UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA

Aline Cristine Fadel

TÉCNICAS DE TESTES APLICADAS A SOFTWARE EMBARCADO EM REDES ÓPTICAS

Limeira, 2011

Aline Cristine Fadel

Técnicas de testes aplicadas a software embarcado em redes ópticas

Dissertação apresentada ao Curso de Mestrado da Faculdade de Tecnologia da Universidade Estadual de Campinas, como requisito parcial para a obtenção do título de Mestre em Tecnologia.

Área de concentração: Tecnologia e Inovação

Orientador: Prof.ª Dr.ª Regina Lúcia de Oliveira Moraes

Co-orientador: Prof.ª Dr.ª Eliane Martins

Limeira
2011

FICHA CATALOGRÁFICA ELABORADA POR SILVANA MOREIRA DA SILVA SOARES – CRB-8/3965
BIBLIOTECA UNIFICADA FT/CTL
UNICAMP

F12 Fadel, Aline Cristine, 1984-
Técnicas de testes aplicadas a software embarcado de redes ópticas / Aline Cristine Fadel. – Limeira, SP : [s.n.], 2011.

Orientador: Regina Lúcia de Oliveira Moraes.
Coorientador: Eliane Martins.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Tecnologia.

1. Engenharia de software – Injeção de falhas. 2. Software embarcado. I. Moraes, Regina Lúcia de Oliveira. II. Martins, Eliane. III. Universidade Estadual de Campinas. Faculdade de Tecnologia. IV. Título.

Informações para Biblioteca Digital

Título em inglês: Tests techniques applied to embedded software in optical networks

Palavras-chave em inglês (Keywords):

1- Embedded software validation
2- Fault injection
3- Model-based testing
4- GPON
5- CoFI

Área de concentração: Tecnologia e Inovação

Titulação: Mestre em Tecnologia

Banca examinadora: Ana Maria Ambrósio, João Batista Camargo Júnior

Data da Defesa: 19-12-2011

Programa de Pós-Graduação em Tecnologia

DISSERTAÇÃO DE MESTRADO EM TECNOLOGIA

Técnicas de testes aplicadas a software embarcado em redes ópticas

Autor: Aline Cristine Fadel

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof.ª Dr.ª Regina Lúcia de Oliveira Moraes
FT/UNICAMP

Prof.ª Dr.ª Ana Maria Ambrósio
INPE

Prof. Dr. João Batista Camargo Júnior
USP

À Marilza Rodrigues Cruz Fadel e Wilson Antonio Fadel, meus pais,
dedico este trabalho.

Agradecimentos

Primeiramente à Deus por estar sempre presente em minha vida, além de tornar tudo possível.

A realização desse trabalho também não teria sido possível sem a orientação e amizade das Prof.ª Dr.ª Regina Lúcia de Oliveira Moraes e Prof.ª Dr.ª Eliane Martins, que proporcionaram o total apoio durante o projeto de mestrado e na escrita da dissertação.

À UNICAMP e ao Programa de Pós-Graduação da FT que possuem professores com alta qualidade profissional capazes estimular a mente e a criatividade de seus alunos.

Ao CPqD, que além do incentivo ao mestrado, proporcionou meios para que ele fosse realizado.

Aos gerentes do CPqD, Alberto Paradisi e Marcos Sanches, pelo incentivo e cessão do tempo para a execução desse trabalho.

Aos colegas do projeto GPON, que sempre estiveram à disposição para esclarecimentos de qualquer dúvida e dispostos a contribuir para esse trabalho: Antonio Cesar Bizzetti, Renata Bastianon, Fernanda Matsuda, Mariela Mayumi Sasaki, Andre Berti Sassi, e Sandro Rossi.

Ao colega Marcos Perez Mokarzel, que me ofereceu a ideia inicial para esse trabalho.

À minhas amigas Yara Perim e Juliana Maria Silva, que sempre me proporcionaram palavras amigas e de incentivo durante todo o mestrado.

E um agradecimento muito especial à minha família, e ao meu namorado, Renato Resende da Silva, pelo afeto, compreensão e incentivo nas situações mais adversas.

"Não há nada melhor que alcançar uma meta. E não há nada pior que fazer só isso, e não prosseguir."

Amir Klink.

Resumo

Esse trabalho apresenta os detalhes e os resultados de testes automatizados e manuais que utilizaram a técnica de injeção de falhas e que foram aplicados em redes ópticas. No primeiro experimento o teste foi automatizado e utilizou a emulação de falhas físicas baseadas na máquina de estados do *software* embarcado dessa rede. Para esse teste foi utilizado uma chave óptica que é controlada por um robô de testes. O segundo experimento foi um teste manual, que injetou falhas nas mensagens de comunicação do protocolo dessa rede, a fim de validar os mecanismos de tolerância a falhas do *software* central dessa rede. Esse experimento utilizou a metodologia *Conformance and Fault injection* para preparar, executar e relatar os resultados dos casos de testes. Nos dois experimentos também foi utilizado um padrão de documentação de testes que visa facilitar a reprodução dos testes, a fim de que eles possam ser aplicados em outros ambientes. Com a aplicação desses testes, a rede óptica pode alcançar uma maior confiabilidade, disponibilidade e robustez, que são características essenciais para sistemas que requerem alta dependabilidade.

Palavras-chave: validação de *software* embarcado, injeção de falhas, teste baseado em modelos, GPON, CoFI, OMCI

Abstract

This work presents the details and the results of automatic and manual tests that used the fault injection technique and were applied on GPON network. In the first experiment the test was automated, and it performed the emulation of physical faults based on the state machine of the embedded software in this network. In this test is used an optical switch that is controlled by a test robot. The second experiment was a manual test, which injected faults on protocol communication message exchanged through the optical network, in order to validate the main software fault tolerance mechanisms. This experiment used a *Conformance and Fault injection* methodology to prepare, execute and report the results of the test cases. In both experiments, it was used a standard test documentation to facilitate the reproduction of the tests, so that they can be applied in other environments. With applying both tests, the optical networks reach greater reliability, availability and robustness. These attributes are essential for systems that require high dependability.

Key words: embedded software validation, fault injection, model-based testing, GPON, CoFI, OMCI

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ARPANET – *Advanced Research Project Agency Network*

BIP-8 – *Bit-Interleaved Parity* 8

CLI – *Command Line Interface*

CPqD – Centro de Pesquisas e Desenvolvimento em Telecomunicações

CoFI – *Conformance and Fault injection*

CRC – *Cyclic Redundancy Check*

FSM – *Finite state machine*

FTTB – *Fiber to the Building*

FTTC – *Fiber to the Curb*

FTTH – *Fiber to the Home*

FTTN – *Fiber to the Node*

GPON – *Gigabit Passive Optical Network*

HD IPTV – *High Definition Internet Protocol Television*

ITU-T – *International Telecommunication Union - Telecommunication Standardization Sector*

MIB – *Management Information Base*

ODN – *Optical Distribution Networks*

OLT – *Optical Line Terminal*

OMCI – *ONT Management and Control Interface*

ONU / ONT – *Optical Termination Unit / Optical Network Terminal*

OXC – *Optical Cross-Connect*

OPEX – *Operational Expenditures*

PON - *Passive optical Network*

POTS – *Plain Old Telephone Service*

PMD – *Physical Media Dependent*

QoSFT – *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*

SUT – *System under Test*

SWIFI – *Software-Implemented Fault Injection*

TC – *Transmission convergence*

TCP/IP – *Transmission Control Protocol – Internet Protocol*

U2TP – *UML Testing Profile*

VoIP – *Voice over Internet Protocol*

SUMÁRIO

1 Introdução

Há pelo menos dois séculos, alguns grandes inventores tentavam criar formas para diminuir a distância entre as pessoas. Iniciou-se com o telegrafo, em 1844 (Samuel Morse), e posteriormente com o telefone, em 1876 (Alexander Graham Bell).

Impulsionada por questões militares, em 1968, foi criada a ARPANET (*Advanced Research Project Agency Network*), que se tornou a primeira rede de microcomputadores. Após isso, muitas redes foram criadas, porém elas não se comunicavam da mesma forma. Para garantir a comunição entre elas, foi criado o protocolo aberto TCP/IP, *Transmission Control Protocol – Internet Protocol* (Vinton Gray Cerf, 1983). Esse protocolo possibilitou que os pacotes de mensagens pudessem percorrer as redes de Internet integrando todos os serviços já disponíveis nessa época (*email*, WWW, *chat*). A partir desse momento, a Internet e os computadores se popularizaram e ganharam espaço mundial.

As redes de telecomunicações ganharam novas utilizações, e passaram a ser também empregadas como meio de entretenimento, repositórios de informações e redes corporativas. E com isso, os seus usuários passaram também a demandar mais banda, maior confiabilidade, além da busca por soluções a um custo mais baixo.

Diante dessas novas necessidades, as redes de telecomunicações precisaram ser aprimoradas. Os antigos pares de fio de cobre ficaram ultrapassados, e surgiram novos meios físicos para as telecomunicações, como as fibras ópticas.

As redes de fibras ópticas são capazes de suportar uma grande carga de usuários, altas taxas de transferência e oferecer serviços que exigem alta qualidade mesmo em situações adversas.

1.1 Contexto

Um fator fundamental de qualidade em redes de telecomunicações é o grau de disponibilidade de serviço, que raramente é de 100%. A quebra de fibras ópticas pode ocorrer entre 4 e 7 vezes ao ano, para cada 1600 km de extensão e o tempo médio de recuperação destas falhas é de 12 horas. As redes ópticas chegam a atingir a disponibilidade de 99,999%, que corresponde a um período de indisponibilidade de até 5 minutos ao ano. Para atingir esse alto grau de disponibilidade devem ser implantados mecanismos de sobrevivência a falhas (BICUDO, 2005). O alto grau de disponibilidade exigido para as redes ópticas colabora para a confiabilidade desse tipo de infraestrutura de comunicação.

Devido ao alto grau de confiabilidade, as redes ópticas estão ganhando espaço no mercado de telecomunicações, principalmente nos continentes europeus e asiáticos (REGGIANI, 2010). Além da confiabilidade, as redes ópticas possuem como características principais:

- Perda muito baixa de transmissão;
- Não sofrem interferências eletromagnéticas;
- Transmissões seguras e privadas;
- Decréscimo no custo de implantação e manutenção;
- Altas taxas de transmissão.

Os defeitos que podem surgir em sistemas de telecomunicações, assim como em sistemas de outros domínios, em alguns casos são inevitáveis. Nas redes de telecomunicações podem ocorrer falhas como a ausência de comunicação e a perda de dados. Estas geram grandes prejuízos às operadoras de telecomunicações e a seus usuários, devendo ser feito um grande esforço para que seus efeitos sejam minimizados.

Os defeitos de *software* podem ser evitados, ou minimizados, através da aplicação de técnicas de testes, que exercitam o *software* com a intenção de encontrar erros. Além de que, quanto antes os defeitos forem encontrados, menores serão os custos com a correção. Porém

vale destacar, que não existem garantias de que o *software* funcionará sem a presença de erros (MYERS, 2004).

Nesse trabalho serão destacadas técnicas de testes aplicadas ao *software* das redes ópticas passivas com capacidade para *gigabit*, denominadas redes GPON (*Gigabit Passive Optical Network),* que estão sendo desenvolvidas no CPqD (Centro de Pesquisas e Desenvolvimentos em Telecomunicações). As redes GPON são soluções de acesso de alta capacidade para serviços *triple-play* (voz, vídeo e dados) utilizando a transmissão de dados via fibra óptica (REGGIANI, 2010).

Esse trabalho expõe dois experimentos de testes de *software* que foram realizados na rede GPON, visando tornar essas redes mais confiáveis e disponíveis.

Para a complementação dos testes foram pesquisados métodos de injeção de falhas e testes baseados em modelos de estados em *software* embarcados, a fim de verificar se o sistema apresentará um comportamento adequado mesmo na presença de falhas. Essas falhas foram fundamentadas nas falhas de comunicação e na perda e alteração dos pacotes de mensagens de comunicação dos componentes da rede.

Com a finalidade de que os experimentos possam ser facilmente compreendidos e reproduzidos, eles estão documentados através de um padrão para testes de robustez.

1.2 Objetivo

O objetivo desse trabalho é experimentar técnicas de testes de software, de modo a garantir um bom nível de qualidade nos serviços prestados pelas redes GPON, evitando a interrupção de comunicação e perda de dados.

Não é o objetivo desse trabalho comparar as técnicas de injeção de falhas e testes baseados em modelos de estados, com as técnicas anteriormente aplicadas ao GPON (técnicas de testes caixa branca ou testes de regressão, por exemplo) e sim complementá-las.

1.3 Contribuições

Os resultados desse trabalho trazem contribuições para todos os grupos de desenvolvimento de *software* embarcado que buscam que seus sistemas atinjam uma maior confiabilidade e excelência nas prestações de seus serviços.

Esse estudo foi realizado em sistemas de telecomunicações e foi capaz de demonstrar ao grupo de desenvolvimento de *software* do GPON no CPqD, que as técnicas de testes aplicadas se mostraram eficientes para a validação de sistemas em situações adversas, e se espera que possam ser reproduzidas em sistemas de outros domínios.

As contribuições atingidas com os resultados desse trabalho são:

- Criação de uma abordagem para emular e automatizar a execução de testes em presenças de falhas físicas;
- Utilização de uma metodologia para testar a robustez do *software* da OLT (*Optical Line Termination)*, quando alguns pacotes do protocolo OMCI (*ONT Management and Control Interface*) são corrompidos;
- Emprego de uma abordagem para documentar os experimentos de testes realizados, de forma que possam ser reproduzidos futuramente;
- Criação de um robô de testes (ferramenta pertencente ao CPqD) capaz de executar casos de testes para testes de regressão e também testes para aplicação de injeção de falhas.

1.4 Terminologia

Para um melhor entendimento desse trabalho alguns conceitos relacionados aos testes de *software* serão definidos a seguir (LAPRIE LEITE & LOQUES, 1987 *apud* LAPRIE, 1995):

- Falha (*fault*): é um componente, de *software* ou de *hardware*, que apresenta comportamento irregular dentro do sistema, que é a causa suposta ou constatada de um erro;
- Erro (*error*): após a ocorrência e a identificação de uma falha em um sistema, essa falha poderá levar o sistema a um estado incorreto. Esse estado causado pela falha é denominado erro.
- Defeito (*failure*): após uma falha levar a ocorrência de um erro, esse erro pode ocasionar um defeito no sistema. Um defeito ocasiona uma prestação de serviços fora do especificado.

Nas redes GPON, alguns termos serão bastante utilizados (ITU-T G.984.1, 2008):

- ONU *(Optical Network Unit)* e ONT (*Optical Network Terminal)*: são os dispositivos localizados nas terminações dos assinantes, que podem estar presentes em vários locais na rede. Como a utilização desses dispositivos é semelhante, nesse trabalho só será utilizado o termo ONU;
- OLT: é um dispositivo central das redes GPON, é responsável por gerenciar e realizar a manutenção da rede e das ONUs;
- OMCI: protocolo de comunicação utilizado nas redes GPON;
- OXC (*Optical Cross-Connect):* chave óptica, utilizada no segundo experimento.

1.5 Estrutura do Trabalho

Esse trabalho está dividido da seguinte forma, a Seção 2 abordará as redes GPON e suas características, além de informações sobre o protocolo OMCI. A Seção 3 apresentará os trabalhos relacionados. A Seção 4 abrangerá as técnicas de testes utilizadas nos dois experimentos desse trabalho. A Seção 5 abordará a metodologia CoFI (*Conformance and Fault injection)* utilizada no segundo experimento. Na Seção 6 serão apresentados os estudos

de caso aplicados na GPON, divididos entre o experimento 1 e 2. E, finalmente, na Seção 7, serão relatadas as conclusões deste trabalho além das perspectivas para os trabalhos futuros.

2 *Gigabit Passive Optical Network* (GPON)

Atualmente existem inúmeras opções de acesso à rede mundial de computadores, além do oferecimento de uma diversidade de serviços, tais como: TV, voz e dados. Todas essas informações necessitam trafegar em velocidade cada vez mais rápida, com qualidade e segurança.

Dá-se o nome de redes de acesso, ou última milha, ao trecho de oferecimento de cobertura do serviço desde a central telefônica até o ponto de acesso, que podem ser residências, ou corporações, por exemplo. As redes GPON são classificadas como redes de acesso, pois chegam até o usuário ou muito próximo a ele.

Nas redes de telecomunicações, o cabo mais utilizado atualmente é o par trançado feito de cobre. Contudo, com o aumento de pesquisas em fibras ópticas, descobriu-se que esta apresenta diversas vantagens em relação ao cobre. As fibras possuem como características básicas a imunidade à interferência eletromagnética, menor perda de energia por quilômetro e maior velocidade de transmissão (tanto no sentido *downstream* quanto no *upstream*). Além das características citadas, o fato da fibra ser constituída de sílica traz uma redução do valor comercial das redes que utilizam a fibra como meio físico (REGGIANI, 2010).

A rede externa óptica é totalmente passiva, ou seja, não necessita de alimentação elétrica, o que facilita a manutenção e aumenta a durabilidade (redução das despesas operacionais *Operational Expenditures* – OPEX).

Porém, apesar de todas as vantagens até aqui apresentadas, até recentemente os equipamentos necessários para a implantação de redes ópticas apresentavam custos elevados e a demanda por banda não era suficiente para justificar o investimento. As redes ópticas passivas surgiram como soluções às necessidades dos novos serviços que as operadoras desejam prover aos assinantes agregando, às vantagens acima, o baixo custo com equipamentos e compartilhamento da infraestrutura.

Em diversas partes do mundo, tais como o Japão, Alemanha, Inglaterra e Canadá, as redes PON (*Passive Optical Network)* já estão difundidas devido, principalmente, à utilização de redes de baixo custo entre as suas unidades de negócios (SANCHEZ, 2004).

As redes GPON estão em desenvolvimento no CPqD há pelo menos 5 anos. Além de todos os componentes de *software* embarcados (OLT, ONU e OMCI) desenvolvidos, o *hardware* também é construído e testado na instituição.

2.1 Estrutura e Funcionamento

O sistema GPON é composto por um Terminal de Linha Óptica (OLT), instalado na central da operadora, que é responsável por prover todos os serviços da rede. A rede também contém diversas Unidades de Rede Óptica (ONU), instalados nos clientes, que podem ser residenciais e corporativos. As ONUs podem conter portas ethernet (serviço de dados), canais E1 (telefonia) e portas POTS (*Plain Old Telephone Service* - serviço de voz), entre outros serviços. Ao invés de utilizar sistemas eletrônicos na Rede de Distribuição Óptica (ODN, *Optical Distribution Networks*), o uso de divisores passivos (*splitters*) permite dividir a largura de banda disponível para atender a vários usuários. Os componentes passivos utilizados na rede não necessitam de fontes de alimentação para funcionar. Todos esses elementos podem ser observados na Figura 1.

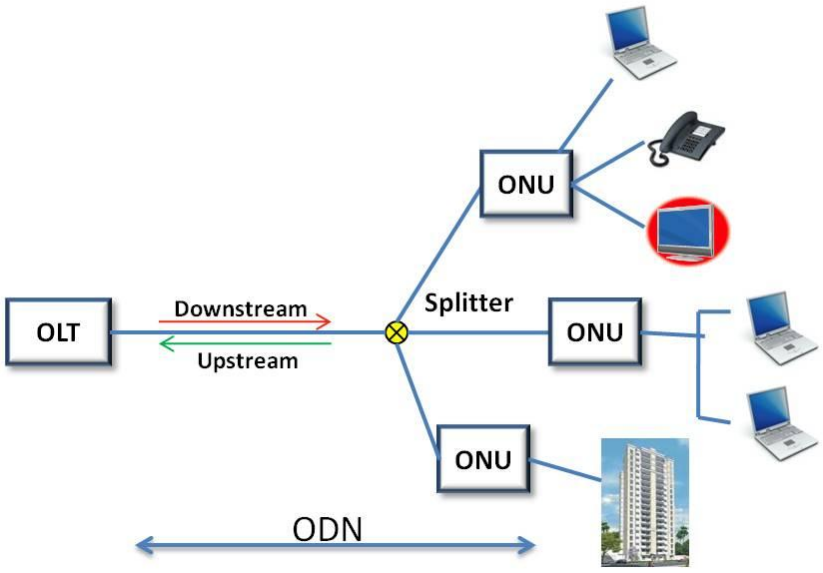

Figura 1: Rede GPON e seus elementos

As informações que trafegam da OLT para as ONUs, são conhecidas por percorrer o sentido *downstream*, e ao contrário, no sentido *upstream* (ITU-T G.984.4). Uma OLT pode ser compartilhada com até 64 ONUs, porém a capacidade esperada, é que esse número chegue a 128 ONUs. A possibilidade de distância para essas redes chega a até 60 km (ITU-T G.984.1).

