que serão injetadas na interface escolhida pelo TS para a execução dos casos de teste. Utilizando uma estratégia que se baseia nos erros (GHOSH et al., 1997) permitimos ao testador que identifique de antemão possíveis fontes de defeitos e erros, assim, desenvolvendo técnicas de injeção para cada categoria de falha.

Como definido anteriormente, o *faultload* é composto por três elementos: modelo, biblioteca e roteiro.

- Modelo de Falhas – uma definição das falhas em alto nível de abstração a serem injetadas. Bonfiglio et al. (2015) permite especificar as possíveis falhas de um sistema como a degradação de seus serviços e classifica-os como: (i) falhas de valor, valores incorretos, mistura de dados, *single event upsets* são o maior exemplo desta categoria; (ii) falhas de provisão, erros de exceção, entradas ou comandos não esperados completam esta categoria e; (iii) falhas relacionadas a tempo, atrasos, supressão de mensagens, falhas de sincronismo definem esta categoria. Considera-se neste trabalho uma quarta categoria de falha, relacionada ao meio físico. Este tipo de falha compreende falhas exclusivamente relacionadas à camada física da interface, *single event latchup*, queima de componente e outras falhas inerentes e exclusivas do *hardware* são parte desta categoria.
- Biblioteca de Falhas – conjunto de falhas implementadas em código no

FEM. Do modelo de falhas, a biblioteca representa as funções reais que o injetor irá operar de forma a emular os defeitos. Falhas de valor podem ser representadas por *bit-flips* em mensagens, falhas de provisão podem ser codificadas como troca de bytes inteiros em um pacote. Injetando atrasos suficientemente longos, podemos gerar supressão de mensagens (falhas relacionada ao tempo) e forçar o desligamento de uma interface, podendo emular a queima, e perda, de um componente ou subsistema, como uma falha do meio físico.

- Roteiro de Falhas – o principal condutor do FEM. Como a biblioteca especifica quais falhas podem ser injetadas e como elas trabalham, o roteiro habilita a ordenação das falhas. Cabe ao roteiro estabelecer ONDE injetar, QUANDO injetar, O QUE injetar e COMO injetar.

Assim, o *Faultload* é o cerne do FEM, definindo as falha a serem injetadas, desde sua abstração (modelo) até a execução (roteiro), passando pela implementação (biblioteca).

4.4 Injetor de Falhas

O injetor de falhas é a plataforma onde o FEM é efetivamente implementado. É o responsável pela ativa atuação na interface entre os SUT (camada física), implementa o modelo e a biblioteca de falhas e interpreta o roteiro instrumentando a injeção de falhas em cada um dos casos de teste. Também monitora a interface, realimentando o controlador com informações sobre as mensagens trocadas.

A escolha da plataforma deve contemplar a seguinte restrição: mínima interferência na interface em termos de tempo de execução a fim de evitar atrasos significativos na comunicação das mensagens. O atraso imposto pelo FEM pode causar problemas de *handshaking* consumindo as margens de tempo estipuladas na implementação dos SUT.

Uma vez que a função do FEM é atuar no canal de comunicação entre os SUTs comunicantes, *a priori*, quando injetando falhas, ou não, sua interferência deve ser quase imperceptível do ponto de vista dos SUT. Em outras palavras, intrusividade é uma preocupação. Assim, as restrições de tempo na comunicação também devem ser levadas em consideração quando implementando o FEM na plataforma escolhida.

Em suma, o injetor de falhas usa a interface física para interceptar as mensagens trocadas entre os SUT1 e SUT2, injetando as falhas definidas pelo faultload e retor-

nando ao controlador o que está acontecendo no canal de comunicação.

5 ESTUDO DE CASO: NanoSatC-Br2

O NanoSatC-BR2 é o segundo nanossatélite do programa de missões científicas NanoSatC-BR desenvolvida no Instituto Nacional de Pesquisas Espaciais - INPE - em cooperação com a Universidade Federal de Santa Maria - UFSM. O primeiro satélite da família, o NanoSatC-BR1, é um CubeSat de 1U ($10 \times 10cm^3$) lançado em 2014 e ainda operacional em órbita para aquisição de dados da magnetosfera terrestre.

NanoSatC-BR2 é um CubeSat de 2U ($10 \times 10 \times 20cm^3$) também com propósito científico embarcado com cinco cargas úteis (SCHUCH et al., 2017). O software de gestão de bordo (em ingles OBDH - *On Board Data Handling*) e uma das cargas úteis científicas são desenvolvidas localmente no INPE e as demais são providas por diferentes parceiros de pesquisa em universidades brasileiras. As cargas úteis do NanoSatC-BR2 são: um conjunto de magnetômetros, um *chip* ASICS, uma FPGA tolerante a falhas, uma sonda de Langmuir e um sistema de determinação de atitude tolerante a falhas.

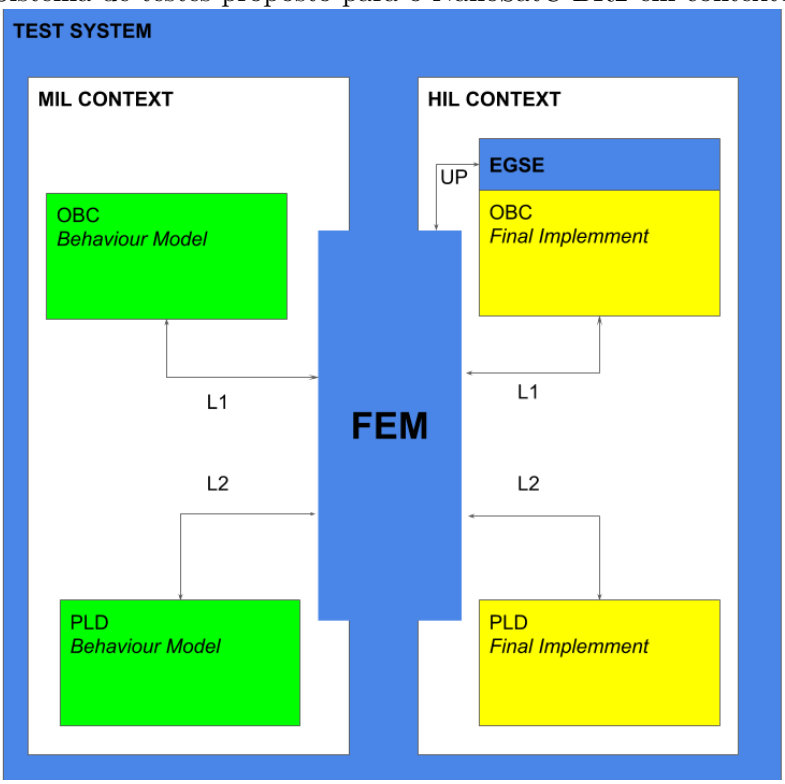
O FEM se enquadra no apoio aos testes de robustez e avaliação de interoperabilidade do OBC (*On Board Computer*) da plataforma com as cargas úteis sendo parte do sistema de testes desenvolvido por Conceicao et al. (2016).

Baseado na arquitetura do FEM apresentada na Figura 4.1, um FEM particular foi prototipado para o sistema de testes do NanoSatC-BR2 seguindo o *framework* proposto no capítulo anterior para ser utilizado em fases diferentes do desenvolvimento. A Figura 5.1 apresenta os contexto MIL (MIL *Context*), cujo foco é a verificação de requisitos de robustez do OBC (utilizando um modelo comportamental, *OBC Behaviour model*) com cada carga útil (utilizando modelo comportamental, *PLD Behaviour model*) e HIL (HIL (*Context*), cujo foco é o teste de integração das versões finais da carga útil (PLD *Final Implemment*) com o OBC (OBC *Final Implemment*), proposto para o NanoSatC-BR2. O uso do EGSE (Electrical Ground Support Equipment) é mandatório na execução dos testes de integração do satélite visto que esta atua como interface UP do TS.

5.1 Identificando a Interface

O I^2C é a interface disponível na plataforma, fornecida pela empresa ISIS - *Innovative Solutions in Space*, utilizada pela família NanoSatC-BR para a interação das cargas úteis com o computador de bordo, OBC. Logo, o OBC usa o protocolo da

Figura 5.1 - Sistema de testes proposto para o NanoSatC-BR2 em contexto MIL e HIL



FONTE: Adaptado de Conceicao et al. (2016)

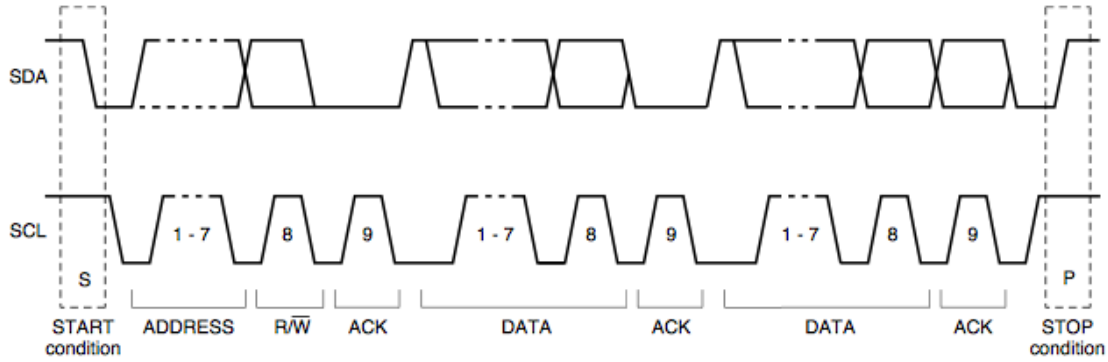
plataforma, I^2C, para a maioria das comunicações no satélite. Este protocolo foi criado pela empresa Philips como Inter-IC, uma interface de comunicação simples e rápida entre circuitos integrados.