A taxa de transmissão no sentido *downstream* chega a 2,5 Gbps e no sentido *upstream*, a taxa é de 1,25 Gbps (ITU-T G.984.1). A diferença de velocidade entre os sentidos, ocorre pois no sentido *downstream* a transmissão é contínua, atingindo maiores taxas. Já no sentido *upstream*, a transmissão deve ser dividida entre todas as ONUs conectadas. Para haver o sincronismo entre todas essas ONUs a velocidade é reduzida, permitindo que todas as ONUs transmitam a sua rajada de dados em *slots* de tempos separados.

As redes GPON são normatizadas por recomendações elaboradas pela ITU-T (*International Telecommunication Union - Telecommunication Standardization Sector*). As principais normas que regem o GPON estão listadas e resumidas na Tabela 1.

Tabela 1: Normas ITU-T para as redes GPON (baseado em BONILLA, 2008)

ITU-T	Recomendação	Resumo
G.984.1	*Gigabit-capable passive optical networks* (GPON): *General characteristics*	• Fornece resumo das características essenciais; • Taxas de transmissão desde 1.244/0.155 até 2.488/2.488 Gbps; • 20 km: Máximo alcance físico; • 60 km: Máximo alcance lógico.
G.984.2	*Gigabit-capable Passive Optical Networks* (GPON): *Physical Media Dependent* (PMD) *layer specification*	• Fornece as especificações da camada PMD (*Physical Media Dependent*), que é responsável pela comunicação com o ambiente físico.
G.984.3	*Gigabit-capable Passive Optical Networks* (GPON*): Transmission convergence* (TC) *layer specification.*	• Fornece as especificações da camada TC, responsável pela extração de informações desde a camada física.
G.984.4	*Gigabit-capable Passive Optical Networks* (GPON): *ONT management and control interface specification* (OMCI)	• Fornece gerenciamento da ONU; • Definição do protocolo OMCI;
G.984.5	*Gigabit-capable Passive Optical Networks* (GPON): *Enhancement band*	• Melhoramento da banda.

A segurança das redes GPON é garantida pela criptografia dos dados enviados, que utilizam o número de série de cada ONU, e só podem ser decifrados pela respectiva ONU. Em uma rede GPON, também não são aceitas mais de uma ONU com o mesmo número de série.

Diversas arquiteturas podem ser utilizadas em função da distância da ONU até o usuário final: FTTB (*Fiber to the Building*, fibra até o prédio) ou FTTH (*Fiber to the Home,* para residências), para as distâncias mais curtas; as FTTN (*Fiber to the Node*, fibra até o nó de rede), para as distâncias mais longas; e o FTTC (*Fiber to the Curb*, fibras até um ponto de passeio, calçadas, por exemplo) para distâncias intermediárias, utilizadas para a instalação e posicionamento das ONUs.

As redes GPON podem oferecer a confiabilidade e o desempenho necessário para atender os serviços das empresas, mas deve ser cuidadosamente projetadas desde o início. Smith (2006) propõe algumas formas de redundância física, proporcionando proteção tanto para os clientes residenciais como para os corporativos.

A transmissão de informações pela fibra é modular, ou seja, múltiplos serviços podem utilizar a mesma fibra, pois cada um dos serviços pode utilizar um comprimento de onda diferente. Devido a isso, as redes GPON são capazes de oferecer serviços como, por exemplo, VoIP (*Voice over Internet Protocol*), HD IPTV *(High Definition Internet Protocol Television)*, *download* de vídeos, vídeo conferências, entre outros serviços.

No processo evolutivo das redes GPON é previsto um aumento de banda, de 2,5 Gbit/s para 10 Gbit/s, no sentido *downstream*, e de 1,25 Gbit/s para 2,5 Gbit/s, no sentido *upstream*. O que aperfeiçoará ainda mais as necessidades para os seus usuários corporativos e residenciais. Porém é necessário um planejamento adequado para que a migração para os sistemas mais modernos permita a existência de outros GPONs, pertencentes a outras gerações, na mesma rede óptica (TELECO, 2008).

2.2 Protocolo *ONT Management and Control Interface* (OMCI)

O protocolo de comunicação utilizados nas redes GPON é conhecido como OMCI (*ONT management and control interface*), que é normatizado pela norma ITU-T G.984.4. A norma tem como objetivo padronizar o protocolo, a fim de que possa ocorrer a interoperabilidade entre diversos fabricantes de ONUs e OLTs. A ITU-T G.984.4 estabelece a implementação de entidades que nada mais são do que representações dos recursos e serviços oferecidos pela ONU.

O OMCI tem a responsabilidade de controlar a comunicação entre a OLT e as ONUs e vice-versa, gerenciando a configuração das ONUs, as falhas, o desempenho, e a segurança do sistema GPON. Nas redes GPON, a OLT comporta-se como mestre e as ONUs como escravos. Os defeitos que podem ocorrer no sistema GPON são representadas por alarmes. Esses alarmes são tratados pela ONU, para serem enviados por mensagens via OMCI para a OLT.

O protocolo OMCI é composto por mensagens como: *create*, *delete*, *set*, *get*, MIB (*Management Information Base) upload*, *next* MIB *upload*, MIB *reset*, *start software download*, *download section*, *end software download*, *activate image* e *commit image*, que são enviadas pela OLT. Todos esses tipos de mensagens possuem a sua correspondente *response*, que são enviadas pelas ONUs.

As normas da ITU-T recomendam algumas formas para detecção de erros nas mensagens do OMCI, que são o BIP-8 (*bit-interleaved parity 8*) e o CRC (*Cyclic redundance check*). Mais especificamente, o CRC calcula a integridade dos dados dos pacotes enviados. Quando a OLT recebe uma mensagem, ela calcula o CRC da mensagem e compara com o CRC gravado na mensagem. Caso esses valores não sejam idênticos, a mensagem deve ser descartada.

Na Figura 2, são detalhados alguns campos do pacote da mensagem OMCI. Os principais são: o Tipo da mensagem (correspondente ao tipo da mensagem), a prioridade da mensagem, o *Class ID* (correspondente à entidade) e *Entity ID* (correspondente à identificação da entidade), e CRC da mensagem (localizado no OMCI *trailer*).

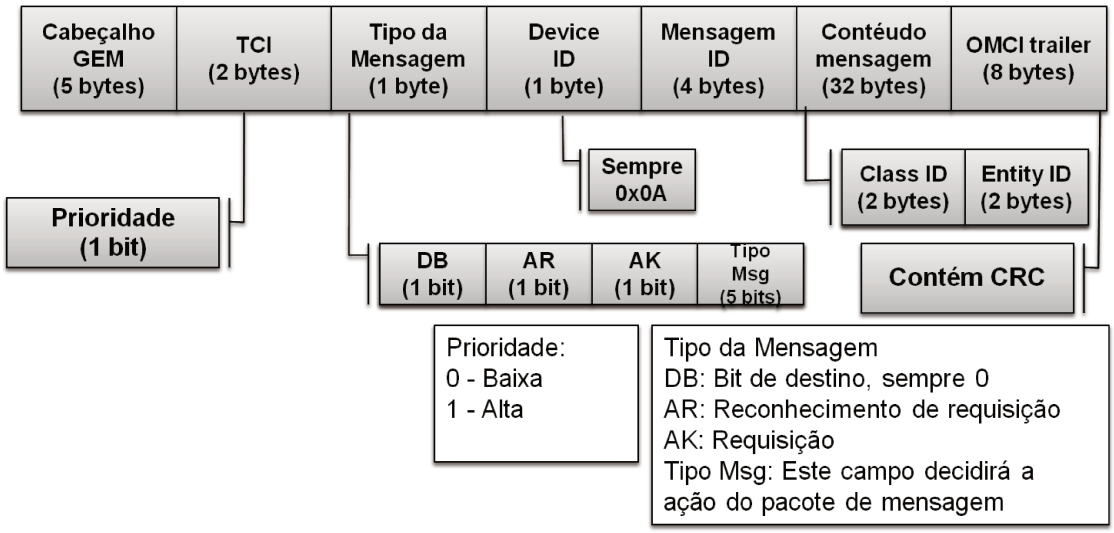

Figura 2: Pacote OMCI (adaptado de ITU-T G.984.4)

O protocolo OMCI é controlado por um mecanismo de fluxo de mensagens denominado *stop-and-wait* (ITU-T G.984.4). Esse processo garante que as mensagens transmitidas para a OLT e ONU tenham sido corretamente recebidas e processadas respeitando a prioridade de cada pacote. O mecanismo de *stop-and-wait* é composto por um campo de identificação da mensagem, um contador de tentativas de reenvio e um contador de tempo limite de espera.

2.3 Considerações Finais

Nesta Seção foi apresentada a estrutura básica e o funcionamento das redes GPON, as normas que regem o GPON e o seu protocolo, o OMCI.

As redes GPON são classificadas como de nova geração, não só por possibilitarem maior banda, mas sim pela infraestrutura que elas oferecem. Os múltiplos serviços oferecidos (voz, vídeo e dados) são compartilhados em uma única rede. São capazes de atender desde condomínios residenciais até grandes corporações, com garantia de qualidade. É uma solução que emerge devido ao baixo custo comparado com as altas taxas de banda.

3 Trabalhos Relacionados

Nessa Seção são apresentados os trabalhos publicados que fundamentam essa pesquisa. Nas próximas subseções serão abordados os temas de: testes de *software*, qualidade de *software*, injeção de falhas, autômatos finitos e automação dos testes de *software*.

3.1 Teste de Software e Sistemas Embarcados

Um dos principais objetivos de se realizar testes de *software* é fornecer um retorno sobre a confiabilidade e cobertura das funcionalidades que o cliente de um sistema necessita. A técnica de teste de *software* é, talvez, o método que mais demonstra se o código satisfaz a codificação lógica especificada (VOAS & MCGRAW, 1998).

A meta no processo de execução dos testes é minimizar possíveis riscos causados pelas falhas inseridas durante o processo de desenvolvimento do *software*. Isso não quer dizer que o *software* estará isento de falhas (DIJKSTRA, 1979), mas sim que muitas técnicas foram executadas para diminuir a quantidade de defeitos no produto final. Porém, testar exaustivamente sistemas computacionais não é a melhor solução e nem sempre isso é possível (VOAS & MCGRAW, 1998). O objetivo dos testes é encontrar falhas e não solucionar os problemas causados ao sistema em decorrência da presença dessas falhas.

Para os testes realizados nos sistemas embarcados as preocupações com as falhas não devem ser muito diferentes. O objetivo da atividade de teste é o mesmo, porém as consequências são muito mais sérias, por se tratarem de sistemas utilizados em automóveis, aviões, sistemas de telecomunicações, equipamentos eletrônicos, equipamentos médicos e maquinários industriais.

Os sistemas embarcados são desenvolvidos para fins específicos, e sua capacidade computacional é colocada dentro de um circuito integrado, equipamento ou sistema, através de

um *software* que controlará o *hardware*. O operador desse sistema só terá acesso a esse equipamento através de interfaces (teclado, botões e *displays*), caso o sistema disponibilize. A ideia fundamental da aplicação do *software* embarcado é que o usuário não perceba a existência do *software*, apenas utilize um conjunto de funcionalidades e personalizações relativas àquele produto (TAURION, 2005). A principal função dos sistemas embarcados é interagir com o ambiente físico (LEE, 2001).

Alguns tipos de sistemas embarcados podem ser considerados sistemas críticos, como os sistemas controladores de aeronaves e automóveis, sistemas de auxílio médico, entre outros. Um sistema crítico é aquele que, em caso de uma falha, possa resultar em perda de vidas, danos materiais ou danos significativos ao meio ambiente (KNIGHT, 2002).

Os segmentos que mais demandam uso de produtos de *software* embarcado são as telecomunicações (35%), a eletrônica de consumo (20%), a automação industrial (19%) e a automotiva (10%). Outras demandas menores, porém não menos importantes, também utilizam *software* embarcado, tais como sistemas médicos e aeroespaciais (TAURION, 2005). O autor também explica que o mercado mundial de *software* embarcado já é grande (faturamento estimado em mais de 21 bilhões de dólares) e cresce rapidamente. Há pelo menos dez vezes mais dispositivos que embutem *software*, como aparelhos celulares, *set-top-boxes* para TVs e dispositivos eletrônicos em automóveis do que em computadores pessoais, e esta diferença aumenta a cada dia.

Normalmente, o *software* para sistemas embarcados é desenvolvido para sistemas com grandes limitações de recursos, como memória e processamento, por exemplo. Algumas das dificuldades enfrentadas no processo de desenvolvimento de produtos de *software* embarcados são (LEE, 2001):

- Sistemas embarcados interagem normalmente com vários processos físicos, além de reagirem simultaneamente a um grande número de sensores e, ao mesmo tempo, serem controlados externamente;
- O *software* embarcado interage com eventos periódicos (amostragem do sinal de sensores, controle de atuadores), mas também com eventos não periódicos, muitas vezes imprevisíveis (alarmes).

Um dos experimentos proposto nesse trabalho tratará da automação da execução dos testes, que será abordada na próxima subseção.

3.2 Automação de testes de software

Uma maneira de aumentar a eficiência na execução dos testes é automatizá-los. A técnica de automação de testes visa à utilização de ferramentas de execução de casos de testes, que executem os casos de testes selecionados de forma automática, garantindo resultados mais rápidos e confiáveis em relação aos testes realizados manualmente. O processo de automação também pode ser definido como sendo a utilização de qualquer ferramenta de *software* que automatiza parte ou a totalidade do processo de verificação (IEEE 610.12-1990).

A automação garante resultados mais precisos em um tempo de execução bem menor em relação ao manual. Porém, em se tratando de *software* embarcado a *interface* pode ser um fator complicador. Cada sistema embarcado pode possuir uma *interface* personalizada e dependente do *hardware* (HANDLER, 2010), necessitando da construção de ferramentas específicas para a automação de cada sistema, para que estas ferramentas possam atender a um número maior de funcionalidades.

Porém, o processo de automação de testes de *software* nem sempre é desenvolvido com estratégias sólidas. Devido a isso, existem muitos insucessos no processo de automação de testes, como os casos descritos em DUSTIN (1999). Nesse trabalho são descritas várias lições aprendidas durante o processo de automação de testes. Destacam – se o fato de que a equipe de testes nunca deve perder o foco durante a automação, isto é, o teste do sistema. Além de que nem tudo pode e deve ser automatizado, sempre necessitando de uma análise prévia do escopo de teste. Ao adquirir uma ferramenta para a automação, não se deve criar grandes expectativas em torno dela, pois em um primeiro momento ela só vai aumentar o escopo dos testes, posteriormente ela deverá trazer outros retornos. Além disso, existem testes manuais que nunca poderão ser automatizados.

De acordo com as informações listadas em (FANTINATO *et al*, 2004, *apud* FEWSTER & GRAHAM, 1999), existem quatro tipos principais de técnicas de automação de testes de *software*: *record & playback*, programação de *scripts*, *data-driven* e *keyword-driven*. A técnica de *record & playback* consiste em capturar as ações executadas pelo usuário em uma interface gráfica e transformá-la em *scripts* de testes, que podem ser reexecutados a qualquer momento. Porém, as suas desvantagens são: “alto custo e dificuldade de manutenção,

baixa taxa de reutilização, curto tempo de vida e alta sensibilidade a mudanças no *software* a ser testado e no ambiente de teste" (FANTINATO *et al*, 2004).

A técnica de programação de *scripts* é baseada na técnica *record & playback*, porém a sua diferença é que os *scripts* podem ser alterados em cada execução (FANTINATO *et al*, 2004), o que aumenta a sua eficiência.

A técnica de *data-driven* extrai os *scripts* de testes a partir de uma base de dados. Isso facilita a atualização dos testes já que a base de dados pode ser modificada em qualquer momento, sem acarretar consequências aos *scripts* (FANTINATO *et al*, 2004).

A técnica *keyword-driven*, consiste em extrair, dos *scripts* de teste, o procedimento de teste que representa a lógica de execução. Essa técnica também possui a vantagem do caso de teste ser alterado em qualquer momento (FANTINATO *et al*, 2004).

Quando o processo de automação é executado corretamente, o processo torna-se uma das melhores formas de reduzir o tempo de teste no ciclo de vida do *software*, diminuindo o custo e aumentando a produtividade do desenvolvimento de *software* como um todo, além de, consequentemente, aumentar a qualidade do produto final (FANTINATO *et al*, 2004).

Vale acrescentar que o projeto deve estar em um grau de maturidade suficiente para que os *scripts* não sejam perdidos ou necessitem ser refeitos para as novas versões dos sistemas. Porém mesmo tomando essas precauções, durante o desenvolvimento de um sistema, há grandes chances dele sofrer alterações de melhorias ou de correções de defeitos que podem impactar os casos de testes já elaborados. Em KANER *et al* (2002), há a afirmação de que as mudanças nos sistemas são quase constantes, e as ferramentas de testes devem se adaptar à essas alterações.

3.3 Qualidade de software e Dependabilidade

O *software* tem, comumente, sua qualidade medida somente ao final de seu desenvolvimento (VOAS & MCGRAW, 1998).

Dependabilidade é um conjunto de características essenciais para definir a qualidade de um *software*. A dependabilidade pode ser definida como a habilidade de entregar um serviço legitimamente confiável, ou seja, é a habilidade de um sistema evitar falhas de serviço mais graves e mais frequentes que o aceitável (AVIZIENIS *et al*, 2004). Para WEBER (2010), a dependabilidade é uma junção da qualidade do serviço fornecido por um sistema com a confiança depositada no serviço fornecido.

A dependabilidade é qualificada como um atributo do sistema, quantificado através de medidas específicas. Três dos principais atributos da dependabilidade são confiabilidade, disponibilidade e robustez. A confiabilidade é a capacidade de sobreviver (sem falhas) durante um intervalo de tempo; disponibilidade é a capacidade de estar operacional (sem falhas) em um dado instante de tempo (CLARK & PRADHAN, 1995); robustez é caracterizada pela resistência do sistema quando submetido a falhas (AVIZIENIS *et al*, 2004).

Os testes de robustez envolvem testes em um sistema na presença de falhas ou de condições ambientais estressantes.

A evolução da dependabilidade sobre o ciclo de vida dos sistemas pode ser caracterizada por noções de estabilidade, aumento e redução da intensidade dos efeitos, que ocorre através da percepção do usuário com a utilização do sistema. Inicialmente, um sistema possui uma redução dos defeitos (confiabilidade aumenta), depois estabiliza (estabilidade da confiabilidade) e depois de certo período de operação, o número de defeitos aumenta (confiabilidade diminui) e o ciclo recomeça (AVIZIENIS *et al,* 2001).

Algumas técnicas que podem ser utilizadas para atingir a dependabilidade são (AVIZIENIS *et al*, 2004):

- Prevenção de falhas: maneira para prevenir a ocorrência ou introdução de falhas;
- Tolerância a falhas: métodos para garantir a entrega de serviços corretos mesmo em presença de falhas, ou seja, entregar o serviço com confiança;
- Remoção de falhas: métodos para reduzir o número e a gravidade das falhas;
- Previsão de falhas: meios para estimar o número atual, a incidência futura, e as prováveis consequências de falhas.

Na próxima subseção serão tratadas as injeções de falhas, técnica que é utilizada para verificar se um sistema é tolerante a falhas.

3.4 Injeção de falhas

O crescimento da dependência dos computadores em muitas áreas de aplicações torna os custos associados com as falhas dos sistemas cada vez mais altos (KARLSSON *et al*, 1991). Para minimizar o custo gasto com a eliminação e consequencia das falhas, é necessário fornecer sistemas de computadores com mecanismos efetivos de manipulação de falhas. Em algumas aplicações é necessário utilizar sistemas tolerantes a falhas, para que o sistema possa oferecer seu serviço mesmo na presença de falhas.

Um sistema ou um componente é tolerante a falhas quando ele é capaz de continuar operando, apesar da presença de falhas no *hardware* ou *software* (IEEE 610.12-1990).