A interface elétrica para comunicação é baseada no TWI (*Two Wire Interface*), com um fio para a linha de dados serial, SDA, e outro para a linha de *clock* serial, SCL. Ambas as linhas são conectadas via um resistor ao nível lógico alto, $3V3$ ou $5V$. A transmissão ocorre colocando a linha de dados para nível lógico baixo e amplificadores de alta impedância monitoram a tensão sobre o barramento para a recepção dos dados. Sempre no contexto mestre-pergunta-escravo-responde (PHILIPS SEMICONDUCTORS, 2003).

Toda comunicação é controlada pelo dispositivo mestre, responsável por iniciar as transmissões e controlar a sincronização.

Na maioria dos projetos CubeSat, toda a plataforma de serviço e carga úteis são conectados a um único barramento I^2C de difusão, com cada subsistema identificado com um endereço de 7-bit único.

Figura 5.2 - Estrutura do *frame* I^2C



FONTE: Philips Semiconductors (2003)

O *frame* de comunicação do I^2C segue a estrutura apresentada na Figura 5.2.

No NanoSatC-BR2 toda a comunicação entre o OBC e as cargas úteis usa a interface I^2C seguindo o protocolo mestre-escravo, com o OBC sendo o único mestre.

5.2 Instanciando a Arquitetura do FEM

Uma vez que a interface está definida, a arquitetura proposta para o FEM pode ser instanciada para apoiar os testes de integração de qualquer missão de nanossatélite compatível com a família NanoSatC-BR.

De acordo com a Figura 4.1, o FEM fora instanciado como parte do TS do NanoSatC-BR2. Pode-se observar que os elementos internos ao software implementado no FEM são representados em blocos verdes. Itens do *Faultload* e o Controlador estão embarcados no *hardware* alvo. A comunicação do TS com o OBC é realizado via interface UPPER para propósitos de controle o observação (PCO). A interface LOWER é o barramento I^2C, que conecta os SUT1 e SUT2, respectivamente computador de bordo (OBC) e carga útil (PLD), por meio do FEM. O SUT1 comporta o conjunto *hardware* e *software* embarcado no OBC assim como SUT2 representa a carga útil (*hardware* e *software*).

Uma vez instanciada a arquitetura do FEM, sua implementação pode ser reutilizada para verificar o comportamento das interações do OBC com cada carga útil do NanoSatC-BR2.

5.3 Definindo o *Faultload*

A construção do modelo de falhas depende da análise de algumas situações de falhas comuns em sistemas espaciais. No caso de robustez, as falhas relacionadas ao ambiente espacial (e.g. *single event effects* - SEE) merecem grande atenção ao iniciarmos a construção do modelo de falhas, uma vez que os SEE relacionam-se à radiação e aos eventos de raios cósmicos que podem atingir o satélite.

Estes eventos podem resultar em diferentes falhas, categorizadas como:

- SEU (Single Event Upset) – evento único onde um próton pesado atinge o veículo. Pode ser o causador de *bitflips* em mensagens ou registradores de memória.
- SEL (Single Event LatchUp) – devido a radiação, alguns semicondutores podem começar a elevar sua corrente elétrica.
- SEB (Single Event Burnout) – o choque de um próton pesado diretamente sobre um equipamento pode levá-lo a "queimar"completamente.
- SEGB (Single Event Gate Burnout) – mais presente em componentes MOSFET, este SEE causa a destruição do terminal GATE de um transitor de efeito de campo (FET).

As falhas causadas por SEE (Tabela 5.1) podem ser adicionadas ao modelo em diferente categorias (ex. SEU como falhas de valores, SEB como falha física). Outras falhas a serem consideradas para este estudo compreendem falhas de exceção, comandos em momentos inoportunos levando a estados inesperados, falhas de provisão. Como sistemas espaciais são sistemas de tempo real, restrições de tempo são prioritárias em análises e testes de robustez.

Com o modelo e biblioteca definidos, o *script* usa o critério conforme definido na seção 3.3. WHERE, onde injetar é demonstrado como um commando `WRITE` ou resposta `READ` do dispositivo mestre. WHEN, quando injetar é controlado pelo índice `cnt_msg`, um contador do número de mensagens trocadas. WHAT, o que injetar é um valor numérico único para cada falha implementada na biblioteca (*bit-flip* para emular SEU, *change* para emular casos de exceção, *delay* para emular atrasos e outras falhas relacionadas a tempo e *lock_down* para obstruir a linha de dados para emular um dispositivo com defeito). HOW, como injetar é o argumento de cada falha implementada pela biblioteca (ex. que *bit* mudar em um *byte*, qual novo

Tabela 5.1 - FEM *Faultload* para o contexto do NanoSatC-BR2

MODEL	**LIBRARY**	**SCRIPT**			
		WHERE	WHEN	WHAT	HOW
VALUE					
SEU	`bitflip(int bit2flip)`	*R/W*	*cnt_msg*	*0x01*	*bit2flip*
PROVISION					
Exception	`change(int new_cmd)`	*R/W*	*cnt_msg*	*0x05*	*new_cmd*
TIME					
delay message	`delay(int time)`	*R/W*	*cnt_msg*	*0x03*	*time <byte_timeout*
suppress message	`delay(int time)`	*R/W*	*cnt_msg*	*0x03*	*time >byte_timeout*
PHYSICAL					
SEB	`lock_down(int time)`	*R/W*	*cnt_msg*	*0x07*	*time >reset_timeout*

comando substituir, quantos microssegundos atrasar uma mensagem ou quantos microssegundos simular um travamento da comunicação)

Para futura implementação do roteiro no injetor de falhas, cada critério do *faultload* (WHERE, WHEN, WHAT e HOW) é definido e identificado por um único código ou tipo. Como o seguinte:

```
enum WHERE {
    READ  = 0x00,
    WRITE = 0x01
};

uint8_t WHEN;

enum WHAT {
    BITFLIP = 0x01,
    CHANGE  = 0x03,
    DELAY   = 0x05,
    LOCK    = 0x07
};

uint8_t HOW;
```

Tabela 5.2 - Especificações básicas para os Arduinos UNO e DUE

	Arduino UNO	**Arduino DUE**
uController	ATMega328P	AT91SAM3X8E
Operation Voltage	5 V	3.3 V
nbr. of Digital I/O	6	54
Flash Memory	32 KB	512 KB
Input Voltage	7-12 V	7-12 V
Clock Speed	16 MHz	84 MHz
Nbr of I2C Interfaces	1	2

5.4 Implementando o Injetor de Falhas

Com a interfce e o *faultload* descritos é possível escolher e implementar o injetor de falhas.

Na tentativa de manter a filosofia do "rápido e barato"para esta implementação, foram escolhidas as placas Arduino como solução de plataforma para o injetor de falhas. As placas selecionada foram a Arduino Uno, modelo mais simples com processador ATMega328P de 8-bit e apenas uma interface TWI, e a Arduino DUE, placa mais robusta com um processador ARMCortex-M3 de 32-bit e duas interfaces TWI.