As técnicas de tolerância a falhas visam evitar o fracasso de um sistema e são realizados através de detecção de erros e recuperação do sistema. A detecção de erros objetiva identificar a presença de erros e a recuperação do sistema transforma o estado do sistema que contem um ou mais erros e possíveis falhas, em um estado consistente (AVIZIENIS *et al*, 2004).

O uso de mecanismos de tolerância a falhas torna-se importante também devido ao crescimento baseado em serviços contínuos (sem parada) (DE MILLO *et al*, 1994), como no caso dos sistemas embarcados, onde muitos desses sistemas só são interrompidos quando é necessário realizar algum tipo de manutenção.

Ao desenvolver sistemas que requerem alta confiabilidade, apenas a implementação de mecanismos de tolerância a falhas não é suficiente [MARTINS, 1995]. É muito importante que seja realizada a validação desses sistemas, a fim de garantir que elas sejam corretamente implementadas, ou seja, que todos os serviços oferecidos pelo sistema sejam fornecidos de acordo com suas especificações.

Uma forma de se medir se as técnicas de tolerância a falhas são eficientes é através da cobertura de testes do código (DE MILLO *et al*, 1994). Uma maior cobertura exercitada do código trará uma maior qualidade ao *software*.

As técnicas de injeção de falhas podem ser aplicadas para aprimorar os mecanismos de tolerância a falhas dos sistemas. A injeção de falhas em *software* foca em determinar o quanto bem ou mal o *software* irá se comportar sob uma variedade de circunstâncias (VOAS & MCGRAW, 1998).

Quando um experimento de injeção de falhas é realizado, dois pontos devem ser analisados: a carga de trabalho (*workload*) e o modelo de falhas (*faultload*). O *workload* é basicamente o(s) programa(s) e a(s) tarefa(s) que está(ão) rodando no sistema quando o experimento é realizado. O *workload* pode estender os esforços habituais de testes, submetendo o sistema para testes de carga superior (ReSIST, 2006).

Faultload pode ser definido como um conjunto de falhas ou condições de estresse que podem afetar o funcionamento de um sistema. O *faultload* é utilizado para se observar o comportamento do sistema quando submetido a um determinado conjunto de falhas (ReSIST, 2006). Elas podem ser divididas em três principais tipos: falhas de *software*, falhas de *hardware* e falhas causadas pelo operador (VIERA & MADEIRA, 2002).

O princípio da injeção de falhas pode ser exemplificado na Figura 3. Como entradas o *workload* e *faultload*, que atuam sobre o sistema, gerando saídas esperadas e defeitos.

Figura 3: Princípio da injeção de falhas

A injeção de falhas testa a detecção de falhas, isolamento de falhas, a reconfiguração e a capacidade de recuperação do sistema em uso (HSUEH *et al*, 1997). Uma vez ativada a falha, a propagação de erros ocorre quando um estado transitório interfere no estado sucessor (VOAS & MCGRAW, 1998).

Podem existir basicamente duas intenções em se realizar testes de injeção de falhas que são: o de encontrar um grande número de falhas no sistema, ou o de observar o sistema em um ambiente real com falhas (CRNKOVIC & LARSSON, 2002).

Nesse trabalho a técnica de injeção de falhas será explorada em conjunto com o uso de máquinas de estado finitas. As máquinas de estados podem ser definidas como um conjunto de estados únicos que representam circuitos sequenciais. Um estado especial é o inicial, um ou mais estados podem ser os finais. Quando algum evento acontece, ocorre a transição para um novo estado (THOMAS & HUNT, 2002).

As máquinas de estados são utilizadas em programas que possuem entradas e dependendo do comportamento, a máquina pode passar por múltiplos estados (THOMAS &

HUNT, 2002). O comportamento de um sistema com máquinas de estados depende dos valores enviados em suas entradas e do conteúdo do seu estado.

O teste que utiliza máquina de estados e outros modelos é conhecido como teste de modelos de estado. O ponto chave desse teste é automatizar o desenvolvimento de testes caixa-preta (UTTING & LEGEARD, 2007), ou seja, os testes que analisam o comportamento do sistema.

A combinação das técnicas de injeção de falhas e modelos de estados mostrou-se eficiente em aplicações espaciais. Este experimento utilizou uma ferramenta que gerou casos de teste de forma automática baseados na máquina de estados (AMBROSIO *et al*, 2007).

As máquinas de estados finitas também são chamadas de autômatos finitos que será o foco da próxima subseção.

3.5 Autômatos Finitos

As máquinas de estados finitos são definidas como um modelo matemático de sistemas com entradas e saídas discretas e que especificam comportamentos através de estados, ações e transições. Por definição, esses sistemas contêm um número finito e pré-definido de estados. Cada estado sintetiza somente as informações do passado recente necessárias para determinar as ações a serem tomadas para a próxima entrada (MENEZES, 2002). Os principais termos utilizados na teoria dos autômatos são os seguintes:

- Alfabeto ($\sum$): conjunto finito e não vazio de símbolos que podem ser interpretados pelo autômato. Um exemplo é o alfabeto binário, $\sum = \{0,1\}$;
- *String* (também pode ser chamada de palavra ou cadeia): é uma sequência finita de símbolos escolhidos de um alfabeto. Uma das palavras de um alfabeto binário poderia ser *string*=111;
- Linguagem: conjunto de *strings* compostas por símbolos do alfabeto e geradas pelo autômato finito.

- Estado (Q): Situação atual do autômato. Nos autômatos, existe um conjunto finito de estados (S), um estado inicial (q_0) e um conjunto finito de estados finais (F).
- Função transição (δ): função que determina a transição entre estados, baseado no estado atual e no símbolo de entrada. Matematicamente, pode ser definida como: $\delta: \Sigma \times Q \rightarrow Q$.

Formalmente, os autômatos podem ser descritos da seguinte forma:

M = {Q, ∑, δ, q_0, F}

Q: conjunto finito de estados (q_0, q_1, q_2, ..., q_n)

∑: conjunto finito de símbolos de entrada;

δ: função transição que toma como argumento um estado e um símbolo de entrada e retorna um estado ($\delta: \Sigma \times Q \rightarrow Q$);

q_0: estado inicial;

F: conjunto de estados finais ($F \subseteq Q$).

Os autômatos podem ser classificados em "determinístico", quando dados o estado corrente e um símbolo do alfabeto o autômato executa uma transição para um único estado determinado; ou o autômato pode ser definido como "não determinístico", significando que dados o estado corrente e um símbolo do alfabeto o autômato executa uma transição para um ou mais estados. É importante destacar que os autômatos finitos determinísticos e não determinísticos são equivalentes, ou seja, é possível transformar um autômato finito não determinístico em um autômato finito determinístico e vice-versa (HOPCROFT *et al*, 2002).

Uma linguagem em um autômato finito é o conjunto de todas as *strings* que resultam em uma sequência de transições de estados, desde o estado inicial até um dos estados finais (HOPCROFT *et al*, 2002).

A função transição de um autômato finito pode ser representada através de diagramas de transições ou por tabelas de transições. O diagrama de transição de estados é um grafo, onde cada nó representa um estado e as arestas são as possíveis transições (HOPCROFT *et al*, 2002).

Na Figura 4, o diagrama é composto de 3 (três) estados possíveis. O estado inicial é q_0, o único estado final possível é q_1, que é representado por um circulo duplo. Os arcos correspondem aos valores de δ. O alfabeto utilizado nesse diagrama é o binário $\sum = \{0,1\}$.

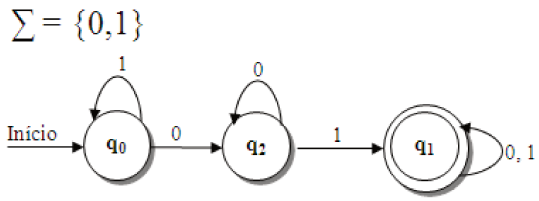

Figura 4: Exemplificação de um diagrama de transições (HOPCROFT *et al*, 2002)

A tabela de transição é uma representação convencional e tabular de uma função δ, que recebe dois argumentos e retorna um valor (estado). As linhas da tabela correspondem aos estados, e as colunas correspondem às entradas (HOPCROFT *et al*, 2002).

Na Tabela 2, pode-se observar que o estado inicial q_0 é indicado por uma seta e os estados finais por um asterisco, nesse caso só q_1 é considerado um estado final. Essa tabela é uma representação tabular do diagrama de transições da Figura 4.

Tabela 2: Tabela de transição de estados (HOPCROFT *et al*, 2002)

	0	1
→q_0	q_2	q_0
* q_1	q_1	q_1
q_2	q_2	q_1

3.6 Experimentos de injeções de falhas automatizadas baseadas em máquinas de estados

Os estudos de testes de *software*, qualidade de *software*, injeção de falhas, máquinas de estados e automação de testes contribuíram para o desenvolvimento dos experimentos de testes que serão detalhados na Seção 7.

Existem poucas referências que versam sobre injeção de falhas baseadas em casos de teste a partir de máquinas de estados. Uma metodologia que se encontra na literatura e que foi utilizada no contexto de aplicações espaciais foi CoFI (*Conformance and Fault-Injection*), que integra duas abordagens de teste existentes: teste de conformidade e técnica de injeção de falhas (AMBROSIO, 2006). O processo inclui uma abordagem para derivar casos de testes e de falhas, combinando conceitos de testes de conformidade de *software* com a técnica de injeção de falhas implementada por *software* (SWIFI, *Software-Implemented Fault Injection*) (AMBROSIO, 2006). O SWIFI tem sido utilizado na validação de dependabilidade de sistemas críticos (VOAS, 1998). A abordagem utilizou casos de testes derivados de máquinas de estados finitos e mostrou grande eficiência nas atividades de teste, além de melhorar a confiabilidade nos aplicativos de *software* espaciais (AMBROSIO, 2006).

Na próxima subseção contem informações que visam facilitar a criação de uma documentação para experimentos de testes de robustez, a fim de facilitar o entendimento do que foi feito, e simplificar a reprodução do experimento.

3.7 Documentação para testes de robustez de Software

O padrão de documentação de testes de *software* elaborado por MORAES & WAESELYNCK (2011) é formado por um conjunto de modelos no formato UML, que representam os casos de testes e ambientes computacionais utilizados para a obtenção dos resultados experimentais dos testes de robustez. O objetivo de se documentar os testes é esclarecer as particularidades envolvidas nos experimentos, facilitando o entendimento a respeito dos componentes utilizados, possibilitando que os experimentos possam ser reproduzidos para obtenção de resultados complementares ou confirmação dos resultados apresentados. No trabalho de MORAES & WAESELYNCK (2011), são descritos três casos de estudos, com base em trabalhos já publicados, onde a documentação de testes foi aplicada e o resultado foi validado com os respectivos autores.

Já existem outros padrões para testes de *software* como: *UML Testing Profile* (U2TP) e *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* (QoSFT). Porém esses padrões não possuem alguns detalhes, tais como:

- Características de testes que permitem que o sistema consiga lidar com entradas inesperadas;
- Documentar os testes quando falhas ou ataques devem ser considerados;
- Documentar o ambiente de testes, quando o injetor de falhas/ataques possuir um papel relevante.

Os objetivos desse novo padrão, derivado dos *profiles* anteriormente citados, é tratar da documentação de testes de robustez utilizando a UML, sugerindo elementos adicionais aos padrões já existentes. Adicionalmente, o modelo utilizado possibilita que os experimentos criados para testar a robustez do sistema possam ser representados, de modo que a comunicação entre os testadores e desenvolvedores seja facilitada.

O pacote proposto consiste no metamodelo, exibido na Figura 5. A função deste metamodelo é descrever os componentes necessários para injetar as falhas, monitorar o sistema, e armazenar os resultados de cada caso de teste para, posteriormente, analisar o comportamento do sistema quando ocorre uma falha.

Os componentes apresentados complementam componentes existentes nos outros padrões (tais como, o U2TP e o QoSFT). O <<*faultload*>> especifica o conjunto de falhas que podem ser injetadas no sistema, cada falha sendo representada por <<*faultloadElt*>>. O <<*Injector*>> fornece mecanismos para o testador realizar as campanhas de testes. O <<*workload*>> é um mecanismo para especificar a carga do sistema em teste, onde as falhas serão inseridas. Cada estímulo ao sistema é representado por <<*workloadElt*>>.

O <<*GoldenRun*>> representa a execução normal do sistema e é utilizado para capturar os resultados do comportamento normal do sistema. O <<*InjectionRun*>> representa o experimento em presença de falhas e possibilita observar o comportamento do sistema quando submetido à falhas.

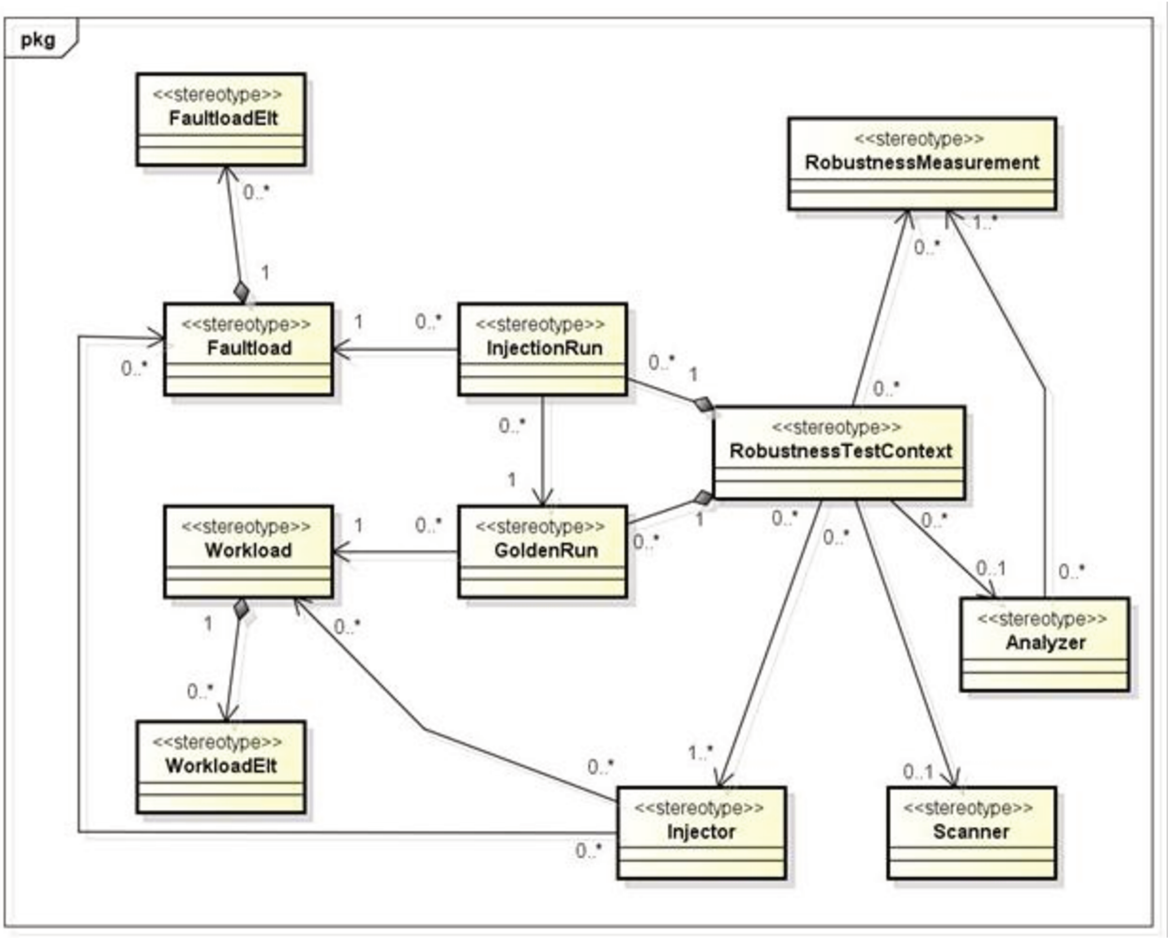

Figura 5: Pacote de Injeção de falhas e monitoramento, retirado de (MORAES & WAESELYNCK, 2011)

O metamodelo proposto inclui elementos que são utilizados quando injeção de ataques é necessário. Neste caso, o estereótipo <<*scanner*>> é utilizado para identificar previamente as vulnerabilidades dos sistemas para que, em um segundo momento, os ataques possam ser injetados com o intuito de obter acesso a recursos não públicos.

Este padrão de documentação foi utilizado neste trabalho para documentar os experimentos aplicados a rede GPON. Como neste trabalho é utilizada apenas a injeção de falhas, o componente com o estereótipo <<*scanner*>> não será representado.

4 Técnicas de testes estruturados

Nessa Seção serão abordadas as duas principais técnicas de testes utilizadas nos experimentos que estão descritos na Seção 6. A seguir serão apresentadas outras características de injeções de falhas e testes de modelos de estados.

4.1 Injeções de falhas

A confiabilidade de sistema computacional pode ser alcançada evitando-se a introdução de falhas e incluindo recursos de tolerância a falhas, que permitem que o sistema continue operacional mesmo que alguma falha tenha ocorrido no sistema (SOMERVILLE, 2005).

A técnica de injeção de falhas pode ser utilizada para verificar se o sistema é ou não tolerante a falhas. A injeção de falhas também tem como objetivo observar o comportamento do sistema na presença de falhas que foram deliberadamente incluídas, a fim de validar o sistema em análise.

A injeção de falhas aplicadas em testes de validações de sistemas produz resultados relevantes devido à capacidade de reproduzir falhas que possam ocorrer na fase operacional do sistema.

Alguns objetivos da injeção de falhas são definidos em HSUEH *et al* (1997):

- Identificar os gargalos da dependabilidade;
- Estudar o comportamento na presença de falhas;
- Determinar a cobertura de detecção de erros e mecanismos de recuperação;
- Evoluir os mecanismos de tolerância a falhas.

Todos os mecanismos de tolerância a falhas são baseados em suposições e devido a isso são designados para lidar com um conjunto específico de falhas e erros (VOAS & MCGRAW, 1998). Existem algumas técnicas de injeções de falhas, que podem ser aplicadas nos sistemas

computacionais, porém cada uma delas possui um alvo diferente na aplicação. No caso em que o experimento exija falhas permanentes, elas podem ser localizadas no *hardware*, por exemplo, em um determinado ponto do circuito. Porém se o interesse é corrupção de dados, o mais apropriado que esta falha seja inserida por *software* (HSUEH *et al*, 1997).

A seguir serão detalhadas as técnicas de injeções de falhas: por *hardware*, por *software* e por simulação.

4.1.1 Injeções de falhas por hardware

A injeção de falhas por *hardware* permite a perturbação de pinos e componentes internos de chips, o que não é possível através da injeção de falhas por *software*. Além de que é capaz de provocar falhas reais (ROSA, 2008).

Dependendo do tipo e localização das falhas elas podem ser divididas em dois tipos (HSUEH *et al*, 1997):

- Com contato: a falha está em contato com o sistema, por exemplo, alteração da voltagem ou da corrente;
- Sem contato: as falhas não estão diretamente ligadas ao sistema, ou seja, as falhas são externas, como radiação ou eletromagnetismo. Porém essas falhas não possuem um controle efetivo, devido a não precisão da emissão de íons ou do campo eletromagnético.

Porém na utilização na injeção de falhas por *hardware* são encontradas algumas dificuldades, tais como: na monitoração da ativação e da detecção das falhas, na complexidade para automatizar todo o processo e além de poder danificar o sistema alvo (ROSA, 2008).

4.1.2 Injeções de falhas por software

A técnica de injeção de falhas por *software*, do inglês *Software Implemented Fault Injection* (SWIFI), tem sido mais atrativa por ser mais econômica e possibilitar maior poder de controle e repetição dos experimentos (ARLAT *et al.*, 2003).

Essencialmente, existem duas formas de implementação de injeção de falhas por *software* (DE MILLO *et al*, 1994). A primeira é a injeção de falha no estado (alteração do estado), e a segunda é a injeção de falha no código (alterar o código fonte). A injeção da falha pode ser em tempo de compilação, onde a falha, localizada no código fonte ou no código *assembly*, é carregada antes do sistema ser executado. Ao executar o *software*, a falha injetada será ativada e causará um erro.

As injeções de falhas também podem ocorrer, em tempo de execução, quando *triggers* podem ser utilizadas explorando as seguintes situações (HSUEH *et al*, 1997):

- *Time-out*: Após um determinado tempo há a invocação de uma injeção de falha.
- Exceção: As falhas são inseridas quando existe a ocorrência de eventos ou condições de erros.
- Inserção de código: Instruções são adicionadas nos programas, para que as falhas ocorram antes de uma determinada instrução.