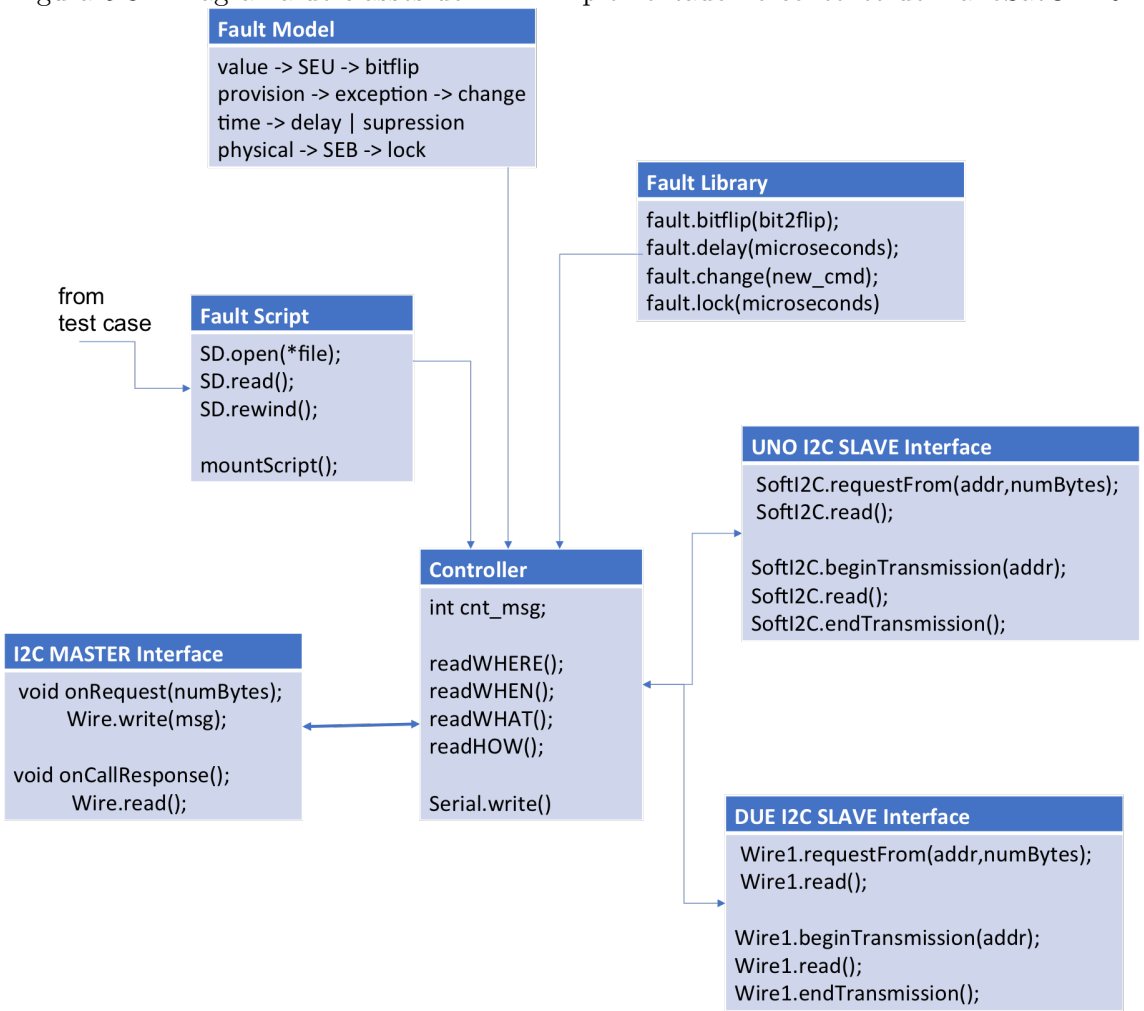
A Tabela 5.2 apresenta as especificações básicas das duas placas.

Como a placa UNO possui apenas uma interface I^2C se fez necessária a utilização de outros dois pinos de I/O para manutenção da arquitetura idelizada para o interceptador. Assim, utilizando uma biblioteca *open-source*, `SoftI2CMaster.h`, em adição à bilbioteca padrão de comunicação TWI do Arduino, `Wire.h`, assim sendo possível utilizar outro par de pinos I/O como SDA e SCL. Este problema não se apresentou na placa Due, dado que esta já possui duas interfaces TWI e diferentes bibliotecas para gerenciar cada par de pinos separadamente, `Wire.h` e `Wire1.h`.

A Figura 5.3 apresenta as funções do *software* implementado para a comunicação mestre-escravo via I^2C usado no testes entre o OBC e a PLD. O controlador recebe todas as informações do *faultload* (modelo, biblioteca e roteiro) e executa a injeção de falhas usando diferentes bibliotecas para emular a interface.

A função `Serial.write()` retorna ao testador o que esrtá acontecendo durante o teste com o FEM atraveś de um monitor serial. Para o carregamento do roteiro de falhas, é utilizado um módulo cartão SD com dois arquivos `*.TXT`, configuração e

Figura 5.3 - Diagrama de classes do FEM implementado no contexto do NanoSatC-BR2



FONTE: Próprio Autor

Tabela 5.3 - Resultados do FEM implementado nas diferentes placas Arduino

	Arduino UNO	**Arduino DUE**
Memory Used	92%	7%
DELAY INTERFERENCE		
repass WRITE	800 us	70 us
repass READ	750 us	50 us
value WRITE	2 ms	150 us
value READ	1.6 ms	100 us
provision WRITE	1 ms	95 us
provision READ	1 ms	90 us
time WRITE	800 us	70 us
time READ	750 us	50 us

caso de teste. Ambos os arquivos são matrizes de inteiros, a primeira é uma matriz 1×4 e a segunda, uma matriz de $n \times 4$, onde n é o número de falhas a serem injetadas, esta informação encontra-se no arquivo de configuração.

```
uint8_t array CONFIG[id, SUT1, SUT2, nfaults];
```

```
uint8_t array FEM[nfaults][where, when, what, how];
```

Os outros elementos do *faultload*, modelo e biblioteca, estão já implementados no código do Arduino assim como o ambiente do controlador. O uso desse arquivos explora a abstração do FEM para o usuário, Já que a sintaxe para esses arquivos é bem conhecida, qualquer caso de teste abstrato pode ser construído e executado sem nenhum, ou com pouco, ao menos, retrabalho na implementação do FEM.

Um dos problemas apresentado pela placa ATmega é a sua limitação de memória. O código gerado para o *faultload* e controlador usa, aproximadamente, 92 % da memória do microcontrolador. A tentativa de codificar novas funcionalidades no UNO resultou em instabilidades no programa.

A migração do código para a placa DUE, já com modelo e biblioteca definida, fora rápida e resultou, também, numa interferência temporal menor por parte do injetor. Abrindo, assim, caminho para a implementação de novas funcionalidades para o FEM.

A Tabela 5.3 apresenta a comparação entre as duas placas em termos de interferência temporal em cada uma das funções desempenhadas pelo FEM.

As comparações foram feitas utilizando a mesma implementação em ambas as placas: modelo, biblioteca e controlador a fim de manter a coesão nos resultados.

O Arduino DUE se mostra uma solução viável, com duas interfaces TWI nativas, maior velocidade de *clock* e maior espaço de memória que seu irmão menor. Como dito, a placa DUE possibilita a adição de novas funcionalidades ao FEM.

6 USO DO FEM

Buscando validar o protótipo do FEM em uma aplicação real, dois SiS comunicantes a bordo do NanoSatC-BR2 foram utilizados no contexto MIL com o propósito de verificação de requisitos.

Os subsistemas interoperantes são: o OBC, responsável pelas funções de manejo dos dados a bordo da plataforma e a sonda de Langmuir (SLP), uma carga útil científica do NanoSatC-Br2. A Figura 6.1 mostra o diagrama de sequência representando o comportamento esperado das interações entre OBC e SLP através do canal de comunicação I^2C.

6.1 Modelos de Interoperabilidade OBC – SLP

Com o intuito de gerar códigos que simulam o comportamento dos dois subsistemas comunicantes, OBC e SLP, foram utilizados os modelos de interoperabilidade, construídos com o formalismo *timed automata* (PALAMIN; MATTIELLO-FRANCISCO, 2017).

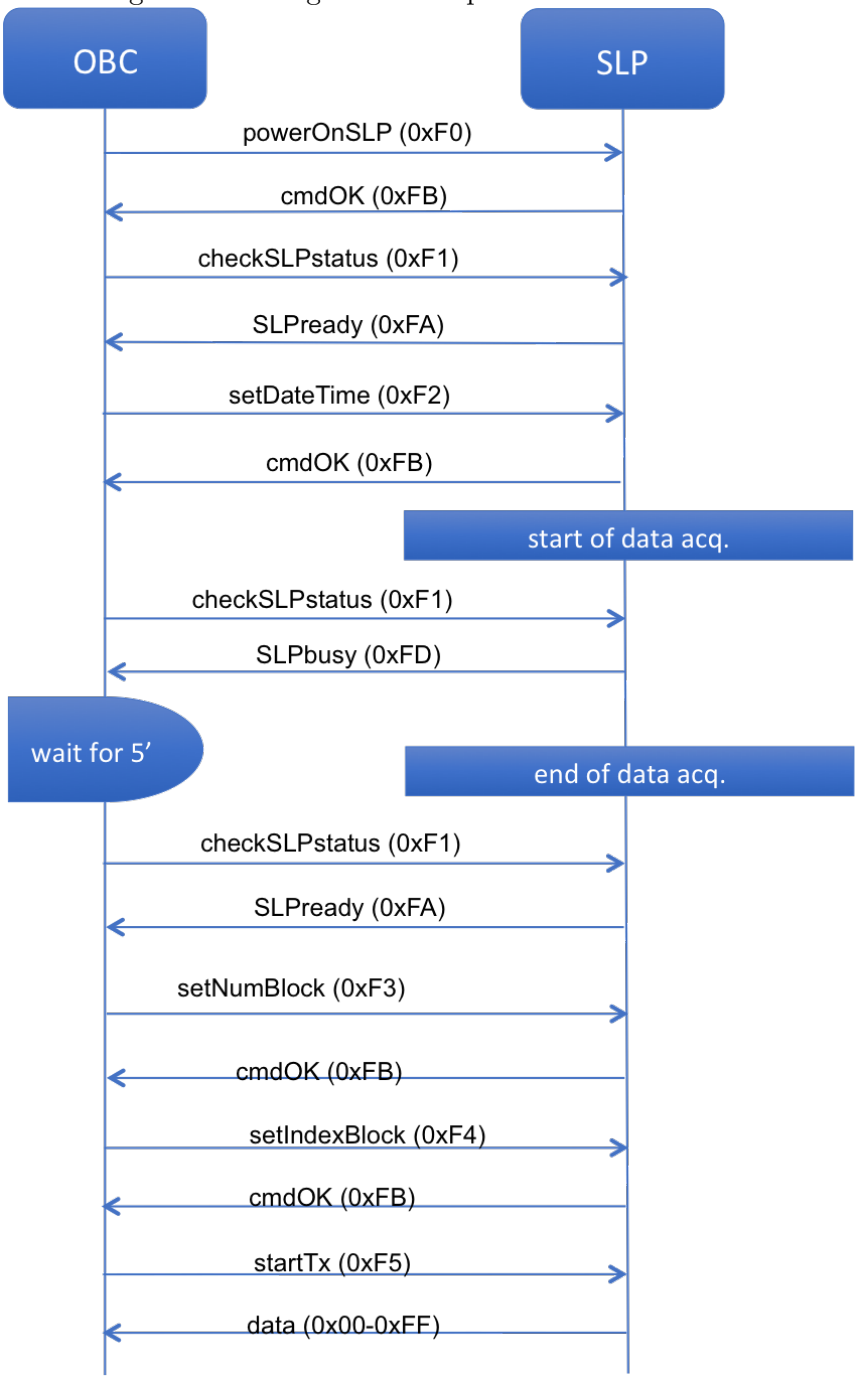
Baseado nesses modelos, foi possível desenvolver código do comportamento esperado de cada subsistema para ser carrega-do em uma placas Arduino Uno, prototipando o cenário MIL. Assim, a interoperabilidade entre OBC e SLP fora explorada com a finalidade de demonstrar a efetividade do protótipo do FEM.