Para que o processo de injeção de falhas ocorra com sucesso, a instrumentação do *software* não deve perturbar o trabalho em execução no sistema alvo e até mesmo alterar a estrutura do *software* original, o que em muitos casos torna-se uma dificuldade. Outra deficiência é que não se podem injetar falhas em locais inacessíveis pelo *software* (HSUEH *et al*, 1997).

4.1.3 Injeções de falhas por simulação

A injeção de falhas por simulação é aplicada em modelos do sistema alvo. A sua principal vantagem é que sua aplicação pode ser feita durante o desenvolvimento do sistema, já nas fases iniciais do projeto (CHOI & IYER, 1992).

Essa técnica torna-se bastante atrativa porque durante o seu uso é possível controlar e monitorar as falhas injetadas, pois é possível controlar o local, o tempo, e a duração das falhas (CHOI & IYER, 1992). Mas também existe a dependência do nível de abstração do simulador que pode determinar um maior ou um menor grau de precisão dos resultados (CUNHA, 2000).

Porém a injeção de falhas por simulação pode-se mostrar imprecisa na modelagem de comportamentos de sistemas com falhas, pois em alguns casos não existem especificações para os estados falhos ou essas especificações podem ser vagas (STOTT, 1998).

4.1.4 Comparação entre abordagens de injeções de falhas

Um comparativo com as principais características da técnica aplicada em *software*, *hardware* e por simulação é realizado na Tabela 3.

Para a injeção de falhas em *hardware* é necessário um *hardware* especial que seja capaz de injetar essas falhas (HSUEH *et al*, 1997). Porém essa técnica é útil para verificar e analisar os mecanismos tolerantes a falhas aplicados no *hardware*.

Já na injeção de falhas por *software*, as falhas são introduzidas no nível de *software*, tais como o sistema operacional e suas aplicações (HSUEH *et al*, 1997), e devido a isso não é necessário nenhum *hardware* especial para utilizá-la.

Porém, quando ainda não existe um produto final, pode-se utilizar a injeção por simulação. Mas ela requer a construção de simuladores que se adaptem ao sistema em desenvolvimento.

Tabela 3: Comparação entre as abordagens de injeção de falhas

Abordagem	Vantagens	Desvantagens
Injeção de falhas por *hardware*	• Utilização de *hardware* e *software* reais; • Gera falhas reais;	• Requer um *hardware* especial para injetar as falhas, que normalmente possui um alto custo; • Pode danificar o sistema alvo; • Monitoração da ativação e da falha;
Injeção de falhas por *software*	• Não é necessária a utilização de *hardware* especial; • Técnica de baixo custo; • Possui um maior poder de controle e repetição dos experimentos; • Facilidade de inclusão de novos casos de testes;	• Pode alterar o sistema alvo, já que é uma técnica intrusiva; • Não se podem injetar falhas em locais inacessíveis ao *software*;
Injeção de falhas por simulação	• Há um maior controle do tempo, tipo de falha e do componente afetado pela injeção; • Pode ser aplicada nas fases iniciais do desenvolvimento do sistema;	• O custo com o desenvolvimento de simuladores é alto, além de ser complexo;

4.2 Testes baseados em modelos de estados

Uma máquina de estados finita (FSM - *Finite state machine*) é um modelo composto por um número finito de estados e transições. As transições são responsáveis por fazer a conexão entre os estados. A cada instante, a máquina possui somente um estado, e ao receber um evento de entrada, em resposta, a máquina atualiza seu estado novamente.

O teste baseado em modelos é uma técnica de teste de *software* que gera casos de testes a partir de modelos do comportamento do *software*.

A utilização de modelos durante o desenvolvimento de *software* é de grande valia, pois facilita o entendimento do sistema pela equipe. Para os desenvolvedores, auxilia na implementação e na solução de alguma falha. Para a equipe de testes, facilita o planejamento da execução dos testes, tais como ações e resultados esperados.

Porém, mesmo com as vantagens apresentadas, o teste baseado em modelos de estados ainda não é integrado ao processo de desenvolvimento de *software,* como também é minimamente empregado no ambiente industrial (NETO *et al*, 2007).

Existem algumas possíveis explicações para esse fato, uma delas é que durante o desenvolvimento de um projeto de *software*, devido a problemas com os prazos, o tempo de teste pode ser reduzido. E outro fator é a falta de conhecimento de técnicas, como o teste baseado em modelos de estados, por profissionais da área.

Um modelo que represente o *software* pode ser definido, tais como a estrutura e o comportamento do *software* (EL-FAR & WHITTAKER, 2002). Vários modelos podem representar o comportamento do *software*, tais como: fluxo de controle, fluxo de dados, UML e máquinas de estados.

Os modelos criados devem ser concisos e precisos. Concisos, de modo que não se gaste muito tempo para escrevê-los, tornando-os fáceis para validar em relação aos seus requisitos, mas devem ser suficientemente precisos para descreverem o comportamento que está sendo testado (UTTING & LEGEARD, 2007).

Os modelos finitos de estados são compostos por um conjunto de estados, um conjunto de eventos de entrada, e as relações entre eles. Dado um estado atual e sua entrada pode-se determinar o próximo estado do modelo (por favor, ref. à Seção 2.3).

As máquinas de Estados Finitos podem ser utilizadas para qualquer modelo que esteja descrito com um número finito de estados específicos (EL-FAR & WHITTAKER, 2002).

O processo de teste baseado em modelos pode ser dividido em cinco passos principais (UTTING & LEGEARD, 2007):

1. Modelar o sistema em teste e/ou seu ambiente.
2. Gerar testes abstratos a partir do modelo.
3. Concretizar os testes abstratos para torná-los executáveis.
4. Executar os testes no sistema
5. Analisar os resultados dos testes.

No primeiro passo de testes ocorre a escrita de um modelo abstrato do sistema que se deseja testar. Denomina-se modelo abstrato, pois ele deve ser simples e omitir muitos dos detalhes do sistema. Após a montagem do modelo, é aconselhável a utilização de ferramentas que verifiquem se o modelo está consistente e possui o comportamento desejado.

O segundo passo é a geração de testes abstratos a partir do modelo definido no primeiro passo. Devem-se definir alguns critérios de seleção dos testes, para não gerar um número infinito de testes possíveis. O resultado deste passo é um conjunto de testes abstratos, que são as sequências de operações a partir do modelo. Porém, os testes abstratos são carentes de alguns dos detalhes necessários do sistema em teste e não são diretamente executáveis. Os testes abstratos serão transformados no terceiro passo, utilizando muitas vezes ferramentas de transformação.

O quarto e quinto passos são mais tradicionais, e aplicados em outras técnicas de testes. Para a execução dos casos de testes são utilizadas ferramentas, que executam e armazenam os resultados dos testes gerados no terceiro passo. No quinto passo é realizada a análise dos resultados da execução do teste e são aplicadas as ações corretivas. Para cada caso de teste que reportar um defeito, deve-se determinar a falha que causou este defeito. A causa dessa falha pode ser devido a uma falha na geração do caso de teste, ou no modelo, ou nos documentos de requisitos.

4.3 Considerações Finais

Existem inúmeras técnicas de testes que podem ser aplicadas a diversos tipos de *software*.

Nesta Seção foram apresentadas algumas dessas técnicas que podem ser aplicadas em sistemas embarcados.

Em se tratando de *software* embarcado, a combinação das técnicas de injeções de falhas e teste de modelos de estados provê um processo de validação e verificação bastante completo. Pois é capaz de testar o sistema em situações de falhas reais e garantir uma cobertura adequada do sistema.

A próxima Seção abordará a combinação dessas duas técnicas com o uso da metodologia CoFI. Esta metodologia agrega os testes de conformidade à injeção de falhas e define passos para transformar a especificação em máquinas de estado finita, motivo pelo qual se mostrou adequada para este trabalho.

5 Conformance and Fault injection (CoFI)

Nessa Seção, a metodologia CoFI será caracterizada. Serão apontados os passos para sua execução, além das principais vantagens de seu uso.

5.1 Características da metodologia CoFI

A metodologia CoFI é uma combinação de testes de conformidade e testes por injeções de falhas, primeiramente desenvolvida para validar *software* embarcado de comunicação a bordo de satélites (AMBROSIO, 2005).

Os testes de conformidade permitem não somente revelar falhas que estão presentes no escopo do sistema, como também, demonstrar que o sistema em teste programa todas as capacidades requisitadas (BINDER, 2000).

Segundo Ambrósio (2005), a metodologia CoFI consiste na execução de uma série de passos para transformar uma descrição textual de serviços (da área espacial) em uma especificação formal (em máquina de estados finita). E afirma também: "A metodologia prevê a geração de uma sequência abstrata de teste apoiada por algoritmos de geração automática de testes a partir de especificações em máquinas de estados finitas. A sequência de teste é composta tanto por casos de teste (de conformidade) quanto por casos de falha (aqueles que serão executados pela técnica de injeção de falhas)". Esses passos podem ser observados na Figura 6, as entradas dessa metodologia são as Especificações textuais e a Arquitetura de teste, e como saída são obtidos os Casos de Testes e Casos de Falhas.

No processo CoFI, o comportamento normal e excepcional são modelados independentemente a partir da descrição do serviço de teste e derivação do caso de falhas.

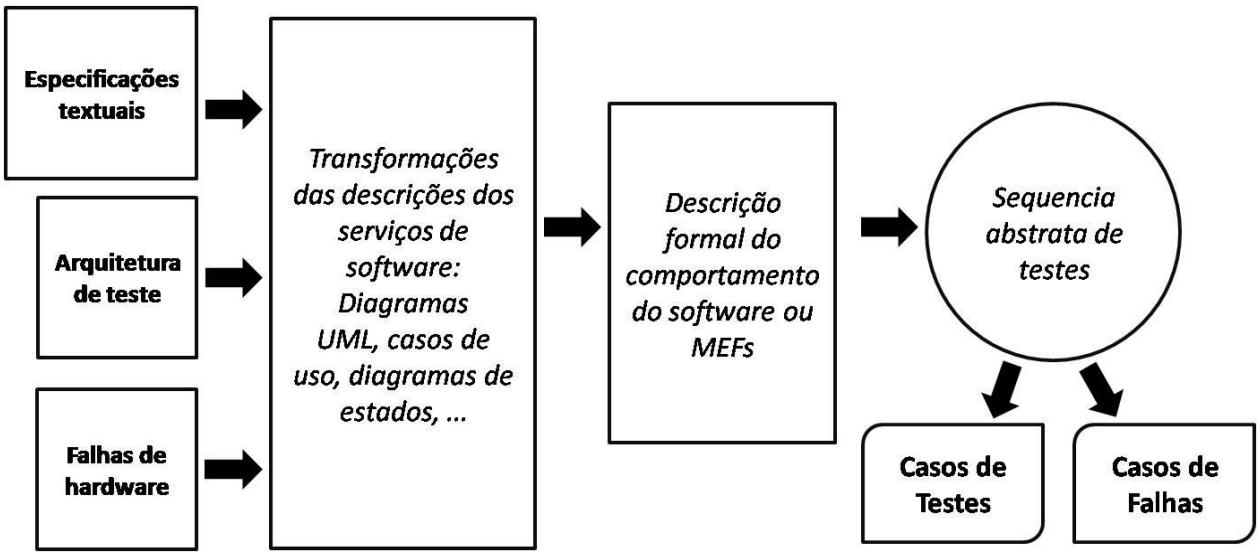

Figura 6: Visão geral da metodologia CoFI (baseado em AMBROSIO, 2005)

A metodologia CoFI pode ser resumida pelas fases descritas na Figura 7. Na primeira fase, é definido o que será testado. Nesse trabalho será testada a confiabilidade do *software* da OLT, quando o protocolo de comunicação, OMCI, é submetido a falhas. Na segunda fase, serão descritos os casos de testes e de falhas. Na terceira fase serão selecionados os casos de testes que serão executados. A quarta fase corresponde à execução dos testes selecionados na fase anterior. Na quinta fase é feita uma análise dos resultados dos testes executados, que gerará um relatório para a última fase.

A metodologia CoFI faz uso da ferramenta Condado (MARTINS *et al*, 2000), que permite a criação automática de casos de testes a partir das máquinas de estados.

Algumas conclusões e vantagens são obtidas com a utilização da CoFI (AMBROSIO, 2005) (AMBROSIO *et al*, 2006) (AMBROSIO *et al*, 2007):

- Possibilidade de reutilização dos casos de testes, pois eles são baseados nas especificações;
- Redução da distância entre a prática (geração de casos de testes a partir de definições textuais) e a utilização de métodos formais (especificação dos autômatos, máquinas de estados);
- Os modelos gerados são pequenos;

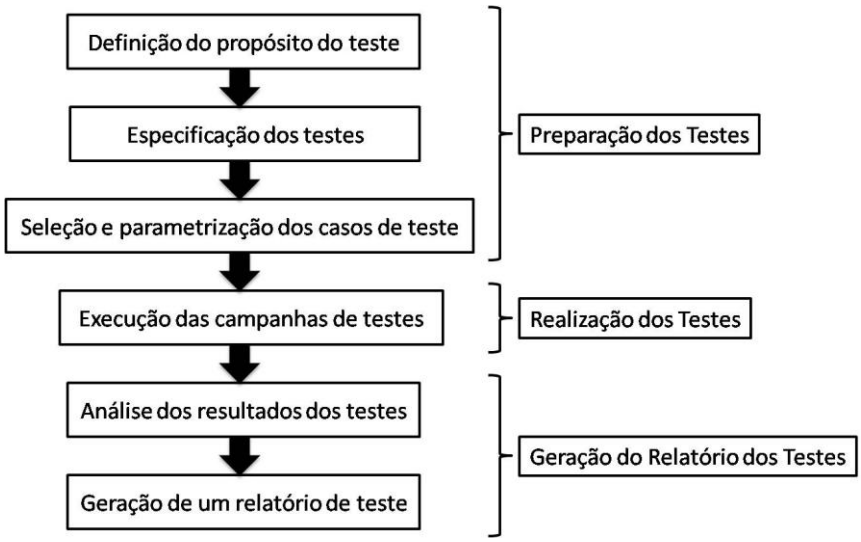

Figura 7: Fases da Metodologia CoFI (baseado em (AMBROSIO *et al*, 2006))

Segundo resultados reportados em trabalhos anteriores (AMBROSIO, 2005) (AMBROSIO *et al*, 2006) (AMBROSIO *et al*, 2007), defeitos de grande relevâncias foram revelados nas aplicações que foram testadas utilizando a CoFI.

5.2 Considerações Finais

A CoFI executado em aplicações espaciais, mostrou-se bastante robusta, devido a criação de máquinas de estados, e geração casos de testes e de falhas, abrangendo também a injeção de falhas. A metodologia também permite o planejamento da atividade de testes desde o início do processo de desenvolvimento do sistema, agregando mais qualidade ao produto final.

Devido a uma definição bem clara das etapas dessa metodologia, a CoFI pode ser estendida para outros sistemas que possuam *software* embarcado.

A próxima seção contem todos os detalhes dos dois experimentos de testes *software* realizados no sistema GPON. Além do uso dos conceitos e metodologias citados nesse mestrado tais como: injeção de falhas, testes baseados em modelos de estados, CoFI e documentação de testes de robustez.

6 Experimentos de testes

Antes da realização dos experimentos relatados nessa Seção, o GPON já havia sido submetido a outras técnicas de testes, tais como: testes caixa branca (análise de fluxos, verificação de ponteiros, entre outros), além de testes de regressão, que posteriormente também foram automatizados. Porém, essas técnicas se mostraram insuficientes para garantir uma maior dependabilidade do sistema. Devido a isso, se buscou a aplicação de novas técnicas, a fim de complementar os resultados que as técnicas anteriormente aplicadas trouxeram.

Os experimentos relacionados nesse trabalho tratam da injeção de falhas que emulam falhas físicas na OLT e emulam falhas de software no protocolo de comunicação OMCI. O primeiro experimento é uma emulação, por *software*, de falhas físicas injetadas automaticamente no sistema GPON, que causam a interrupção de comunicação da rede. Esse experimento tem como base as máquinas de estado do *software* da OLT, que representam o comportamento do *software*. O segundo experimento utiliza a metodologia CoFI (AMBROSIO *et al*, 2006) para preparar, executar e relatar os resultados dos testes. Os casos de testes desse segundo experimento consistem nas injeções de falhas de comunicação emuladas por *software* no protocolo OMCI, para que o *software* da OLT possa ser analisado.

Os defeitos encontrados nos resultados dos experimentos foram classificados de acordo com os seguintes critérios: (i) Baixa criticidade - são defeitos triviais ou melhorias a serem feitas, que não afetam o funcionamento do sistema; (ii) Média criticidade - são defeitos que podem afetar o desempenho da rede, mas não levam a interrupção do seu funcionamento; (iii) Alta criticidade - são defeitos que podem interromper parte do sistema ou comprometer o desempenho de parte da rede; (iv) Muito Alta criticidade - são defeitos que podem prejudicar totalmente o funcionamento do sistema ou de alguma funcionalidade essencial. Tal classificação foi determinada com base em resultados de experimentos obtidos no projeto GPON do CPqD além de relatos obtidos em RELIASOFT e ALCORN *et al* (2008), assim como a análise de impacto que tais defeitos poderiam ocasionar a seus usuários.

6.1 Primeiro Experimento

Esse experimento trata da emulação de falhas físicas no sistema GPON, através de injeção de falhas via *software*. O objetivo desse teste é validar os mecanismos de tolerância às falhas, desenvolvidos no *software* da OLT. Para validá-los, o sistema GPON pode ser analisado quando submetido a falhas, tais como interrupção do fornecimento de energia elétrica ou a quebra da fibra óptica de comunicação entre a OLT e as ONUs. Essas falhas são situações que podem ocorrer em um sistema em operação, ou seja, são falhas representativas desse tipo de sistema.

Em um sistema ideal, quando tais falhas ocorrem, após as fibras serem substituídas ou o sinal de comunicação ser recuperado, a conexão entre os dispositivos deve ser restabelecida. A OLT e as ONUs devem retornar ao estado anterior antes da interrupção da comunicação e devem reestabelecer a comunicação, sem qualquer interação humana.

Nessas situações, é inevitável que o usuário não perceba a ocorrência desses defeitos, porém eles devem ser solucionados o mais breve possível, para que os efeitos dessas falhas sejam minimizados.

6.1.1 Testes manuais

Os testes por injeção de falhas eram realizados manualmente (injeção de falhas físicas), onde a fibra era desconectada e reconectada, manualmente, ou um cabo de energia era retirado. Porém, essa técnica não era eficiente por dois motivos:

- Quando algum defeito era encontrado, ele era de difícil reprodução, pois era problemático definir o exato momento de desconexão e conexão da fibra ou da interrupção de energia;
- Não se conhecia a cobertura dos testes realizados.

A solução encontrada para resolver esses dois problemas foi a automação da injeção de falhas, visando o controle das falhas injetadas, que será descrito a seguir.

6.1.2 Fases para execução do experimento

Para a execução desse experimento, foram percorridas as fases definidas na Figura 8. No primeiro grupo, chamado Preparação dos Testes, na fase de Definição do Propósito de Teste foi definido o sistema alvo, o GPON. Também foram identificadas as falhas que seriam injetadas no sistema, falhas que emulam falhas físicas, interrompendo a comunicação no sistema GPON. Na segunda fase, denominada Análise das máquinas de estados, todas as máquinas de estados criadas para o desenvolvimento do *software* da OLT foram analisadas, para que na terceira fase, Criação dos Casos Testes, elas fossem utilizadas para a definição dos casos de testes. Essas duas últimas fases serão detalhadas na próxima subseção intitulada Modelos de estados.

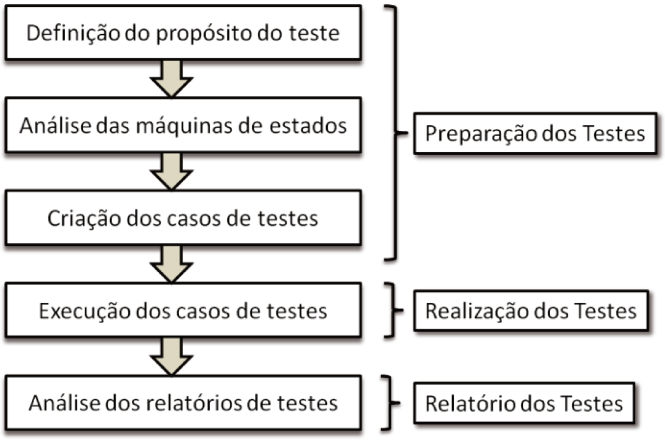

Figura 8: Fases do experimento de injeção de falhas que emulam falhas físicas

No segundo grupo denominado Realização dos Testes, todos os casos de testes elaborados manualmente, nas fases anteriores foram executados com o auxílio de um robô de testes, que será detalhado no próximo parágrafo. No último grupo, Relatório dos Testes, durante a de fase de Análise dos Relatórios de Testes é realizada uma a análise de toda execução através de relatório, com a finalidade de identificar os defeitos encontrados.

Para viabilizar a automação da execução dos testes, um robô de testes foi desenvolvido para injetar as falhas e executar os casos de testes no sistema GPON. O robô foi desenvolvido no âmbito deste trabalho e foi implementado na linguagem C, utilizando o sistema Linux. A

comunicação entre o robô e a OLT é feita através de *sockets* TCP/IP. O robô (Figura 9) controla o processo de injeção de falhas, e possui algumas funcionalidades tais como:

- Enviar comandos para a OLT;
- Receber os *logs* do sistema GPON;
- Enviar comandos para o OXC;
- Executar os casos de testes;
- Analisar os resultados obtidos.

O equipamento OXC é uma chave óptica, que nesse experimento é utilizada para conectar e desconectar as fibras que interligam as ONUs na OLT. Com a introdução da chave óptica, houve uma perda de potência óptica, assim como qualquer equipamento óptico poderia ocasionar, devido a isso, o número de ONUs utilizadas no experimento foi reduzido.

O *switch* conecta todos os dispositivos de rede envolvidos no experimento. Os casos de testes foram gerados manualmente e armazenados em arquivos de textos que, posteriormente, foram interpretados pelo robô de testes.

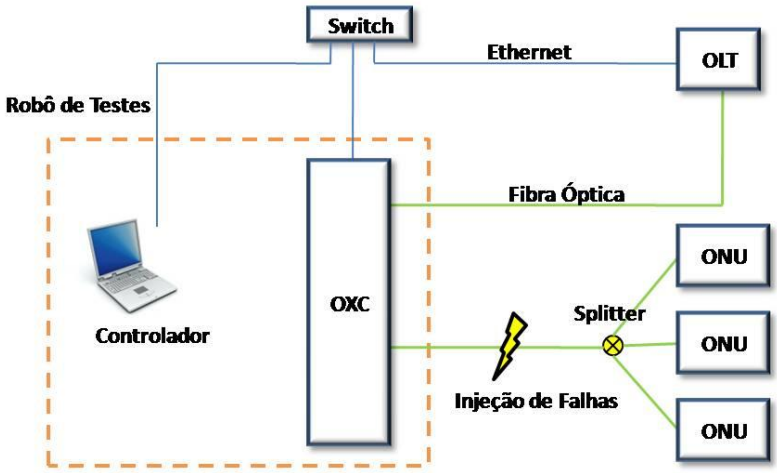

Figura 9: Robô de Testes (destacado por linhas tracejadas)

Os passos executados pelo robô para a injeção de falhas são representados através do diagrama de sequência da Figura 16. Uma instância do robô executa os casos de testes, selecionando instâncias do *workload* (i.e., troca de mensagens), através de comandos enviados para uma instância do sistema GPON e recebe a resposta de cada comando enviado, assim como possíveis eventos e alarmes do sistema. Quando o robô recebe um *log* do sistema GPON

com um estado esperado pelo caso de teste, ele envia um comando para uma instância da chave óptica (OXC) que realiza a desconexão das ONUs, injetando a falha. Após alguns segundos a conexão é restabelecida pela chave óptica, e o robô permanece recebendo os *logs* do sistema. Caso o robô detecte algum *log* não esperado, tal como um erro ou um aviso, o resultado desse caso de teste é classificado como defeito.

Após a execução dos casos de testes, o robô gera um relatório com todos os resultados e estatísticas obtidas, tais como, número de casos de testes executados com sucesso e com defeitos.

6.1.3 Modelos de estados

Para a elaboração dos casos de testes executados nesse primeiro experimento, foram utilizados máquinas de estados finitas do *software* da OLT. Nesse caso, as máquinas de estados já haviam sido previamente elaboradas pela equipe de desenvolvimento e foram utilizadas para planejar os casos de testes.

As falhas foram introduzidas em cada transição de estado, buscando reproduzir a ocorrência de quebras de fibras ou falta de fornecimento de eletricidade em todos os estados do *software* da OLT. Na Figura 10, uma das máquinas de estado da OLT é representada, assim como uma exemplificação dos locais onde é possível injetar as falhas. Devido a razões confidenciais, a máquina de estado foi modificada para a representação nesse trabalho.

A máquina de estado da Figura 10 é uma representação da ativação da ONU na OLT. Cada caso de teste injeta uma falha (interrupção de comunicação) em uma das transições. O conjunto de casos de testes tem como objetivo cobrir todas as transições de estados.

Os casos de teste gerados são realizados pelo robô de testes, e visam exercitar a máquina de estados da Figura 10. A execução inicia quando o comando de ativação da ONU é enviado para a OLT através do robô. O robô permanece no aguardo de um recebimento de um *log* de alteração de estado da ONU. Ao receber esse *log*, o robô imediatamente envia um comando para o equipamento OXC desconectar a fibra, introduzindo a falha. Enquanto o robô não recebe esses *logs*, o processamento do sistema GPON continua o fluxo normal de execução.

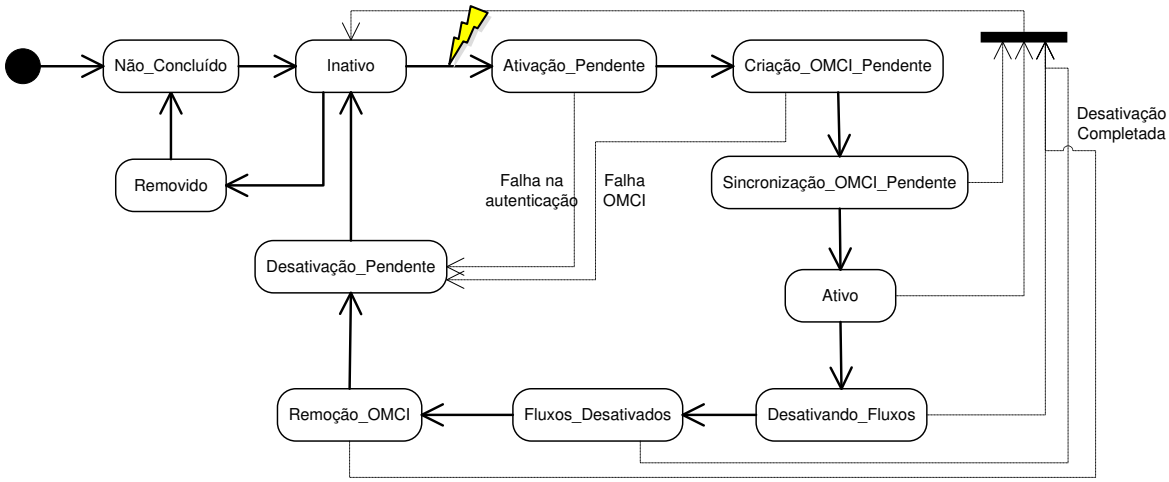

Figura 10: Máquina de estados da ativação da ONU na OLT

A execução da máquina de estados da Figura 11 pelo robô de teste inicia-se quando o comando de ativação da ONU é enviado para a OLT. O robô permanece no aguardo de um recebimento de um *log* de alteração de estado da ONU. Ao receber esse *log*, o robô imediatamente envia um comando para o equipamento OXC desconectar a fibra, introduzindo a falha. Enquanto o robô não recebe esses *logs*, o processamento do sistema GPON continua o fluxo normal de execução.

Algumas máquinas de estados do *software* da OLT se interoperavam, ou seja, duas ou mais máquinas de estados operam de forma simultânea, através de *threads* independentes, como exemplificados na Figura 11.

Na Figura 11, o estado ATIVO da Figura 10 foi detalhado. Observa-se que o estado ATIVO é composto por outras máquinas de estados: CONTROLE e VERIFICAÇÃO. A execução dos estados dessas "submáquinas de estado" ocorre em paralelo até que todas as máquinas cheguem ao estado final. Quando as duas máquinas de estados aninhadas alcançam seus estados finais, os controles dos dois subestados concorrentes se unem novamente em um único fluxo, e o estado da máquina é atualizado para ATIVO. Tais situações também foram cobertas pelos casos de testes.

Figura 11: Interoperação de máquinas de estado do software da OLT

6.1.4 Execução dos Testes

Na execução desses testes foram utilizadas as 14 máquinas de estados do *software* da OLT que estavam em operação na versão testada, cada uma delas apresentando de 3 a 34 transições de estados. A partir dessas máquinas de estados foram gerados 160 casos de testes que cobriam todas as transições das máquinas de estados abordadas.

Para a execução dos casos de testes, os cenários também foram alterados para efetuar uma validação mais abrangente dos resultados. Os fluxos nesses cenários representam a conexão física por onde trafegam as informações, que podem ser dados, vídeos ou voz. As alterações realizadas foram relacionadas à quantidade de ONUs conectadas e de fluxos ativos no sistema GPON. Tais cenários estão representados na Tabela 4.

Os resultados dos testes aplicados com essas alterações de cenários, não se mostraram muito diferentes dos cenários mais simples, ou seja, as falhas ocorriam independentes da carga colocada na rede.

Todos os testes desse experimento utilizaram somente a transmissão de dados, e não foram utilizados voz ou vídeo, pois no momento de experimento essas funcionalidades não estavam implementadas no projeto do GPON.

Tabela 4: Cenários de Testes explorados no experimento de injeção de falhas

Cenário de Testes	Situação
1	1 ONU conectada sem fluxo
2	3 ONUs conectadas sem fluxo
3	1 ONU conectada com 1 fluxo
4	2 ONUs conectadas com 1 fluxo
5	1 ONU conectada com 5 fluxos
6	2 ONUs conectada com 5 fluxos

O período em que o sistema ficou interrompido, através da chave óptica, não interferiu nos resultados dos experimentos, já que os resultados foram os mesmos para um tempo de interrupção de 2 segundos ou 2 minutos. Esse fato ocorreu, pois as máquinas de estados ficam paradas nos mesmo estados aguardando alguma nova entrada.

6.1.5 Documentação do Primeiro Experimento

A fim de facilitar o entendimento e a reprodução desse experimento, foram criados os diagramas que estão representados nesta Seção, tendo como foco o ambiente e os componentes do teste realizado. Esses diagramas são baseados no trabalho de Moraes e Waeselynck (2011).

Os elementos envolvidos no Primeiro Experimento estão exibidos na Figura 12. A OLT possui uma interface de comunicação com o protocolo OMCI, que se comunica através de outra interface com as ONUs conectadas. A OLT também possui outra interface de comunicação com os operadores da rede, denominada CLI (*Command line interface*), pela qual os operadores interagem com o sistema GPON através de comandos.

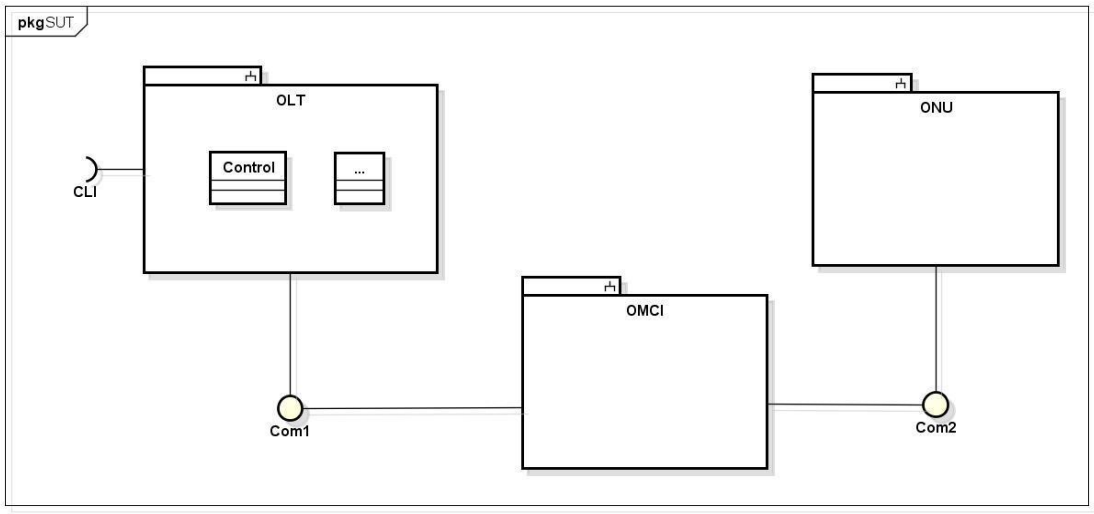

Figura 12: Sistema sob Teste (SUT, S*ystem under Testing*)

Baseado no metamodelo de Moraes e Waeselynck (2011), o estereótipo <<*faultload*>> é composto por todas as falhas que são passíveis de serem injetadas via *software* e que serão inseridas no ambiente, que nesse caso, será a interrupção de comunicação das ONUs com a OLT. O <<*workload*>> é composto pelas mensagens possíveis de serem trocadas entre os componentes do sistema GPON. O robô é responsável por interceptar as mensagens e injetar as falhas no sistema, recebe o estereótipo <<*injector*>>. Todo o funcionamento do sistema é monitorado por um programa que recebe e armazena os logs, chamado de Winlogger, que é uma ferramenta proprietária do CPqD. Após a execução dos casos de testes é gerado um relatório através de componente com o estereótipo <<*analyzer*>>, que analisa e resume os resultados gravados pelo Winlogger.

A arquitetura de teste está representada na Figura 13. O pacote de sistema em teste (SUT) já foi apresentado na Figura 12. A classe *Control*, controla o sistema GPON, e o equipamento OXC controla a injeção de falhas no sistema. O OXC se comunica com o robô através da interface IReqRobo. O pacote FIM_GPON_1 representa os componentes utilizados para conduzir os testes por injeção de falhas, isto é, o robô, o Winlogger e o *Analyzer*. O GPON_Suite representa os casos de testes que serão utilizados para validar o sistema.

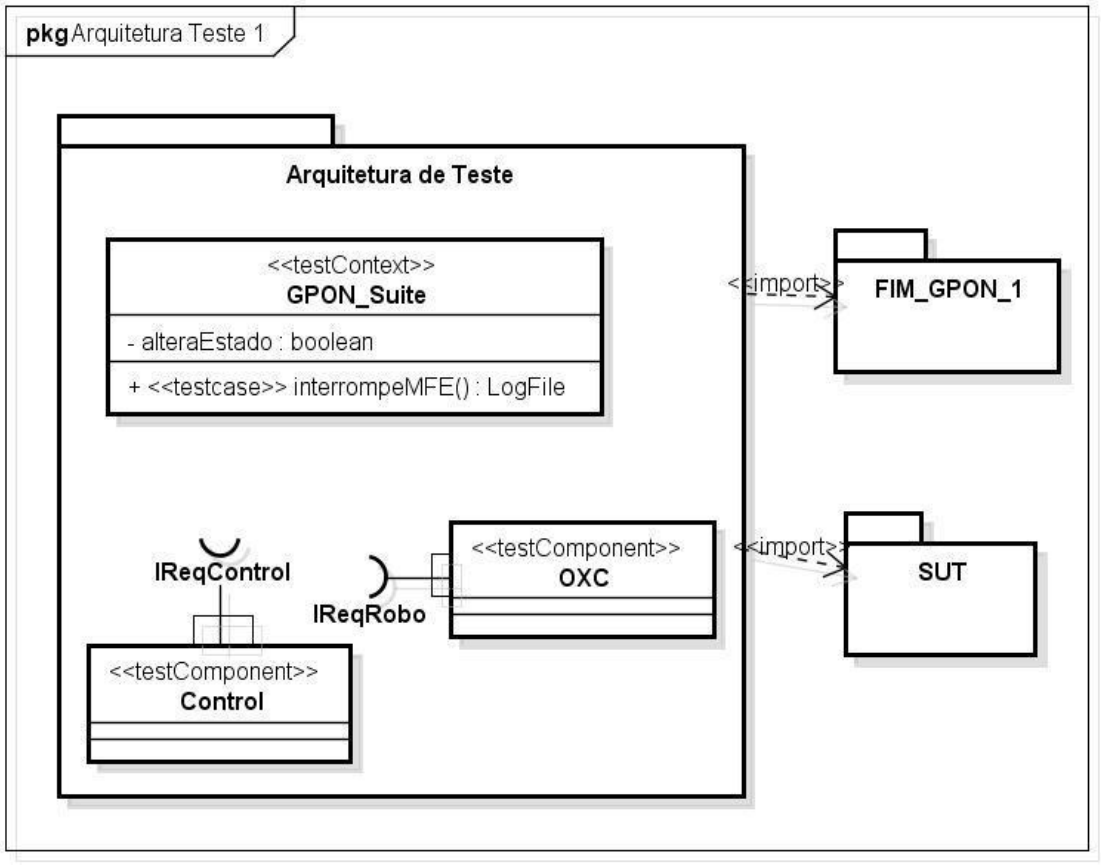

Figura 13: Arquitetura de Teste (Primeiro Experimento)

Na Figura 14, é representada a estrutura interna do contexto de teste, onde é exibida a ligação dos pacotes SUT e FIM_GPON_1.

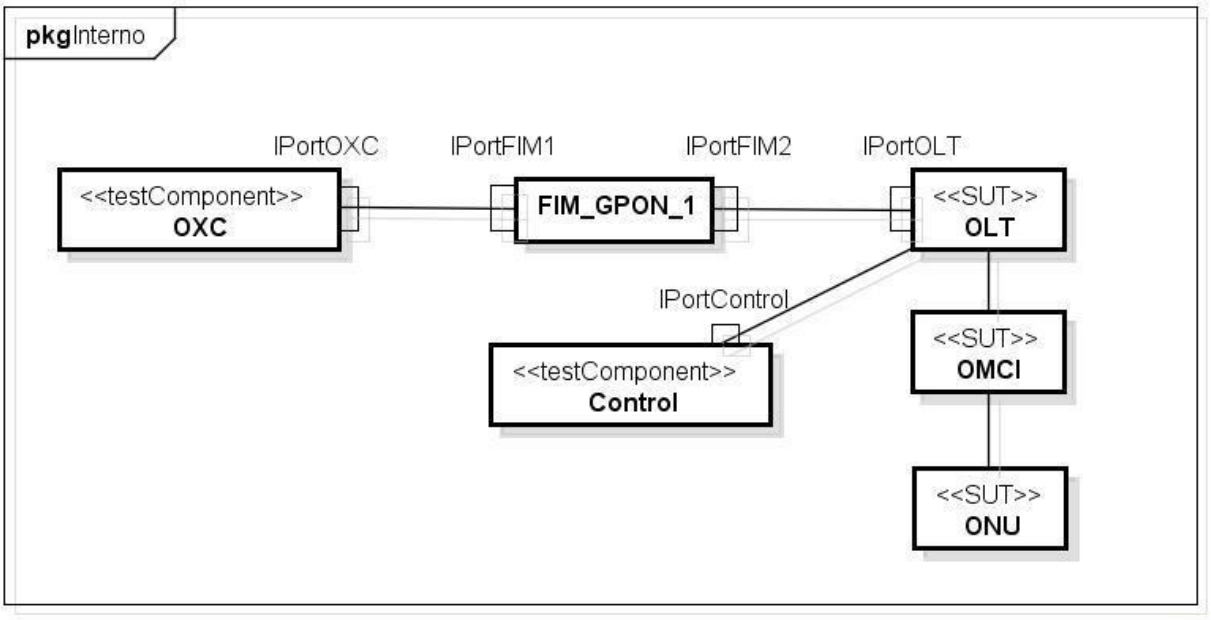

Figura 14: Estrutura interna do Contexto de Teste (Primeiro Experimento)

A Figura 15 contém um diagrama de sequência do comportamento do sistema GPON, quando não existem falhas. Essa execução é conhecida como *Golden Run*. Nesse diagrama, o robô também pode ser representado como o usuário do sistema, enviando comandos ao sistema, e recebendo respostas. O equipamento OXC não faz nenhuma interação nesse ambiente.