Olhando a Figura 6.1, nota-se que cada comando enviado à SLP (seta OBC-SLP, comando `WRITE`) é seguido por um pedido de resposta da SLP (seta SLP-OBC, transmissão `READ`) a fim de checar que todas os comandos foram corretamente recebidos pela SLP. Esta foi uma decisão de projeto uma vez que o I^2C não possui nenhum método de detecção e/ou correção de falhas, assim decorrente de uma das primeiras contribuições do FEM na fase de desenvolvimento, que levou a equipe de desenvolvimento a mudar a comunicação para prevenir desencontros durante a troca de mensagens.

Após ligar a SLP, o OBC envia a exata data e hora como primeiro dado a ser armazenado no *buffer* da SLP e habilita a aquisição de dados. Isto é feito para que se possa controlar os dados adquiridos uma vez que a equipe de solo necessita saber a posição orbital do satélite durante cada medida da SLP.

Estando a SLP no modo de aquisição de dados, qualquer comando enviado pelo mestre será seguido de uma resposta da SLP, referida por `SLPBUSY` na Figura 6.1,

Figura 6.1 - Diagrama de sequência do OBC e SLP



FONTE: Próprio Autor

até que a aquisição de dados tenha terminado. O tempo que o OBC deve aguardar foi definido pela equipe de desenvolvimento da SLP e é em torno de cinco minutos (um *buffer* de 300 kB com uma taxa de aquisição de dados de 1 kB/s). Durante esse tempo o OBC está livre para trabalhar com outros subsistemas e a SLP, terminada a aquisição de dados, aguardará pelo comando do OBC para início da transferência de dados.

Antes da transferência, o OBC indica o número de bytes a serem transferidos e o índice na pilha de dados em que a transferência irá iniciar. Assim que todos os dados presentes no buffer tenham sido transferidos um novo ciclo se reinicia, reescrevendo todo o *buffer* da SLP e reiniciando a aquisição de dados.

A robustez implementada nesses modelos segue a lógica de que dada a identificação de uma falha, pelo mestre, durante uma troca de mensagens, este tenta o mesmo comando novamente.

Para validar o FEM, alguns conjuntos de casos de teste foram executados, *ad hoc* e gerados automaticamente por terceiros, com o intuito de explorar as funcionalidades do FEM. As seguintes seções buscam apresentar os resultados e discussão dessas execuções.

6.2 Resultados: Casos de Teste *Ad hoc*

Com os SUT modelados e carregados em suas respectivas placa UNO e o FEM, também, modelado e carregado em sua plataforma, pôde-se avaliar seu funcionamento aplicando nele casos de testes específicos.

Casos de teste *ad hoc* foram gerados e executados com a finalidade de validar o uso da solução FEM.

Este conjunto de casos de teste foram concebidos pela mesma equipe que desenvolveu o FEM focando nas funcionalidades do mesmo com o objetivo de explorar as funções implementadas.

A Tabela 6.1 representa um dos casos de teste para o FEM. Neste caso específico, o diagrama de sequência de interoperabilidade entre OBC e SLP é executado repetidas vezes até o fim do roteiro de falhas proposto.

A primeira parte da tabela representa a identificação do caso de teste (i.e. testcase03), os SUT envolvidos no teste (i.e. OBC e SLP) e o número de falhas a serem

Tabela 6.1 - Caso de Teste *Ad Hoc* para validar o uso do FEM em teste do OBC e SLP

ID	SUT1	SUT2	NFAULTS
testcase03	OBC	SLP	10 faults

WHERE	WHEN	WHAT	HOW
write	3	*DELAY*	5 ms
read	6	*LOCK*	100 ms
write	15	*DELAY*	5 ms
read	21	*LOCK*	100 ms
read	35	*LOCK*	100 ms
write	36	*BITFLIP*	LSB 2
write	40	*DELAY*	5 ms
read	44	*CHANGE*	0x02
write	50	*DELAY*	5 ms
write	55	*DELAY*	5 ms

Tabela 6.2 - Resumo das funções do FEM em teste em mensagens de WRITE, *Ad hoc*

MASTER Tx	FAULT	PARAM	DELAY	SLAVE Rx
0xF0	null	N/A	888 us	F0
0x3C	delay	5 ms	5611 us	FF
0x3C	delay	20 ms	5602 us	FF
0xF0	bitflip	lsb 2	1453 us	0xFB
0xF0	delay	5ms	5534 us	FF
0x3C	delay	5 ms	5559 us	FF
0x3C	delay	5 ms	5661	FF

injetadas (i.e. NFAULTS=10) como referência para a segunda parte da tabela onde se encontra o roteiro de falhas. A escolha de QUANDO injetar a falha foi completamente randômica.

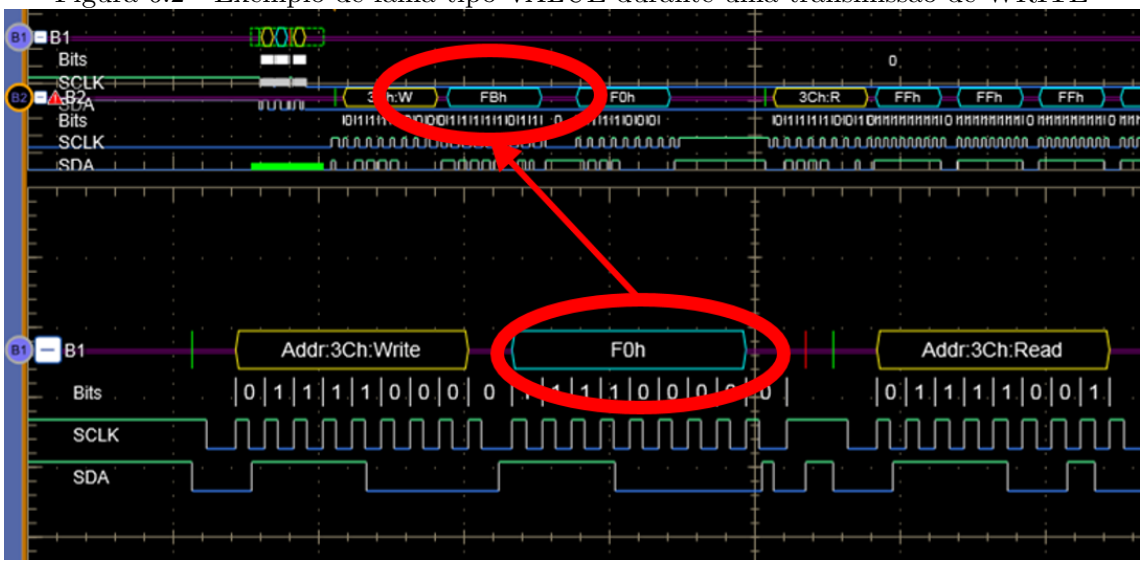
As Tabelas 6.2 e 6.3 mostram a execução do teste com o FEM a fim de analisar o seu comportamento e intrusividade na comunicação dos SUT, respectivamente nas demandas de `WRITE` e `READ` do subsistema mestre.

Os resultados apresentados foram observados pelo monitor serial (via PC) e usando um osciloscópio com um decodificador I^2C como demonstrado na Figura 6.2. Uma falha do tipo VALUE foi injetada na tentativa do OBC enviar `WRITE 0xF0` que se torna `0xFB` devido a um *bit-flip* no `LSB 2`, terceiro *bit* menos significativo no *byte* transmitido.