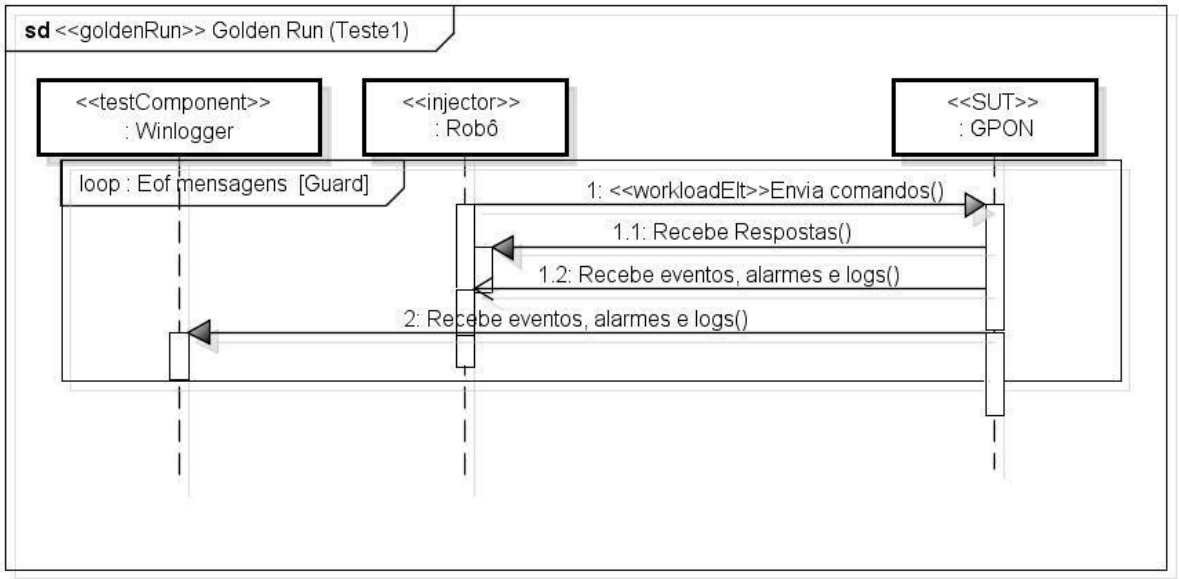

Figura 15: Golden Run (Primeiro Experimento)

Na Figura 16 é exibido o diagrama de sequência que representa o comportamento do sistema quando é inserida uma falha. Nesse ambiente, o robô envia os comandos ao sistema GPON, e recebe resposta. Quando o robô recebe um *log* contendo o estado esperado pelo caso de teste, o robô envia um comando para o OXC, que injeta uma falha através de uma interrupção de comunicação dentro do sistema GPON.

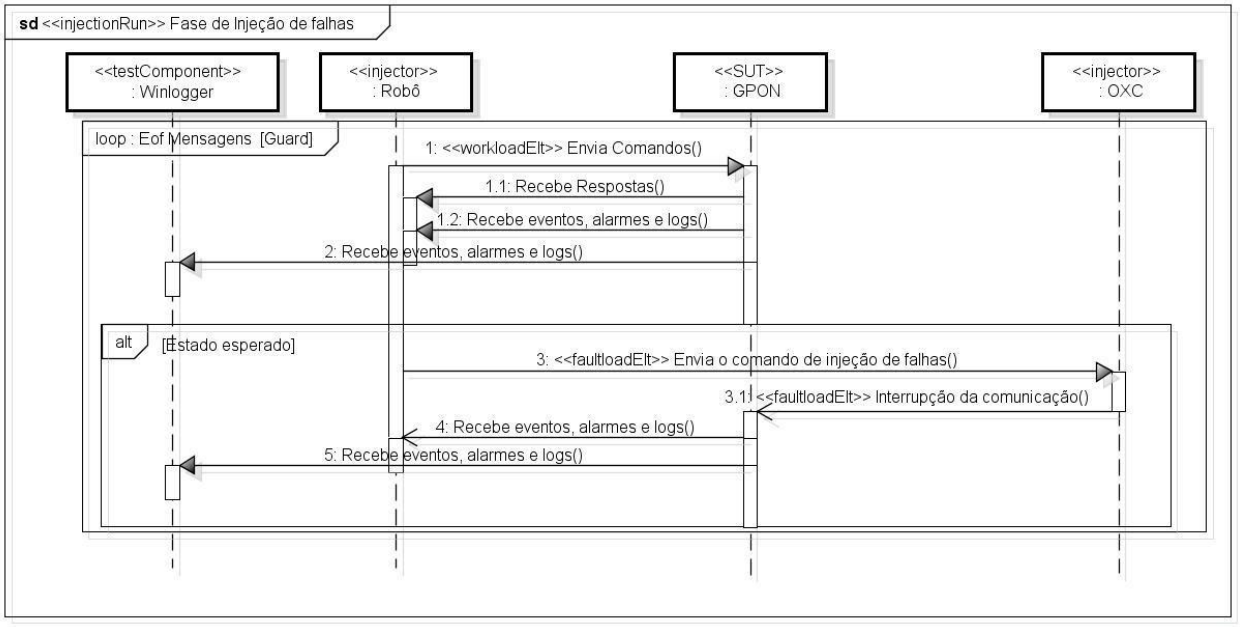

Figura 16: Injeção de falhas via Robô (Primeiro Experimento)

O procedimento experimental dos testes realizados é apresentado na Figura 17. Esse diagrama exibe os passos de execução dos casos de testes. Primeiramente, para a execução desse teste foi executado a *Golden Run* (Figura 15), procedimento que obtém os resultados do comportamento normal de execução, para que posteriormente fossem usados como parâmetro de comparação em relação aos resultados obtidos quando os mesmos experimentos são executados em presença de falhas (Figura 16). Antes da execução de cada caso de teste, o ambiente foi reiniciado, para que um caso de teste não interferisse na execução do próximo caso de teste. Para finalizar, um árbitro (Figura 18), que é um algoritmo de verificação, executa a análise se o caso de teste foi executado com sucesso ou não.

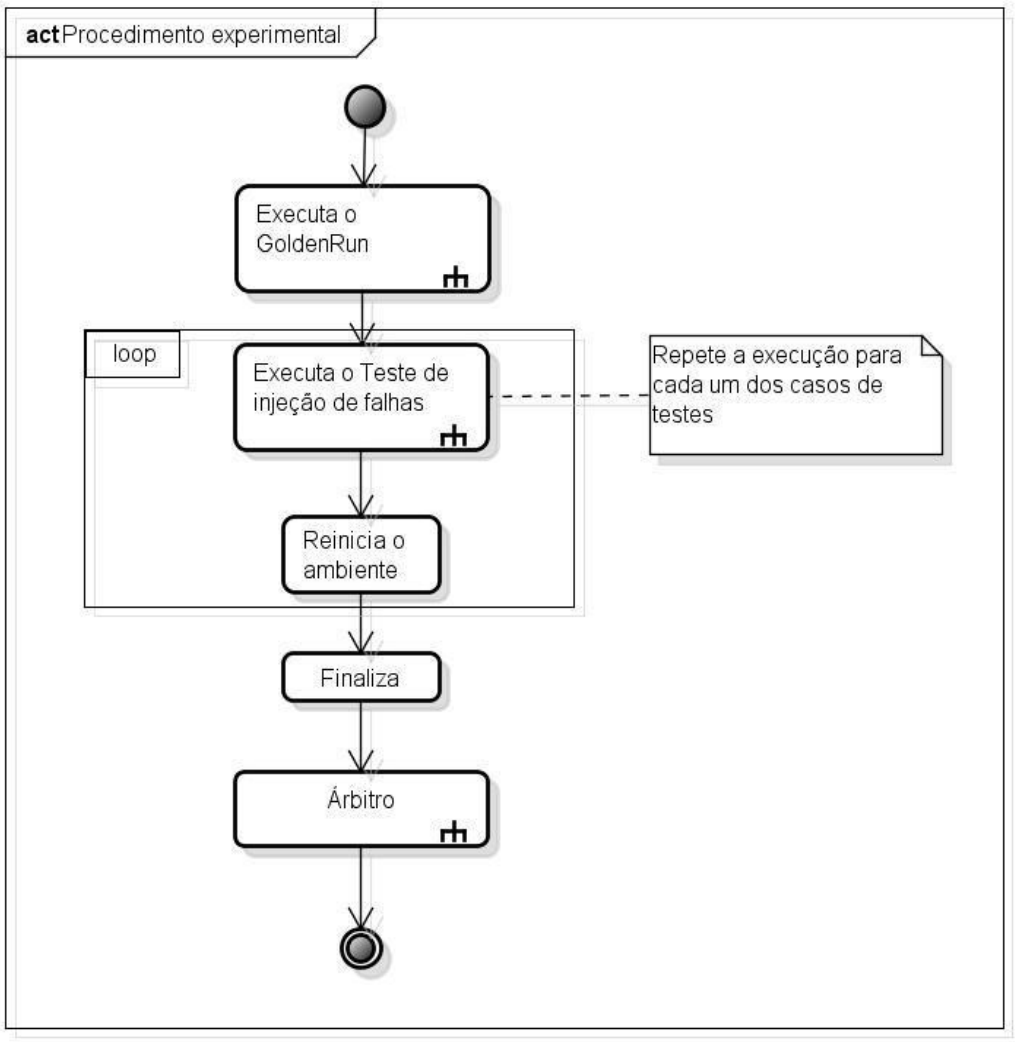

Figura 17: Procedimento experimental de testes via Robô

Para se definir quando um caso de teste teve resultado de sucesso ou não, foram feitas comparações dos *logs* recebidos de aplicação executada sem falhas (*Golden Run*) e com falhas (*Injection Run*), como demonstradas na Figura 18. Nesse experimento, o árbitro está inserido no robô de testes, que ao receber um log com erro do sistema GPON, já caracteriza o caso de teste com insucesso. Caso seja constatado que o caso de teste teve um defeito como resultado, ele é classificado de acordo com o *Failure Mode* (ou seja, a sua criticidade: Muito alta, alta, média e baixa).

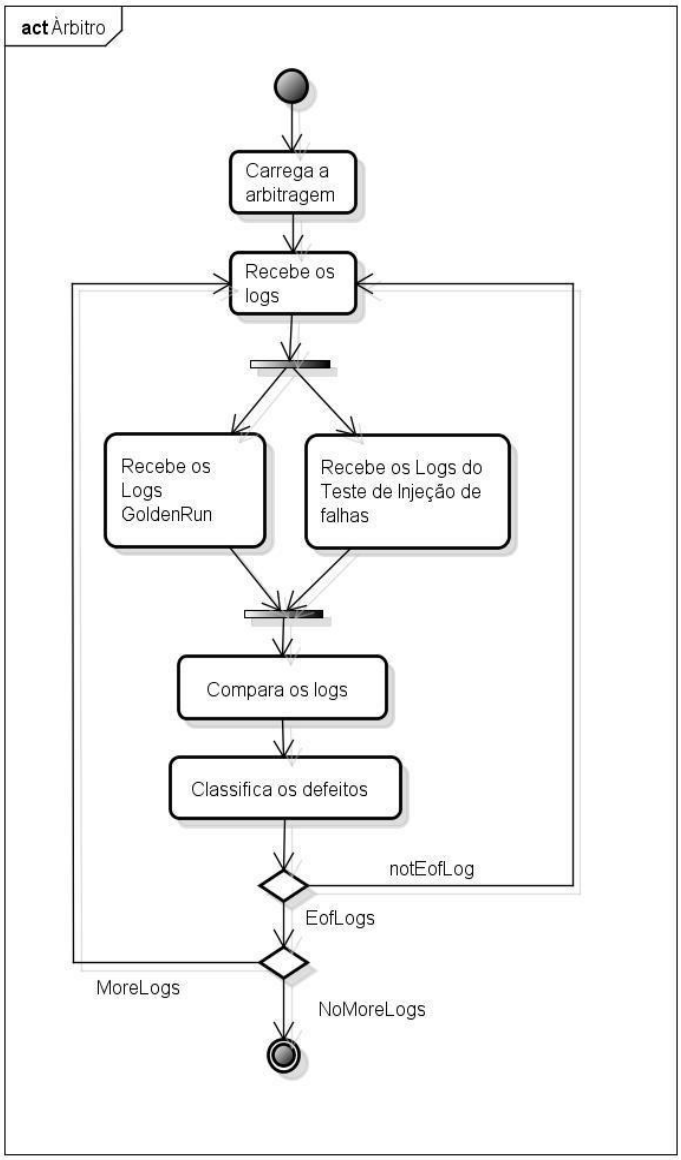

Figura 18: Arbitragem para os resultados dos casos de testes

6.1.6 Resultados do Primeiro Experimento

Após a montagem do ambiente para o experimento da Figura 9, e a execução dos casos de testes, conforme a Tabela 4, os resultados obtidos na execução são os apresentados na Tabela 5. Na primeira coluna da Tabela 5 encontram-se as máquinas de estados, que é

seguido pelo número de transições de cada máquina de estado. A quantidade de defeitos de média criticidade encontrados está localizada na coluna 3, e a quantidade de defeitos de alta criticidade está na coluna 4. A última coluna contabiliza o número total de defeitos encontrados.

Pode-se observar que as máquinas de estados representadas como D e E apresentaram um número maior de defeitos em relação às demais. Uma explicação para essa ocorrência é que essas máquinas possuem maior complexidade por interagir com outras máquinas, e, portanto são mais críticas.

Tabela 5: Resultados da execução dos testes do primeiro experimento

Máquina de estado	Quantidade de transições	Defeito de Média Criticidade	Defeito de Alta Criticidade	Total de defeitos
A	12	2	0	2
B	4	3	0	3
C	3	0	0	0
D	22	6	2	8
E	12	11	0	11
F	4	2	0	2
G	6	0	0	0
H	8	0	0	0
I	24	0	0	0
J	33	6	2	8
L	12	2	1	3
M	4	1	1	2
N	12	0	4	4
O	4	0	4	4
Totais	**160**	**33**	**14**	**47**

A Figura 19 apresenta outra representação dos defeitos encontrados, onde se pode observar que em exatamente trinta por cento (30%) das transições dos estados foram encontrados defeitos.

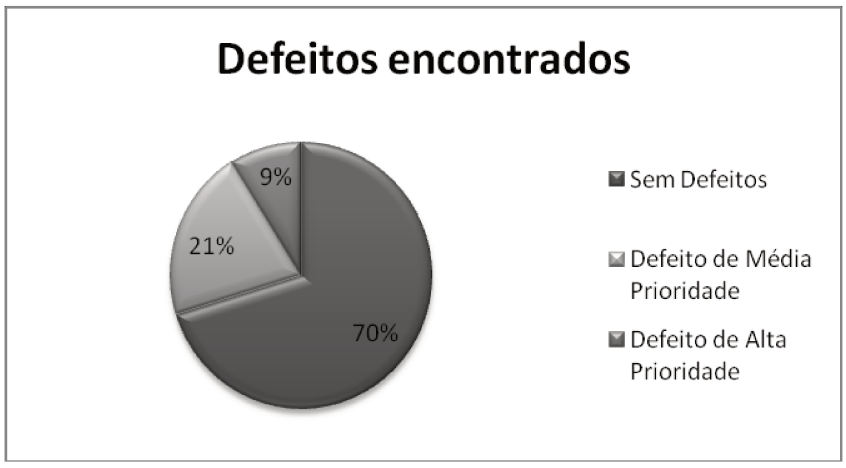

Figura 19: Defeitos encontrados por emulação de falhas físicas por software

Todos os defeitos encontrados nesse experimento não tinham sido observados anteriormente, quando foram utilizadas outras técnicas de teste, como, por exemplo, durante a execução dos testes de regressão que eram compostos por 850 casos de testes (FADEL *et al*, 2011).

Os defeitos encontrados anteriormente com a utilização de outras técnicas, na maioria dos casos, eram de criticidade menor. Além de serem de outro escopo, como defeitos causados por má interpretação dos requisitos dos comandos, que eram facilmente identificados através dos testes de regressão.

Para uma operadora, se tais defeitos ocorressem a seus usuários, e esses defeitos chegassem a afetar a ONU, a operadora teria que enviar um técnico até a residência ou empresa do usuário, para refazer determinadas configurações da ONU. Nesse caso, a OLT perderia o gerenciamento dessas ONUs. Outra situação que poderia ocorrer, seria a necessidade do operador reconfigurar todas as ONUs da rede, devido ao travamento da OLT e provável necessidade de reinicialização da central. Essa última situação poderia afetar todos os usuários da rede, que teriam o seu serviço interrompido até o restabelecimento da OLT.

Os resultados desse experimento também foram publicados no artigo "*Automated Validation of Embedded Optical Network Software*" apresentado no *Fifth Latin American Symposium on Dependable Computing* (LADC), em abril de 2011 na categoria de *Industrial Track*, que se encontra no Anexo A deste trabalho.

6.2 Segundo Experimento

A segunda técnica explorada neste trabalho é injeção de falhas no protocolo de comunicação das redes GPON, denominado OMCI. O objetivo desse segundo teste é avaliar se o *software* da OLT é tolerante a falhas que podem ser introduzidas nas mensagens recebidas do protocolo OMCI. Para essa validação, os dados das mensagens do protocolo OMCI serão manipulados, para que causem um comportamento fora do esperado na OLT.

Nesse experimento foi utilizado a metodologia CoFI (Ambrosio *et al*, 2006), que integra duas abordagens de teste existentes: teste de conformidade e injeção de falhas.

Essa segunda fase de testes também teve como base os modelos de estados que, nesse caso, foram gerados durante o desenvolvimento deste trabalho após estudos realizados no protocolo e normas que regem o OMCI (ITU-T G.984.4). Nas especificações da norma ITU-T G.984.4 não existem máquinas de estados que mostram o funcionamento desse protocolo, em relação a troca de mensagens.

As máquinas de estados geradas contêm o comportamento normal, as exceções especificadas, os caminhos furtivos (eventos normais ocorridos em momentos inesperados) e as falhas físicas.

De maneira geral, os elementos desse experimento são representados na Figura 20. As mensagens recebidas pela OLT serão interceptadas e alteradas, injetando falhas características de comunicação desse ambiente, para que o *software* da OLT seja avaliado.

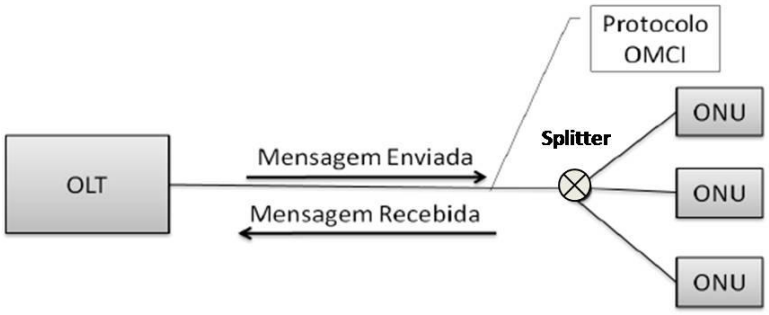

Figura 20: Segundo experimento de testes

6.2.1 Modelos de estados

Para esse experimento foram detalhados modelos de estados que representam o comportamento normal e com exceções das mensagens OMCI dentro do *software* da OLT. Os outros comportamentos (caminhos furtivos e falhas de *hardware*) não serão detalhados, pois não foram testados nesse momento. O caminho normal está detalhado na Figura 21, e representa um fluxo de execução sem falhas.

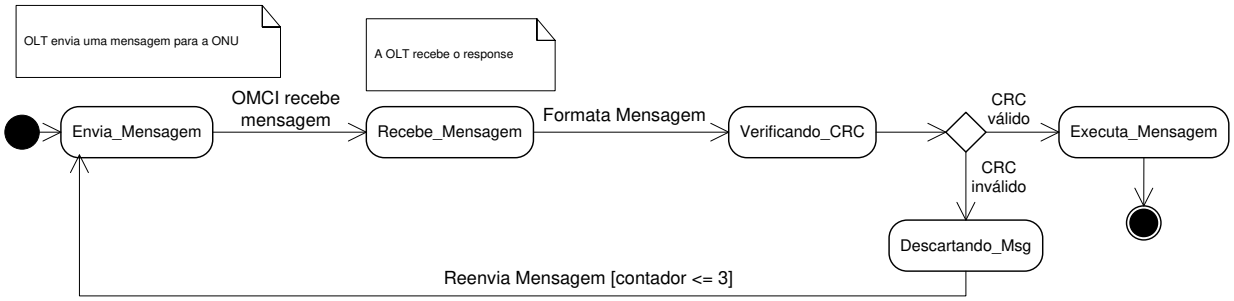

Figura 21: Fluxo normal de execução das mensagens OMCI na OLT

Na Figura 22, é apresentado o modelo de estados que representa o comportamento da OLT quando o pacote de mensagem é submetido a falhas.

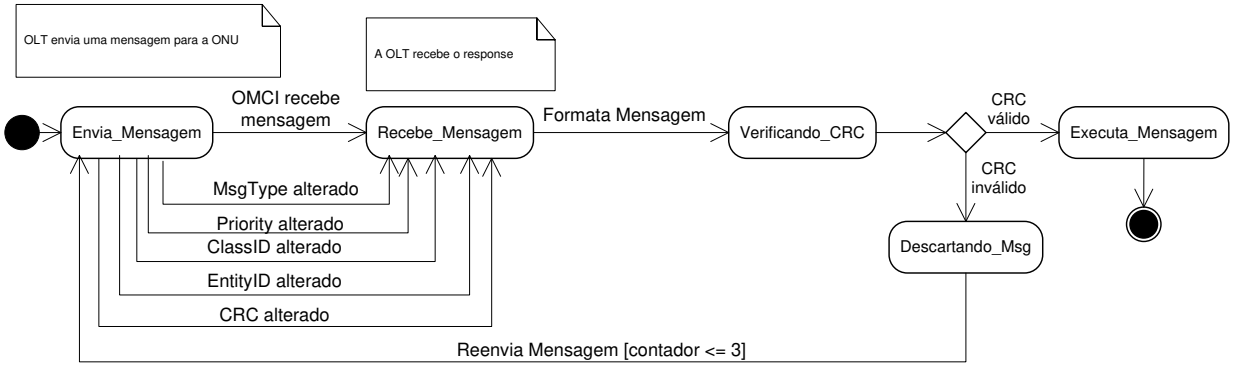

Figura 22: Fluxo de exceções das mensagens OMCI na OLT

Esses comportamentos detalham o funcionamento do pacote de mensagem OMCI ao chegar à OLT. Todas as mensagens ao chegarem à OLT têm o CRC validado, para a execução da mensagem. Se o CRC é invalidado a mensagem é descartada e solicitada novamente (Figura 21). Em presença de falhas (Figura 22), o comportamento é o mesmo, todas as mensagens têm o CRC verificado para serem executadas.

6.2.2 Execução dos Testes

A execução dos testes desse experimento ocorreu em um ambiente que possui uma OLT e uma ONU conectada. Para a execução de injeção de falhas, o componente de *software* do OMCI, localizado na OLT, foi alterado manualmente, com a finalidade de que as mensagens chegassem alteradas na OLT. A única automação utilizada nesse processo foi a utilização de *scripts* para a execução dos comandos.

Para esse experimento foram gerados 60 casos de testes, porém somente 40 deles foram executados. O motivo pelo qual um terço dos casos de testes foi descartado é que algumas funcionalidades não estavam completamente implementadas. A ferramenta Condado (MARTINS *et al*, 2000) foi utilizada para a geração automática desses casos de testes.

Os testes executados abrangeram as alterações dos campos das mensagens nos processos de MIB *Upload* e *Software Download*. O MIB *Upload* representa a sincronização da ONU, isto é, anuncia todos os serviços por ela providos. Quando a ONU é ativada pela OLT, esses mesmos serviços são criados na OLT. Outro processo explorado foi o de *software download*, que é a atualização do *software* da ONU, realizado pela OLT. Em ambos os processos testados ocorre uma troca de mensagens bastante intensa entre a OLT e a ONU.

Um exemplo de execução de caso de teste é no processo de MIB *Upload*, onde há a consulta de parâmetros de entidades da ONU. Nesse processo, a OLT envia uma mensagem do tipo *get*, e recebe uma mensagem do tipo *get response* da ONU, com um CRC calculado pela ONU. Quando a mensagem do tipo *get response* chega a OLT, ela verifica se o CRC é válido. No entanto, se a OLT recebe um *get response* de outra entidade ou com um CRC inválido, a

OLT deve descartar essa mensagem, solicitar uma nova mensagem para ONU e manter-se estável.

6.2.3 Documentação do Segundo Experimento

Nessa Seção serão detalhados alguns diagramas a fim de que esse teste possa ser facilmente reproduzido em ambientes semelhantes a esse, que também são baseados no trabalho de Moraes e Waeselynck (2011).

Para a execução desse segundo experimento, os elementos do sistema em teste são os mesmos utilizados no primeiro teste, e estão representados na Figura 12.

Assim como no primeiro experimento, o estereótipo <<*faultload*>> é composto por todas as falhas que serão injetadas por *software,* que nesse caso, serão a alteração do conteúdo das mensagens recebidas (falhas de comunicação). Essas alterações, que emulam as falhas injetadas, serão feitas nos parâmetros do pacote OMCI. O <<*workload*>> é composto pelo conjunto de mensagens que podem ser trocadas entre os componentes do sistema GPON. Nesse experimento não existe ferramenta de automação, e o relatório é gerado manualmente. Também é utilizado o Winlogger, para o monitoramento dos *logs* recebidos.

Na Figura 23, é apresentada a arquitetura de testes utilizada, que é composta pelo FIM_GPON_2 (constituído pelo Winlogger e pelo *Analyzer*) e pelo SUT. No GPON_Suite são apresentados os parâmetros da mensagem OMCI que podem ser alterados, e os casos de testes utilizados para esse experimento.

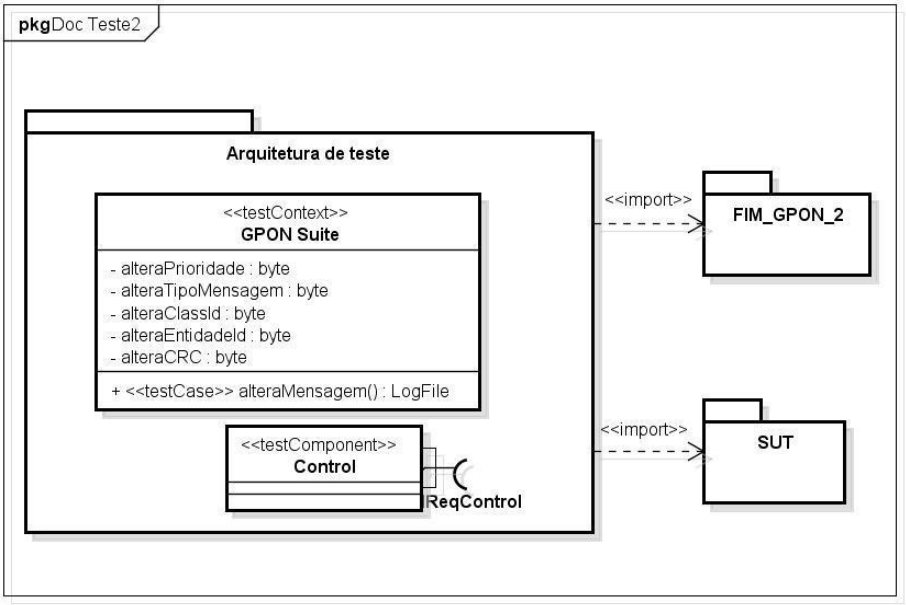

Figura 23: Arquitetura de Teste (Segundo Experimento)

Na Figura 24 é apresentado o pacote de dados. Esse pacote contém todos os parâmetros componentes da mensagem OMCI que podem ser alterados.

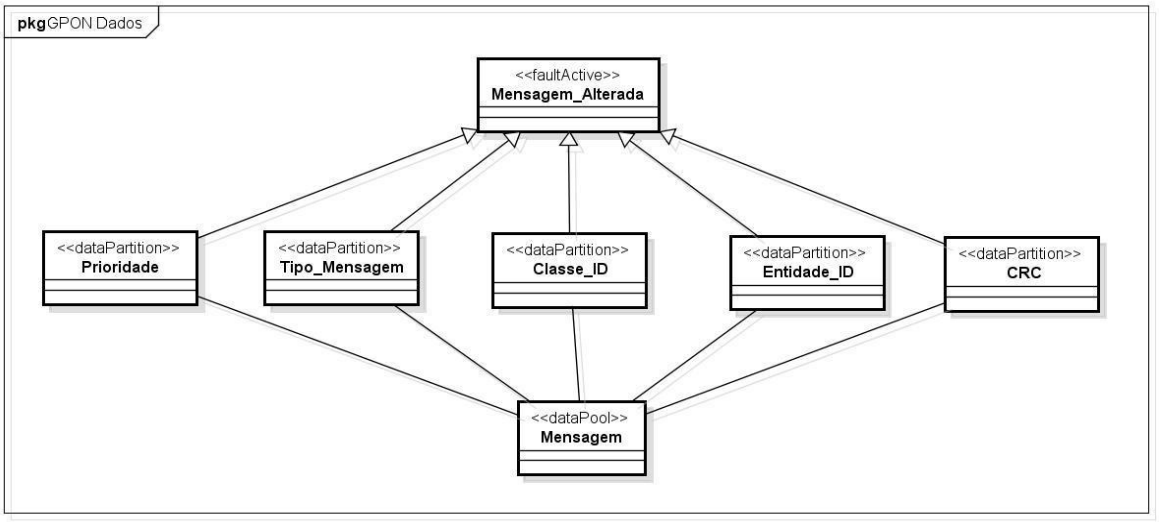

Figura 24: Pacote de Dados (Segundo Experimento)

Na Figura 25, é exibida a estrutura interna do contexto de teste que é composto pelos pacotes FIM_GPON_2 e SUT.

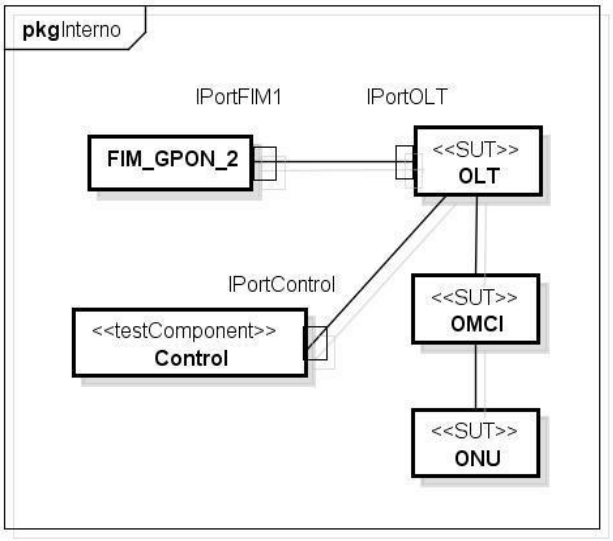

Figura 25: Estrutura interna do contexto de teste (Segundo Experimento)

Na Figura 26, é apresentado a *Golden Run* desse segundo experimento, que representa o fluxo de mensagens dentro do sistema GPON sem falhas. Todas as mensagens passam pelo OMCI, antes de chegarem a OLT ou na ONU.

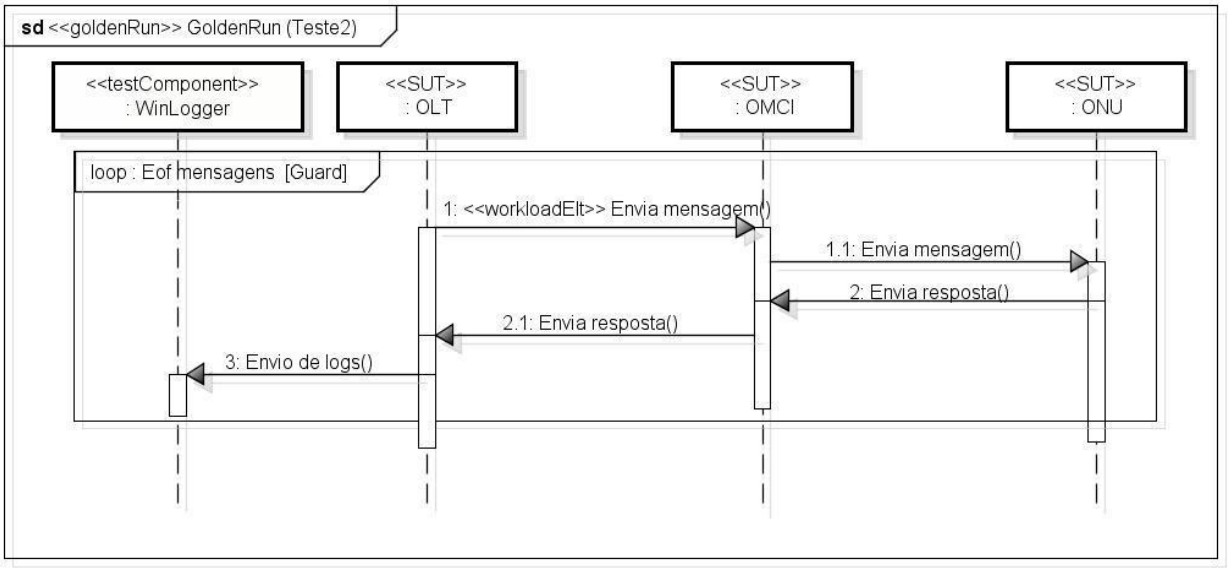

Figura 26: Golden Run (Segundo Experimento)

Na Figura 27, é exibida a sequência de mensagens quando as mensagens do OMCI são submetidas a falhas. Nesse caso, as mensagens que trafegam pelo protocolo OMCI interno à OLT têm seus parâmetros alterados e são entregues à própria OLT. Quando o sistema da OLT

receber alguma mensagem fora dos padrões, ela deveria requisitar novamente a mensagem, com o intuito de receber a mensagem, que vem da instância externa do protocolo OMCI, sem erros. Na OLT esse processo é feito por até 3 (três) vezes, caso a mensagem recebida até esse momento não esteja correta, o sistema descarta e exibe ao operador um *log* com erro.

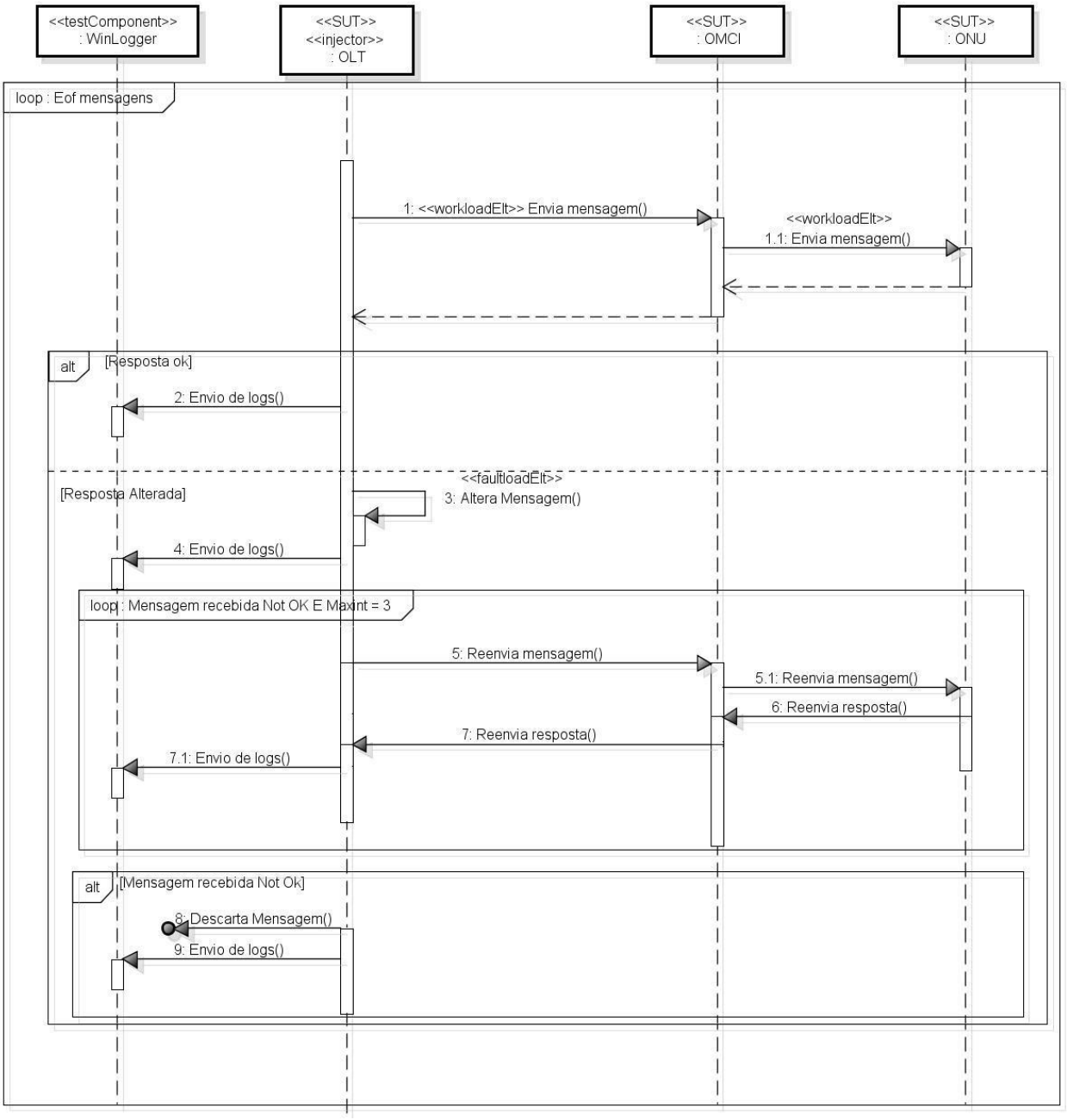

Figura 27: Fase de injeção de falhas (Segundo Experimento)

O procedimento experimental e o de arbitragem são os mesmos utilizados no primeiro experimento, e estão representados respectivamente pelas Figuras 17 e 18. Porém a arbitragem

nesse experimento foi realizada manualmente, devido a não utilização de ferramentas de automação para a execução dos testes.

6.2.4 Resultados do Segundo Experimento

Os resultados desse experimento encontram-se na Tabela 6. Na primeira coluna é especificada a falha inserida no pacote de mensagens OMCI. Na segunda coluna são colocados os casos de testes que tiveram o comportamento adequado. E nas colunas seguintes estão os defeitos distribuídos pela sua criticidade. No total, foram executados 10 casos de testes para cada tipo de alteração da mensagem. Somente a prioridade da mensagem não foi utilizada nos testes, pois o tratamento para esse tipo de mensagem ainda não estava implementado.

Tabela 6: Resultado dos Testes de injeção de falhas no protocolo OMCI

Teste	Passou	Defeito de média criticidade	Defeito de Alta criticidade	Defeito de Muito Alta criticidade
Tipo Msg Alterado	0	7	3	0
ClassID Alterado	0	4	2	4
EntityID Alterado	0	5	1	4
CRC Alterado	9	0	0	1
Totais	**9**	**16**	**6**	**9**
Total de Defeitos				**31**

No gráfico da Figura 28 esses resultados podem ser visualizados de forma mais clara, onde somente 23% dos casos de testes foram executados com sucesso, além de que se pode observar uma grande porcentagem de defeitos de muito alta criticidade (22%).

Os resultados mostram que apesar do protocolo OMCI possuir várias formas de garantir o correto envio de pacotes de mensagens, a OLT não foi tolerante às falhas introduzidas nessas mensagens.

Anteriormente à execução desses experimentos, a interface de recebimento de mensagens do OMCI na OLT não havia sido testada, pois não havia formas de testar a sua robustez.

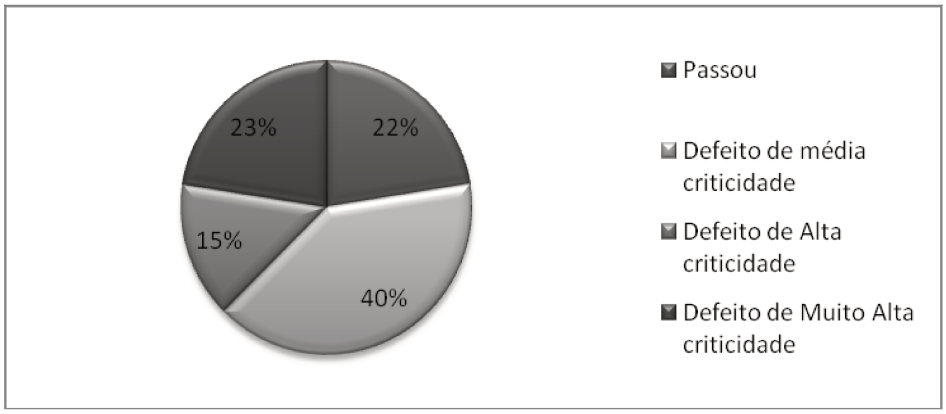

Figura 28: Resultados dos Testes de injeção de falhas no protocolo OMCI

6.3 Experimentos complementares

No primeiro experimento, os testes foram feitos com fluxos de dados, porém os mesmos resultados seriam os esperados para os testes de fluxos de Voz (VoIP) e TV (IPTV). Pois quando algum fluxo é ativado em uma ONU, a máquina de estados desse fluxo na OLT fica parada no estado ATIVO, e só altera quando a ONU ou o fluxo são desativados. Dessa maneira, não existem novas transições que poderiam ser testadas, quando algum dado está sendo transmitido pelo fluxo.