Tabela 6.3 - Resumo das funções do FEM em teste em mensagens de READ, *Ad hoc*

SLAVE Tx	FAULT	PARAM	DELAY	MASTER Rx
0xFA	null	N/A	1101 us	FA
0xFA	delay	5 ms	6655 us	0xFF
0xFA	lock	100 ms	101 ms	0xFF
0xFA	lock	100 ms	101 ms	0xFF
0xFA	change	0x02	1115 us	0x02

Figura 6.2 - Exemplo de falha tipo VALUE durante uma transmissão de WRITE



FONTE: Próprio Autor

Tabelas apresentando exmplos de outros casos de teste *ad hoc* podem ser encontrados no Anexo .

6.3 Resultados: Casos de Teste Gerados Automaticamente

Este conjunto de casos de teste para a validação do FEM foram gerados por terceiros, uma equipe externa ao desenvolvimento do FEM e dos SUT (BATISTA et al., 2019).

Contando apenas com os documentos de descrição dos subsistemas sob teste (OBC e SLP) e os modelos de falhas do FEM instanciado para o barramento I^2C, Modelos de Estados de cada subsistema e foram construídos e, a partir deles, gerados automaticamente casos de teste para os SUT.

Utilizando a lógica da Subseção 6.2, as Tabelas 6.4, 6.5 e 6.6 representam os resultados de um caso de teste gerado automaticamente e carregado no FEM assim como

Tabela 6.4 - Exemplo de caso de teste gerado automaticamente para o FEM

ID	SUT1	SUT2	NFAULTS
testcase1.0	OBC	SLP	5 faults

WHERE	WHEN	WHAT	HOW
write	3	*DELAY*	500 ms
read	4	*BITFLIP*	LSB 2
read	6	*DELAY*	500 ms
write	7	*LOCK*	100 ms
read	8	*DELAY*	500 ms

Tabela 6.5 - Resumo das funções do FEM em teste em mensagens de WRITE

MASTER Tx	FAULT	PARAM	DELAY	SLAVE Rx
0xF0	null	N/A	1044 us	F0
0xF1	delay	500 ms	500 ms	FF
0xF2	lock	100 ms	100 ms	FF

Tabela 6.6 - Resumo das funções do FEM em teste em mensagens de READ

SLAVE Tx	FAULT	PARAM	DELAY	MASTER Rx
0xFA	null	N/A	868 us	0xFA
0xFA	bitflip	LSB 2	1536 us	0xFE
0xFA	delay	100 ms	101 ms	0xFF
0xFA	delay	100 ms	101 ms	0xFF

o esperado de cada demanda do dispositivo mestre, `READ` e `WRITE`.

Mais detalhes quanto à geração destes e outros casos de teste, bom como quanto aos Modelos de Estados elaborados para cada subsistema e FEM podem ser encontrados em (BATISTA et al., 2019).

6.4 Discussão

A execução dos casos de teste em aplicação real usando o FEM prototipado demonstrou a efetividade do *framework* proposto. Aspectos da arquitetura do *framework* proposta para o injetor de falhas, importantes para sua instanciação, foram observados qualitativamente: (i) **representatividade**, uma vez que o *faultload* busca representar modelos reais de falhas, situações típicas na operação de um satélite puderam ser consideradas no modelo de falhas de robustez implementado no protótipo;

(ii) **portabilidade**, o *framework* do FEM, como um todo, demonstrou-se altamente portátil. Percebeu-se que com pequenas modificações é possível mudar de plataforma física; (iii) **flexibilidade**, FEM é uma solução modular e abstrata ao testador, diferentes modelos de falhas podem ser implementa-dos e usados na geração automática de casos de testes requerendo mínimas modificações no FEM; (iv) **intrusividade**, a implementação do FEM para I^2C demonstrou ser como uma solução não intrusiva mesmo utilizando um processador mais lento como o ATMega328P; e (v) **eficiência**; o FEM prototipado mostrou-se eficiente como uma ferramenta para alcançar resultados úteis, apesar de a análise de cobertura dos testes ainda estar a cargo do testador, bem como a definição do propósito de teste e o número de experimentos envolvendo (ou não) injeção de falhas, ativação e propagação de falhas/erros/defeitos.

7 CONCLUSÕES

A verificação de robustez de sistemas intensivos em *software* por meio de testes é crucial para o sucesso das missões espaciais, uma vez que a operação do satélite depende do correto funcionamento de seus serviços em órbita. Assim, além das restrições ambientais e do veículo lançador, as boas práticas da engenharia de sistemas espaciais recomendam verificar e validar durante o processo de desenvolvimento os requisitos operacionais dos subsistemas a bordo do satélite.

Visando apoiar os testes de interoperabilidade e robustez de subsistemas intensivos em software embarcados em missões de nanosatélites, o *framework* do FEM foi proposto. Uma ferramenta foi prototipada a partir da arquitetura do FEM proposto, instanciada como parte do sistema de testes do NanoSatC-BR2. Mais precisamente, o protótipo do FEM dará suporte na verificação de robustez do computador de bordo quando interagindo com as cargas úteis do nanossatélite. Esta ferramenta poderá ser utilizada tanto nos cenários de desenvolvimento: (i) MIL antecipando questões de robustez no ciclo de desenvolvimento dos subsistemas interoperantes; (ii) HIL apoiando a verificação de interoperabilidade dos subsistemas comunicantes durante os testes de integração.

No cenário MIL, códigos gerados a partir dos modelos comportamentais dos sistemas comunicantes são embarcados em Arduinos. A injeção de falhas no canal de comunicação realizada pelo FEM emula possíveis situações não esperadas. Desta forma, permite à equipe de desenvolvimento identificar de forma rápida possíveis eventos e situações que possam vir a causar o *crash* de subsistemas, corrigindo a especificação dos seus requisitos funcionais ainda em fase de modelagem.

Já no cenário HIL, o uso do FEM permite a verificação de quaisquer requisitos de robustez especificado e implementado na versão final do subsistema alvo entregue pelo desenvolvedor para integração no nanossatélite.

A avaliação da robustez através de testes em sistemas intensivos em software é crucial para o sucesso das missões espaciais, uma vez que a completa operação do satélite depende do correto funcionamento de seus serviços.

O *framework* do FEM foi, assim, instanciado como parte do sistema de testes do NanoSatC-BR2, que poderá ser reutilizado nos testes de integração de todos os nanossatélites desenvolvidos pelo INPE, compatíveis, logicamente, com a família NanoSatC-BR.

Vale a pena ressaltar que dado o tempo de desenvolvimento reduzido e a qualidade requerida a cada missão de nanossatélite, o uso de placas Arduino e outros componentes COTS permitem a evolução do sistema de testes com um custo adicional mínimo.

Também vale salientar a "mínima"intrusividade do FEM prototipado em termos de atrasos no canal de comunicação dado o uso de placas COTS.

Assim, o *framework* FEM apresenta uma solução capaz de migrar entre plataformas, buscando se apresentar como resposta ao testes e integração em sistemas distribuídos, podendo, independente de interface ser moldado à necessidade. Implementações desse *framework* podem ser buscadas com o intuito de adicionar funcionalidades ao FEM. Com o devido poder de processamento, novas funcionalidades podem ser adicionadas facilmente à instanciação do *framewoek*. Ao mesmo tempo em que se pode especificar e modelar um FEM com o intuito de avaliar questões específicas, como tempo, por exemplo, modelando e codificando diferentes tipos de falhas relacionadas a temporização e sincronismo.

Ainda cabe a ressalva que o FEM não é um novo tipo de injetor de falhas, ou tenta reinventar conceitos. De fato o *framework* proposto, inspirado no SWFI, concretiza uma arquitetura genérica apropriada para ambiente de desenvolvimento de nanossatélites que utilizam o padrão CubeSat. Protótipos do FEM podem ser implementados a partir da instanciação da arquitetura proposta, conforme demonstrado.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDO, T.; YATSU, H.; HISAZUMI, K.; FUKUDA, A.; MATSUMOTO, M.; MICHIURA, Y. Reference model of specifications toward independent verification and validation. In: **TENCON REGION 10 CONFERENCE**. [S.l.]: Proceedings... IEEE, 2015. p. 1–3. 3

ARLAT, J.; AGUERA, M.; AMAT, L.; CROUZET, Y.; FABRE, J.-C.; LAPRIE, J.-C.; MARTINS, E.; POWELL, D. Fault injection for dependability validation: a methodology and some applications. **IEEE Transactions on Software Engineering**, v. 16, n. 2, p. 166–182, 1990. 11, 15

AVIZIENIS, A.; LAPRIE, J.-C.; RANDELL, B.; LANDWEHR, C. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, 2004. 11

BATISTA, C. L. G.; WELLER, A. C.; MARTINS, E.; MATTIELLO-FRANCISCO, F. Towards increasing nanosatellite subsystem robustness. **Acta Astronautica**, Elsevier, v. 156, p. 187–196, 2019. 5, 41, 42

BAX, M. P. Design science: filosofia da pesquisa em ciência da informação e tecnologia. **Ciência da Informação**, v. 42, n. 2, 2013. 4, 5

BONFIGLIO, V.; MONTECCHI, L.; IRRERA, I.; ROSSI, F.; LOLLINI, P.; BONDAVALLI, A. Software faults emulation at model-level: towards automated software fmea. In: **DEPENDABLE SYSTEMS AND NETWORKS WORSHOP**. 2015: Proceedings... IEEE, 2015. p. 133–140. 24

BÜRGER, E. E.; LOUREIRO, G.; BOHRER, R. Z. G.; COSTA, L. L.; HOFFMANN, C. T.; ZAMBRANO, D. H.; JAENISCH, G. P. Development and analysis of a brazilian Cubesat structure. In: **INTERNATIONAL CONGRESS OF MECHANICAL ENGINEERING, 22**. 2013: Proceedings..., 2013. 1

CALTECH, J. P. L. **Mars Cube One (MarCO)**. 2018. Acessado em: 20 Dez 2018. Disponível em: <https://www.jpl.nasa.gov/cubesat/missions/marco.php>. 8

CARRARA, V.; JANUZI, R. B.; MAKITA, D. H.; SANTOS, L. F. d. P.; SATO, L. S. The ITASAT CubeSat development and design. **Journal of Aerospace Technology and Management**, v. 9, n. 2, p. 147–156, 2017. 1

CARREIRA, J. V.; COSTA, D.; SILVA, J. G. Fault injection spot-checks computer system dependability. **IEEE Spectrum**, v. 36, n. 8, p. 50–55, 1999. 11, 18

CARVALHO, M. J. M. de; LIMA, J. S. dos S.; JOTHA, L. dos S.; AQUINO, P. S. de. Conasat-constelação de nano satélites para coleta de dados ambientais. **SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 15**, Anais... São José dos Campos, 2013, p. 13–18, 2013. 1

CLARK, J.; PRADHAN, D. Fault injection: a method for validating computer-system dependability. **Computer**, v. 28, n. 6, p. 47–56, jun 1995. ISSN 00189162. 11

CONCEICAO, C. A.; MATTIELLO-FRANCISCO, F.; BATISTA, C. L. Dependability verification of nanosatellite embedded software supported by a reusable test system. In: **LATIN-AMERICAN SYMPOSIUM ON DEPENDABLE COMPUTING, 7**. 2016: Proceedings..., 2016. p. 157–163. ISBN 978-1-5090-5120-5. 21, 27, 28

COTRONEO, D.; NATELLA, R. Fault injection for software certification. **IEEE Security & Privacy**, v. 11, n. 4, p. 38–45, 2013. 13

DESMOULIN, A.; VIHO, C. Automatic interoperability test case generation based on formal definitions. In: **INTERNATIONAL WORKSHOP ON FORMAL METHODS FOR INDUSTRIAL CRITICAL SYSTEMS**. 2007: Proceedings... Springer, 2007. p. 234–250. 9

DIAZ, M. A. et al. New opportunities offered by cubesats for space research in latin america: the SUCHAI project case. **Advances in Space Research**, v. 58, n. 10, p. 2134–2147, 2016. 1

DOBLER, R. J.; CECHIN, S.; WEBER, T.; NETTO, J. A Software Fault Injector to Validate Implementations of a Safety Communication Protocol. In: **2016 Seventh Latin-American Symposium on Dependable Computing (LADC)**. [S.l.]: IEEE, 2016. p. 35–42. ISBN 978-1-5090-5120-5. 18, 19

DOWSON, M. The ariane 5 software failure. **ACM SIGSOFT Software Engineering Notes**, v. 22, n. 2, p. 84, 1997. 16

FIGUEIRÓ, G. SERPENS CubeSat mission. In: **ANNUAL CUBESAT DEVELOPERS WORKSHOP**. 2014, San Luis Obispo: Proceedings..., 2014. 1

FORSBERG, K.; CO-PRINCIPALS, H. M. 4 system engineering for faster, cheaper, better. In: INCOSE. **INTERNATIONAL SYMPOSIUM**. 1999, 1999. v. 9, p. 924–932. 3

GHOSH, S.; MATHUR, A. P.; HORGAN, J. R.; LI, J. J.; WONG, W. E. Software fault injection testing on a distributed system–a case study. **INTERNATIONAL QUALITY WEEK EUROPE, 1**, Proceedings..., Brusssels, Belgium, 1997, 1997. 24

GOOD, A. **NASA's first image of Mars from a CubeSat**. 2018. Acessado em: 20 Dez 2018. Disponível em: <https://www.jpl.nasa.gov/news/news.php?feature=7263>. 8

_____. **A pale blue dot, as seen by a CubeSat**. 2018. Acessado em: 20 Dez 2018. Disponível em: <https://www.nasa.gov/feature/jpl/a-pale-blue-dot-as-seen-by-a-cubesat>. 8

GUNNEFLO, U.; KARLSSON, J.; TORIN, J. Evaluation of error detection schemes using fault injection by heavy-ion radiation. In: **The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers**. [S.l.]: IEEE Comput. Soc. Press, 1989. p. 340–347. ISBN 0-8186-1959-7. 15, 16

HEVNER, A. R. A three cycle view of design science research. **Scandinavian Journal of Information Systems**, v. 19, n. 2, p. 4, 2007. 4

HEVNER, A. R.; MARCH, S. T.; PARK, J.; RAM, S. Design science in information systems research. **Management Information Systems Quarterly**, v. 28, n. 1, p. 6, 2008. 4

HODGES, R. E.; CHAHAT, N.; HOPPE, D. J.; VACCHIONE, J. D. A deployable high-gain antenna bound for mars: developing a new folded-panel reflectarray for the first CubeSat mission to Mars. **IEEE Antennas and Propagation Magazine**, v. 59, n. 2, p. 39–49, 2017. 8

HSUEH, M.-C.; TSAI, T. K.; IYER, R. K. Fault injection techniques and tools. **IEEE Computer**, v. 30, n. 4, p. 75–82, 1997. 11, 14, 15

IEEE. **IEEE 610.12-1990: standard glossary of software engineering terminology**. [S.l.: s.n.], 1990. 94 p. 3, 9

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. **Plano Diretor do INPE 2016-2019**. São José dos Campos: INPE, 2016. 2

JEONG, Y. S.; LEE, S. M.; LEE, S. E. A Survey of Fault-Injection Methodologies for Soft Error Rate Modeling in Systems-on-Chips. **Bulletin of Electrical Engineering and Informatics**, v. 5, n. 2, p. 169–177, jun 2016. ISSN 2302-9285. 15

JOHANSSON, A.; SURI, N. Error propagation profiling of operating systems. In: **INTERNATIONAL CONFERENCE ON DEPENSABLE SYSTEMS AND NETWORKS**. 2005: Proceedings... IEEE, 2005. p. 86–95. 18

KAKSONEN, R. **A functional method for assessing protocol implementation security**. [S.l.]: Technical Research Centre of Finland, 2001. 3

KARLSSON, J.; FOLKESSON, P.; ARLAT, J.; Yves Crouzet; LEBER, G.; REISINGER, J. Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture. **Dependable Computing and Fault Tolerant Systems 10**, p. 267–288, 1998. 16

KASLOW, D.; ANDERSON, L.; IWATA, C.; ASUNDI, S.; AYRES, B.; THOMPSON, R. Developing and distributing a CubeSat Model-Based Systems Engineering (MBSE) reference model. In: **SPACE SYMPOSIUM**. 2015: Proceedings..., 2015. p. 1–14. 3

KASLOW, D.; AYRES, B.; CAHILL, P. T.; HART, L.; YNTEMA, R. A Model-Based Systems Engineering (MBSE) approach for defining the behaviors of CubeSats. In: **AEROSPACE CONFERENCE**. 2017: Proceedings... IEEE, 2017. p. 1–14. 3

KASLOW, D.; AYRES, B.; CAHILL, P. T.; HART, L.; LEVI, A. G.; CRONEY, C. Developing an MBSE CubeSat reference model – interim status #4. In: **SPACE AND ASTRONAUTICAL FORUM AND EXPOSITION**. 2018: Proceedings... AIAA, 2018. p. 5328. 3, 8

KLESH, A.; KRAJEWSKI, J. MarCO: CubeSats to Mars in 2016. In: **ANNUAL CONFERENCE ON SMALL SATELLITES, 29**. 2015: Proceedings... AIAA/USU, 2015. 8

_____. MarCO: Mars Cube One – lessons learned from readying the first interplanetary Cubesats for flight. In: **LUNAR AND PLANETARY SCIENCE CONFERENCE**. 2018: Proceedings..., 2018. v. 49. 8, 9

KONECNY, G. Small satellites–a tool for earth observation? In: **ISPRS CONGRESS, 20**. 2004: Proceedings..., 2004. v. 4, p. 12–23. 7

KOOLI, M.; KADDACHI, F.; NATALE, G. D.; BOSIO, A.; BENOIT, P.; TORRES, L.; DI NATALE, G.; BOSIO, A.; BENOIT, P.; TORRES, L. Computing reliability: on the differences between software testing and software fault injection techniques. **Microprocessors and Microsystems**, v. 50, p. 102–112, 2017. 14, 15

KULU, E. **Nanosatellite and CubeSat database**. 2018. Online. Acessado em: 10 Jan 2019. Disponível em: <http://www.nanosats.eu>. 1

LEWIS, R. O. **Independent verification and validation: a life cycle engineering process for quality software**. [S.l.]: John Wiley & Sons, 1992. 9

MABROUK, E. **What are SmallSats and CubeSats?** 2015. Acessado em: 20 Dez 2018. Disponível em: <https://www.nasa.gov/content/what-are-smallsats-and-cubesats>. 7

MARIÑO, G. G. C.; BÜRGER, E. E.; LOUREIRO, G.; BOGOSSIAN, O. L. Mission analysis for a remote sensing CubeSat mission over the Amazon rainforest. In: **LATIN AMERICAN CUBESAT WORKSHOP, 2**. 2017: Proceedings... IAA, 2017. p. 1–14. 1

MATTIELLO-FRANCISCO, F.; MARTINS, E.; CAVALLI, A. R.; YANO, E. T. Inrob: 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. 11

MATTIELLO-FRANCISCO, M. d. F. **InRob–Uma abordagem para teste de Interoperabilidade e de Robustez de subsistemas de tempo-real intensivos em software**. Engenharia da Computćão. Tese (Doutorado) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 2009. 10, 11, 19, 21

MEHRPARVAR, A.; PIGNATELLI, D.; CARNAHAN, J.; MUNAKAT, R.; LAN, W.; TOORIAN, A.; HUTPUTANASIN, A.; LEE, S. **CubeSat Design Specification Rev 13**. Cal Poly: San Luis, Obispo, US, 2014. 1

MOTA, D. F. M. **OPENOBC: uma arquitetura de um computador de bordo open source e de baixo custo para o padrão CUBESAT**. 75 p. Engenharia de Teleinformática. Dissertação (Mestrado) — Universidade Federal do Ceará, 2017. 1

NAG, S. Sensor webs of agile, small satellite constellations and unmanned aerial vehicles with satellite-to-air communication links. In: **LATIN AMERICAN**

SYMPOSIUM ON SMALL SATELLITES, 1. 2017: Proceedings... IAA, 2017. 1

NATELLA, R.; COTRONEO, D.; MADEIRA, H. S. Assessing dependability with software fault injection : a survey. **ACM Computing Surveys**, v. 48, n. 3, p. 44:1–44:55, 2016. ISSN 03600300. 14, 16, 17, 18

NATIONAL ACADEMIES OF SCIENCES ENGINEERING AND MEDICINE; DIVISION ON ENGINEERING AND PHYSICAL SCIENCES; SPACE STUDIES BOARD. Achieving science with CubeSats: thinking inside the box. **Nap 23503**, p. 130, Set 2016. ISSN 00741795. 1

NIBERT, J.; HERNITER, M. E.; CHAMBERS, Z. Model-based system design for mil, sil, and hil. **World Electric Vehicle Journal**, v. 5, n. 4, p. 1121–1130, 2012. 21

O'CONNOR, B. **Software safety guidebook**. Washington: NASA, 2004. (NASA-GB-8719.13). 13

PALAMIN, D.; MATTIELLO-FRANCISCO, M. d. F. Modelling of the interoperability between on-board computer and payloads of the nanosatc-br2 with support of the uppaal tool. In: **IAA LATIN AMERICAN SYMPOSIUM ON SMALL SATELLITES. 1**. 2017: Proceedings... IAA, 2017. 37

PHILIPS SEMICONDUCTORS. **AN10216-01: I2C manual**. [S.l.], 2003. 28, 29

POGHOSYAN, A.; GOLKAR, A. CubeSat evolution: analyzing CubeSat capabilities for conducting science missions. **Progress in Aerospace Sciences**, v. 88, p. 59–83, 2016. 1

SANTOS, W. A. D.; MOURA, C.; YAMAGUTI, W.; SILVA, M. Space education and public outreach for aerospace engineering in a brazilian perspective. In: **INTERNATIONAL SYMPOSIUM ON SPACE TECHNOLOGY ANS SCIENCE, 28**. 2011, Japan: Proceedings..., 2011. 1

SCHUCH, N. J.; DURÃO, O. S. C.; SILVA, M. R. da; MATTIELLO-FRANCISCO, F.; SILVA, A. L. da. NanoSatC-BR status - a joint cubesat-based program developed by INPE and UFSM. In: **CONFERENCE ON UNIVERSITY SATELLITE MISSIONS AND CUBESAT WORKSHOP, 4**. 2017: Proceedings..., 2017. 1, 27

SELVA, D.; KREJCI, D. A survey and assessment of the capabilities of Cubesats for Earth observation. **Acta Astronautica**, v. 74, p. 50–68, 2012. 1

SHOKRY, H.; HINCHEY, M. Model-based verification of embedded software. **Computer**, IEEE Computer Society, v. 42, p. 53–59, 2009. 21

STERPONE, L.; VIOLANTE, M. A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas. **IEEE Transactions on Nuclear Science**, v. 54, n. 4, p. 965–970, 2007. 15

SWARTWOUT, M. Secondary spacecraft in 2016: why some succeed (and too many do not). In: **AEROSPACE CONFERENCE**. 2016: Proceedings... IEEE, 2016. p. 1–13. 2

_____. **Michael Swartwout web page**. 2018. Disponível em: <https://sites.google.com/a/slu.edu/swartwout/home>. 1, 2

_____. You say "PicoSat", i say "CubeSat": developing a better taxonomy for secondary spacecraft. In: **AEROSPACE CONFERENCE**. 2018: Proceedings... IEEE, 2018. p. 1–17. 2

TRETMANS, J.; KARS, P.; BRINKSMA, E. Protocol conformance testing: a formal perspective on iso is-9646. In: **INTERNATIONAL WORKSHOP ON PROTOCOL TEST SYSTEMS, 4**. 1991: Proceedings... North-Holland Publishing Co., 1991. p. 131–142. 22

TWIGGS, R.; PUIG, J.; TURNER, C. Cubesat: the development and launch support infrastructure for eighteen different satellite customers on one launch. In: **SMALL SATELLITE CONFERENCE PROCEEDINS**. 2001: Proceedings..., 2001. p. 1–5. 1

VOAS, J. Fault injection for the masses. **Computer**, n. 12, p. 129–130, 1997. 3

VULI, P.; BADALAMENT, M.; JAIKAMAL, V. **Maximizing test asset re-use across MIL, SIL, and HIL development platforms**. [S.l.: s.n.], 2010. 21

WEBER, T. S. Tolerância a falhas: conceitos e exemplos. Universidade Federal do Rio Grande do Sul, Porto Alegre, p. 24, 2003. (Apostila do Programa de Pós-Graduação). 12

WELLER, A. C.; MARTINS, E.; MATTIELLO-FRANCISCO, F. Inrob-uml: uma abordagem para testes de interoperabilidade e robustez baseados em modelos. **SAST 2015**, p. 71, 2015. 11, 19, 20

WIERINGA, R. Design science as nested problem solving. In: **INTERNATIONAL CONFERENCE ON DESIGN SCIENCE**

RESEARCH IN INFORMATION SYSTEMS AND TECHNOLOGY. 4. 2009: Proceedings... ACM, 2009. p. 8. 4

WIERINGA, R. J.; HEERKENS, J. M. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. **Requirements Engineering**, v. 11, n. 4, p. 295–307, 2006. 4

ZARANDI, H. R.; MIREMADI, S. G.; EJLALI, A. Dependability analysis using a fault injection tool based on synthesizability of hdl models. In: **INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS, 18**. 2003: Proceedings... IEEE, 2003. p. 485–492. 15

ZUFFADA, C.; CASTILLO-ROGEZ, J.; QUADRELLI, M. A consolidated view of science with smallsats/cubesats. In: **COSPAR SCIENTIFIC ASSEMBLY, 42**. 2018: Proceedings..., 2018. v. 42. 9

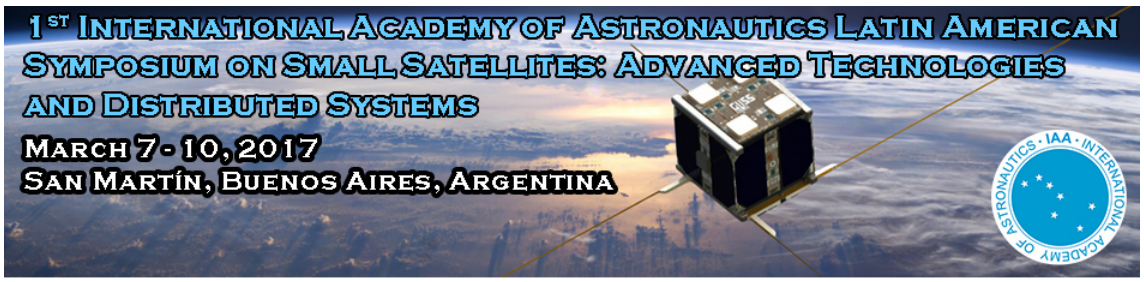
APÊNDICE A - PUBLICAÇÕES e CONGRESSOS

A.1 Ago 2016 - *Workshop* em Engenharia e Tecnologias Espaciais/INPE - São José dos Campos, Brasil

Título	Injetor de Falhas em Ambiente de Integração e Testes de Subsistemas Espaciais para Nanossatelites
Autor	Carlos Leandro Gomes Batista, Maria de Fátima Mattiello-Francisco
Resumo	Dada a evolução de projetos envolvendo nanossatélites, uma crescente demanda vem sendo observada ao redor do mundo em programas espaciais. Esse crescimento no acesso ao espaço trouxe consigo um relaxamento no que abrange aos processos de verificação e validação (V\&V) em tais missões de rápido desenvolvimento e baixo custo. Este trabalho busca apresentar, a partir de um sistema de testes, um cenário para mimetização de falhas em subsistemas operacionais utilizando de um injetor de falhas. Tal técnica amplamente difundida em ambientes de software, apresenta aqui uma solução evolutiva de caráter adaptativo com o intuito de viabilizar um processo de validação baseada em conceitos de interoperabilidade e robustez capaz tornar medidas de dependabilidade mais alcançáveis em projetos espaciais de curto ciclo de vida, tais como missões *CubeSat*.

A.2 Out 2016 - IEEE Latin American Symposium on Dependable Computing - Cali, Colômbia

A.3 Mar 2017 - IAA Latin American Symposium on Small Satellites - Buenos Aires, Argentina



Session 11: SATELLITE TECHNOLOGIES: AIT

Chair: C. Barrientos, CONAE

» **Using Fault Injection on the Nanosatellite Subsystems Integration Testing** – 9:00 – 9:20 | Abstract | Presentation

» Carlos L G Batista (1); Andre Corsetti (2); Fatima Mattiello-Francisco (1) –

» (1) Brazilian National Institute for Space Research – INPE, (2) Omega 7 Systems

A.4 Mar 2018 - IEEE Latin American Test Symposium - São Paulo, Brasil

On the use of a failure emulator mechanism at nanosatellite subsystems integration tests

3 Author(s) Carlos L. G. Batista ; Eliane Martins ; Maria de Fátima Mattiello-Francisco **View All Authors**

39 Full Text Views

A.5 Nov 2018 - Acta Astronautica - Qualis A2 / IF 2.227

Acta Astronautica

Available online 17 November 2018

In Press, Corrected Proof

Towards increasing nanosatellite subsystem robustness

Carlos Leandro Gomes Batista [a], Anderson Coelho Weller [b], Eliane Martins [b], Fátima Mattiello-Francisco [a]

Show more

https://doi.org/10.1016/j.actaastro.2018.11.011

ANEXO A - CASOS DE TESTE *AD HOC*

Aqui são apresentados o roteiro para os casos de teste gerados de forma manual, *ad hoc*.

A Tabela A.1 demonstra um roteiro de testes inicial avaliando falha temporal na comunicação com diferentes atrasos.

Tabela A.1 - Roteiro de Teste *Ad Hoc* 00

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase00	OBC	SLP	15 faults

WHERE	**WHEN**	**WHAT**	**HOW**
write	2	*DELAY*	500 ms
write	3	*DELAY*	100 ms
read	2	*DELAY*	10 ms
write	4	*DELAY*	50 ms
write	5	*DELAY*	100 ms
write	6	*DELAY*	500 ms
write	7	*DELAY*	100 ms
read	4	*DELAY*	50 ms
write	8	*DELAY*	100 ms
read	5	*DELAY*	500 ms
write	9	*DELAY*	10 ms
write	10	*DELAY*	500 ms
write	11	*DELAY*	100 ms
write	12	*DELAY*	50 ms
write	13	*DELAY*	500 ms

A Tabela A.2 demonstra algumas falhas utilizadas para validar diferentes funcionalidades do FEM.

A Tabela A.3 demonstra um roteiro de testes avaliando falhas do tipo VALOR, *bitflip* e *change*.

A Tabela A.4 demonstra um roteiro de testes com diferentes falhas injetadas.

A Tabela A.5 demonstra um roteiro de testes com diferentes falhas injetadas.

A Tabela A.6 demonstra um roteiro de testes maior, em comparação ao anterior, com diferentes falhas injetadas.

Tabela A.2 - Roteiro de Teste *Ad Hoc* 01

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase01	OBC	SLP	5 faults

WHERE	**WHEN**	**WHAT**	**HOW**
write	3	*DELAY*	500 ms
read	4	*BITFLIP*	LSB 2
read	6	*DELAY*	500 ms
write	7	*LOCK*	100 ms
read	8	*DELAY*	500 ms

Tabela A.3 - Roteiro de Teste *Ad Hoc* 02

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase02	OBC	SLP	8 faults

WHERE	**WHEN**	**WHAT**	**HOW**
read	4	*BITFLIP*	LSB 2
write	5	*BITFLIP*	LSB 2
read	6	*BITFLIP*	LSB 2
write	8	*BITFLIP*	LSB 2
write	11	*BITFLIP*	LSB 2
read	15	*CHANGE*	0x02
write	25	*CHANGE*	0x02
write	26	*CHANGE*	0x02

Tabela A.4 - Roteiro de Teste *Ad Hoc* 03

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase03	OBC	SLP	10 faults

WHERE	**WHEN**	**WHAT**	**HOW**
write	3	*DELAY*	5 ms
read	6	*LOCK*	100 ms
write	15	*DELAY*	5 ms
read	21	*LOCK*	10 ms
read	35	*LOCK*	10 ms
write	36	*BITFLIP*	LSB 2
write	40	*DELAY*	5 ms
read	44	*CHANGE*	0x02
write	50	*DELAY*	5 ms
write	55	*DELAY*	5 ms

Tabela A.5 - Roteiro de Teste *Ad Hoc* 04

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase04	OBC	SLP	9 faults

WHERE	**WHEN**	**WHAT**	**HOW**
read	6	*LOCK*	100 ms
write	15	*DELAY*	5 ms
read	21	*LOCK*	10 ms
read	35	*LOCK*	10 ms
write	36	*BITFLIP*	LSB 2
write	40	*DELAY*	5 ms
read	44	*CHANGE*	0x02
write	50	*DELAY*	5 ms
write	65	*DELAY*	5 ms

Tabela A.6 - Roteiro de Teste *Ad Hoc* 05

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase05	OBC	SLP	14 faults

WHERE	**WHEN**	**WHAT**	**HOW**
write	3	*DELAY*	500 ms
read	4	*BITFLIP*	LSB 2
read	6	*DELAY*	500 ms
write	7	*LOCK*	100 ms
read	8	*DELAY*	500 ms
read	10	*LOCK*	100 ms
write	15	*DELAY*	5 ms
read	21	*LOCK*	10 ms
read	35	*LOCK*	10 ms
write	36	*BITFLIP*	LSB 2
write	40	*DELAY*	5 ms
read	54	*CHANGE*	0x02
write	60	*DELAY*	5 ms
write	75	*DELAY*	5 ms

ANEXO B - CASOS DE TESTE GERADOS AUTOMATICAMENTE

Os casos de teste gerados automaticamente, assim com seus respectivos roteiros, possuem propósito inerente ao desejo do testador, caso não coberto por este trabalho.

Tabela B.1 - Roteiro de Falhas para o Caso de Teste 1.0 gerado automaticamente

ID	SUT1	SUT2	NFAULTS
testcase1.0	OBC	SLP	5 faults

WHERE	WHEN	WHAT	HOW
write	3	*DELAY*	500 ms
read	4	*BITFLIP*	LSB 2
read	6	*DELAY*	500 ms
write	7	*LOCK*	100 ms
read	8	*DELAY*	500 ms

Tabela B.2 - Roteiro de Falhas para o Caso de Teste 1.1 gerado automaticamente

ID	SUT1	SUT2	NFAULTS
testcase1.1	OBC	SLP	4 faults

WHERE	WHEN	WHAT	HOW
write	2	*DELAY*	500 ms
write	3	*BITFLIP*	LSB 2
write	5	*BITFLIP*	LSB 2
write	6	*DELAY*	500 ms

Tabela B.3 - Roteiro de Falhas para o Caso de Teste 1.2 gerado automaticamente

ID	SUT1	SUT2	NFAULTS
testcase1.2	OBC	SLP	5 faults

WHERE	WHEN	WHAT	HOW
read	1	*CHANGE*	0x7F
write	3	*CHANGE*	0x7F
write	4	*BITFLIP*	LSB 2
write	5	*BITFLIP*	LSB 2
write	7	*CHANGE*	0x7F

Tabela B.4 - Roteiro de Falhas para o Caso de Teste 1.3 gerado automaticamente

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase1.3	OBC	SLP	7 faults

WHERE	**WHEN**	**WHAT**	**HOW**
write	3	*CHANGE*	0x7F
write	4	*CHANGE*	0x7F
write	5	*DELAY*	500 ms
write	6	*BITFLIP*	lsb 2
write	7	*CHANGE*	0x7F
write	8	*BITFLIP*	LSB 2
write	10	*DELAY*	500 ms

Tabela B.5 - Roteiro de Falhas para o Caso de Teste 1.4 gerado automaticamente

ID	**SUT1**	**SUT2**	**NFAULTS**
testcase1.4	OBC	SLP	4 faults

WHERE	**WHEN**	**WHAT**	**HOW**
write	5	*CHANGE*	0x7F
write	7	*BITFLIP*	LSB 2
write	8	*DELAY*	500 ms
write	9	*BITFLIP*	LSB 2

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.