Entretanto, os testes baseados em máquinas de estados não foram realizados para esses serviços, pois na versão testada, esses serviços ainda não estavam implementados.

Posteriormente a esses experimentos, o comportamento do serviço VoIP e IPTV foram analisados quando a ONU é submetida à falhas de comunicação, nesse caso a fibra que estava conectada a ONU foi retirada.

Para o teste de VoIP, uma chamada estava em execução quando a fibra foi retirada. Ao recolocar a fibra e a ONU voltar ao seu estado anterior à falha, o telefone conectado a essa ONU voltou a apresentar tom de linha, porém a ligação anterior foi perdida.

Para o teste de IPTV, um canal de TV estava sendo transmitido e a fibra foi desconectada. A transmissão do vídeo foi interrompida até a ONU retornar ao estado anterior à falha. O vídeo voltou a ser transmitido, porém o conteúdo que foi transmitido enquanto a ONU não estava conectada foi perdido.

Os comportamentos observados no teste de VoIP e de IPTV são os esperados para redes que oferecem o serviço *triple-play.*

6.4 Considerações Finais

Nessa Seção foram descritos dois experimentos de testes baseado em injeções de falhas e testes baseados em modelos de estados aplicados na rede GPON.

O primeiro experimento é uma automação de injeção de falhas, baseado em casos de testes criados a partir de máquina de estados do *software* da OLT. As falhas são emulações de falhas físicas, que trazem a falta de comunicação entre os dispositivos da rede GPON.

A utilização da validação automática de injeção de falhas nas máquinas de estado trouxe melhorias para os processos de desenvolvimento e depuração do *software*. Sem a utilização dessa técnica a equipe de desenvolvimento desse *software* teria tido muito mais dificuldade para a reprodução dos defeitos. Além disso, com a validação automática foi mais fácil rastrear as falhas, já que dessa maneira conhece-se a exata localização da falha.

O segundo experimento também é baseado em injeções de falhas, porém o alvo é o protocolo OMCI para a validação do *software* da OLT. Nesse caso, a automação só foi utilizada para a criação dos casos de testes, que também foram baseados em modelos de estados, utilizando a ferramenta Condado. A injeção de falhas ocorreu manualmente, através da manipulação do código, quando algumas mensagens eram corrompidas.

Na próxima Seção serão destacadas as conclusões finais de todo esse trabalho, além de ressaltar todas as lições aprendidas com esses experimentos.

7 Conclusões e Trabalhos Futuros

Esse trabalho demonstrou que a realização de testes, com injeção de falhas e modelos de estados, de forma planejada e documentada, traz melhoria na qualidade do sistema. Por essas técnicas não terem sido aplicadas anteriormente ao GPON, o esforço para a implementação dessas atividades foi significativo. Principalmente no primeiro experimento que utilizou o robô de testes, foram despendidas, em média, 160 horas para ser readequado para esse experimento. Porém, todo o investimento foi recompensado através dos resultados obtidos (número de falhas encontradas) e da possibilidade de reexecução desses testes em outras etapas do desenvolvimento do sistema.

A utilização das máquinas de estado foi de grande relevância para esse trabalho, pois elas serviram de base para todos os experimentos. As máquinas de estados determinaram os locais de injeção de falhas através da criação de casos de testes baseado nas transições que essas máquinas apresentam.

A eficiência dos testes realizados nesse trabalho pode ser medida através dos resultados do primeiro e segundo experimento, comparados com os resultados dos testes de regressão relatados em FADEL *et al*, 2011. Os testes de regressão, executados anteriormente a esse experimento, resultaram em 206 defeitos, durante a execução de testes por um período de 1 ano e 6 meses (utilizando 16 versões de *software* da OLT). Enquanto que, os testes do primeiro experimento desse trabalho resultaram em 47 defeitos encontrados em uma execução que durou 5 dias. O segundo experimento foram encontrados 31 defeitos em apenas 1 dia de execução, ambos os experimentos utilizaram somente uma versão de *software* da OLT. A técnica utilizada aumentou a probabilidade de se encontrar falhas no GPON. Com a automação de algumas tarefas da atividade de testes, foi possível obter resultados mais precisos em menos tempo.

A técnica de testes utilizada no primeiro experimento aumentou a probabilidade de se encontrar falhas que são difíceis de reproduzir manualmente. As transições de um estado para o outro são de curta duração (milissegundos) e devido ao curto intervalo de transição o teste manual é impraticável, causando uma cobertura inadequada do sistema. Para a equipe de testes

ficou evidente a cobertura dos testes, uma vez que as transições de estados foram sistematicamente exercitadas, o que não era possível precisar antes da automação da execução dos testes.

Esses experimentos podem ser utilizados em qualquer aplicação de *software* embarcado que possa ser representada por máquinas de estados, e onde objetiva-se validar os mecanismos de tolerância a falhas implementadas. A reprodução desses experimentos foi simplificada devido ao padrão de documentação de testes utilizado. Essa documentação facilitou a comunicação entre as pessoas envolvidas nos experimentos e no desenvolvimento desse trabalho.

Com a utilização dos experimentos já descritos anteriormente, os objetivos desse trabalho, descritos na Seção 1.2, foram atingidos. Muitos defeitos foram observados e repassados para a equipe de desenvolvedores, para que os mecanismos de tolerância a essas falhas fossem melhorados, visando atingir o nível de tolerância a falhas esperado. Assim como as técnicas de injeção de falhas e testes baseados em modelos de estados puderem ser difundidas entre os colegas do CPqD.

7.1 Trabalhos Futuros

Uma melhoria a ser introduzida no primeiro experimento seria automação da geração dos casos de testes a partir das máquinas de estados, utilizando também a ferramenta Condado, uma vez que nesse experimento a geração dos casos de testes foi manual.

No segundo experimento foi observado que a OLT envia e recebe uma mensagem por vez, porém a única informação que ela armazena da mensagem é a identificação da mensagem. Se essa identificação e o CRC da mensagem estão corretos, a OLT executa a mensagem, independente do que seja o conteúdo, mostrando que os mecanismos de tolerância a falhas não foram corretamente implementados.

Apesar da técnica de teste não objetivar encontrar as soluções para os defeitos encontrados, uma melhoria que poderia ser implementada na OLT seria o armazenamento de

outras informações além da identificação da mensagem. Na OLT poderia ser armazenado também o tipo de mensagem e a identificação da entidade. Porém, incluir essa verificação em todas as mensagens que chegam à OLT, pode gerar um custo operacional muito alto para a OLT, que deve ser avaliado, antes de ser implementado.

Apesar de todo o esforço gasto com esses dois experimentos, muitas outras falhas podem estar ocultadas no sistema GPON. Devido a isso podem ser estudas outras técnicas de testes, para serem aplicadas a fim de obter um sistema com maior dependabilidade.

Outro estudo pode ser feito em relação às falhas que os *memory leaks* podem causar em todas as aplicações de *software* embarcadas. Os *memory leaks* são as memória alocadas no sistema e que não são mais liberadas, correndo o risco do sistema ficar sem memória. Isso gera graves problemas, pois a memória é consumida aos poucos, provocando comportamento inadequado do sistema ao longo de seu uso.

Esse futuro trabalho deve objetivar a criação de uma ferramenta de verificação dinâmica da memória alocada e não liberada, diminuindo os riscos da aplicação sofrer os danos causados pelos *memory leaks*.

Referências

ALCORN, G.; SCOTT, S.; MORROW, G.; FIGUEROA, O.; WATKINS, M.: *Risk Management Reporting*. Goddard Space Flight Center. 08 de maio de 2008.

AMBROSIO, A. M.: *CoFI: Uma abordagem combinando testes de conformidade e injeção de falhas para validação de software em aplicações espaciais*. Tese (Doutorado). São José dos Campos. INPE. (2005).

AMBROSIO, A. M.; MARTINS, E.; VIJAYKUMAR, N.L.; DE CARVALHO, S.V. *A Conformance Testing Process for Space Applications Software Services*. JACIC - Aerospace Computing, Information, and Communication. AIAA. (2006)

AMBROSIO, A.; FRANCISCO, M. F. M.; SANTIAGO JR., V.; SILVA, W.; MARTINS, E.: *Designing fault injection experiments using state-based model to a test a space software*. Third Latin-American Symposium on Dependable Computing (LADC). (2007)

ARLAT, J.; CROUZET, Y.; KARLSSON, J.; FOLKESSON, P.; FUCHS, E; LEBER, G.: *Comparison of Physical and software-implemented fault injection techniques*. IEEE Transactions on Software Engineering, v. 25, n.9, p. 1115-1133. (2003).

AVIZIENIS, A., LAPRIE, J. C., RANDELL, B.: *Fundamental Concepts of Dependability*. UCLA CSD Report n. 010028, LAAS Report n. 01-145, Newcastle University Report n. CS-TR-739 (2001).

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, Vol. 1, No. 1 (2004).

BINDER, R. *Testing object-oriented systems - models, patterns and tools***.** Boston: Addison-Wesley. 1191p. (ISBN 0-201-80938-9). (2000).

BICUDO, M. D. D.: *Sobrevivência em redes ópticas transparentes*. Dissertação (Mestrado) UFRJ. Páginas 5 e 47. (2005).

BONILLA, M. L.: *Análise Crítica de Plataformas GPON e EPON para Aplicação em Redes Ópticas de Acesso de Alta Capacidade*. Dissertação (Mestrado). UNICAMP. Página 11. (2008).
CHOI, G.S.; IYER, R.K.: *FOCUS: an experimental environment for fault sensitivity analysis.* IEEE Computer Society. Páginas: 1515 – 1526. (1992).

CLARK, J.; PRADHAN, D.: *Fault injection: a method for validating computer-system dependability*. Páginas: 47 - 56. IEEE Computer Society. (1995).

CUNHA, J.; RELA, M.; SILVA, J.: RT-Exception: Fault-injection for real-time. Coimbra: Centro de Informática e Sistemas da Universidade de Coimbra. (2000)

CRNKOVIC, I.; LARSSON M.: *Building Reliable Component-Based Software Systems*. Artech House Publishers. Chapter 10. ISBN 1-58053-327-2. (2002).

DE MILLO, R.; LI, T.; MATHUR, A.: *Architecture of TAMER: A Tool for dependability analysis of distributed fault-tolerant systems*. Purdue University. (1994).

DIJKSTRA, E.: *Classics in software engineering. Structured programming*. Yourdon Press Upper Saddle River, NJ, USA. 424p., ISBN: 0-917072-14-6 (1979).

DUSTIN, E.: *Lessons in Test Automation*. STQE – The Software Testing & Quality Engineering Magazine. (1999).

EL-FAR, I.; WHITTAKER, J.: *Model-based software testing*. In: Encyclopedia of Software Engineering. San Francisco, CA, USA: Wiley InterScience, v. 1, p. 825–837 (2002).

FADEL, A.; MORAES, R.; MARTINS, E.: *Automated validation of embedded optical network software.* LADC – Industrial Track. São José dos Campos. (2011)

FANTINATO, M.; CUNHA, A.; DIAS, S.; MIZUNO, S.; CUNHA, C.: *AutoTest – Um Framework Reutilizável para a Automação de Teste Funcional de Software*. SBC. SBQS III. Brasília. (2004).

FEWSTER, M.; GRAHAM, D.: *"Software Test Automation"*. Addison-Wesley. (1999).

HANDLER, J.: *Automated Testing of Embedded Software*. Disponível em: https://www.stickyminds.com/sitewide.asp?ObjectId=2049&Function=DETAILBROWSE&ObjectType=ART&sqry=*Z(SM)*J(ART)*R(createdate)*&sidx=1148&sopp=25&sitewide.asp?sid=1&sqry=*Z(SM)*J(ART)*R(createdate)*&sidx=1148&sopp=25. Acessado em 10/2010.

HSUEH, M. C.; TSAI, T.; IYER, R.: *Fault injection techniques and tools*. Páginas 75-82. IEEE Computer Society. (1997)

HOPCROFT, J.; ULLMAN, D.; MOTWANI, R.: *Introdução à Teoria de Autômatos, Linguagens e Computação*. Editora Campos.Tradução da 2ª Edição. Rio de Janeiro. 576p. ISBN: 85-352-1072-5. (2002)
IEEE 610.12-1990 (*Institute of Electrical and Electronic Engineer) - Standard Glossary of Software Engineering Terminology*. (1990).

ITU-T G.984.1: *Gigabit-capable passive optical networks (GPON): General characteristics* (03/2008). Disponível: <http://www.itu.int/rec/T-REC-G.984.1-200803-I/en>. Acesso em junho/2011.

ITU-T G.984.2: *Gigabit-capable passive optical networks (GPON): Physical Media Dependent (PMD) layer specification.* Disponível: <http://www.itu.int/rec/T-REC-G.984.2/en>. Acesso em setembro/2011.

ITU-T G.984.3: *Gigabit-capable passive optical networks (GPON): Transmission convergence layer specification*. Disponível: <http://www.itu.int/rec/T-REC-G.984.3/en>. Acesso em setembro/2011.

ITU-T G.984.4: *Gigabit-capable passive optical networks (G-PON): ONU management and control interface specification* (02/2008). Disponível: http://www.itu.int/rec/T-REC-G.984.4/en. Acesso em junho/2011.

ITU-T G.984.5: *Gigabit-capable passive optical networks (G-PON): Enhancement band*. Disponível: http://www.itu.int/rec/T-REC-G.984.4/en. Acesso em setembro/2011.

KANER, C.; BACH, J.; PETTICHORD, B.: *Lessons learned in software testing: A context drivem approach*. John Wiley & Sons. ISBN: 0471081124. (2002).

KARLSSON, J.; GUNNEFLO, U.; LIDTN, P.; TORIN, J.: *Two fault injections techniques for test of fault handling mechanisms*. Test Conference, Proceedings, International. (1991).

KNIGHT, J.C: *Safety critical systems: challenges and direction.* Software Engineering, 2002. ICSE 2002. P. 547-550. (2002)

LAPRIE, J.C.: *Dependability – Its Attributes, Impairments and Means*. In Predictability Dependable Computing Systems, B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood, Editors, Springer, Berlim, Alemanha, pp. 3-18. (1995)

LEITE, J.; LOQUES, O.: *Introdução à Tolerância a Falhas*. II *SCTF, cap. 4 do minicurso,* Campinas, SP, Brasil. (1987)

LEE, E. A.: *Embedded Software.* UCB ERL Memorandum M01/26, University of California. Berkeley/CA. July. (2001)

MARTINS, E.: *ATIFS: um Ambiente de Testes baseado em Injeção de Falhas por Software* – Relatório Técnico DCC-95-24 – UNICAMP, dez 1995. (1995)

MARTINS, E.; SABIÃO, S. B.; AMBRÓSIO, A. M.: *ConData: a Tool for Automating Specification-based Test Case Generation for Communication Systems.* Proc. 33rd. Hawaii International Conference on System Sciences (HICSS'33), Maui, USA. (2000).

MENEZES, P.: *Linguagens Formais e Autômatos*. 4ª Edição. Nº03. Série livros didáticos. Instituto de Informática da UFRGS. Editora Sagra Luzatto. ISBN: 85-241-0554-2. (2002)

MORAES, R.; WAESELYNCK, H.: *Documenting Robustness Testing Experiments using Extended UML Profiles.* Technical Report LAAS-CNRS, n. 11434. (2011).

MYERS, G. J.: *The Art of Software Testing*, 2º edition, John Wiley & Sons, Nova Jérsei, 256p., ISBN: 0-471-46912-2. (2004).

NETO, A.; SUBRAMANYAN, R.; VIEIRA, M.; TRAVASSOS, G.: *A survey on model-based testing approaches: a systematic review*. In: international workshop on Empirical assessment of software engineering languages and technologies. New York, NY, USA: ACM Press, p. 31–36 (2007).

REGGIANI, A.: *Tendências em Redes Ópticas de acesso e Tecnologia GPON – CPqD*. Disponível em: <http://www.cpqd.com.br/file.upload/p-3_cpqd-giga_atilio-e-regiane_14-05-08.pdf>. Acessado em: agosto de 2011.

RELIASOFT: *Examining Risk Priority Numbers in FMEA*. Disponível em: <http://www.reliasoft.com/newsletter/2q2003/rpns.htm>. Acessado em: janeiro de 2012.

ReSIST: Resilience-Building Technologies: State of Knowledge. Project Resist. Deliverable D12. Setembro de 2006. LAAS-CNRS. Disponível em: http://www.resist-noe.org/. Acessado em: junho de 2011.

ROSA, A. C. A.: *Uma arquitetura reflexiva para injetar falhas em aplicações orientadas a objetos*. Dissertação (Mestrado). UNICAMP. Capítulo 2. (1998).

SANCHEZ, W. P.; *PON: Redes ópticas de Acesso de baixo custo*. Tutorial Teleco. (2004).

SMITH, S.: *Business Class Services over a GPON Network*. Optical Fiber Communication Conference OFC, 2006 and the 2006 National Fiber Optic Engineers Conference. (2006).

SOMMERVILLE, I.: *Engenharia de Software*. 6ª Edição. Pearson – Addison Wesley. ISBN: 85-88639-07-6. (2005).

STOTT, D.; RIES, G.; HSUEH, M.; IYER, R.: *Dependability Analysis of a High speed network using software-implemented fault injection and simulated fault injection*. IEEE Transactions on Computers. [s1], v.47, n.1, p.108-119 (1998).

TAURION, C.: *Software Embarcado, a nova onda da informática.* 204p. ISBN: 8574522287. Rio de Janeiro: Brasport. (2005).

TELECO: *Serviços Banda Larga: O uso de Rede Óptica Passiva GPON*. Publicado em: 25/ago/2008. Disponível em: <http://www.teleco.com.br/tutoriais/tutorialblgpon/default.asp>. Acessado em: out/2011.

THOMAS, D.; HUNT, A.: *State Machines*, IEEE Software, v.19 n.6, p.10-12 (2002).

UTTING, M.; LEGEARD, B.: *Practical Model-based testing. A tool approach.* ISBN 978-0-12-372501-1, Morgan-Kaufmann, 433p. (2007).

VIERA, M.; MADEIRA, H.: *Definition of Faultloads Based on Operator Faults for DMBS Recovery Benchmarking*. Pacific Rim International Symposium on Dependable Computing. IEEE. (2002).

VOAS, J; MCGRAW, G.: *Software Fault Injections: Inoculating programs against errors*. ISBN: 0-471-18381-4. Wiley Computer publishing. 353p. (1998).

WEBER, T.: *Fault Tolerance Lecture Notes* - Disponível em <http://www.inf.ufrgs.br/~taisy/disciplinas/textos/ConceitosDependabilidade.PDF>. Acessado em: setembro de 2011.

Anexo A - Trabalho Publicado - Fifth Latin American Symposium on Dependable Computing - LADC 2011 - Industrial Track

Automated validation of embedded optical network software

Aline Cristine Fadel[1], Regina Moraes[1], Eliane Martins[2]

Industrial Track - Latin-American Symposium on Dependable Computing
[1]School of Technology – UNICAMP – R. Paschoal Marmo, 1888 – CEP: 13484-332 – Limeira – SP – Brazil
[2] Institute of Computing – UNICAMP – Av. Albert Einstein, 1251 – Campinas – SP – Brazil
alinecfadel@gmail.com, regina@ft.unicamp.br, eliane@ic.unicamp.br

Abstract. This work discusses automated tests performed on an optical network for high-capacity triple-play services (voice, video and TV). Given that high availability and reliability are important requirements for this kind of network system, it is mandatory to apply regression testing every time it is updated. For automatic execution of the regression testing a test robot was developed. It is responsible for collect the outputs and compare them with the expected results. In order to complement the regression testing, fault injection campaigns were performed, which were based on state machines of the class of embedded software, seeking a more comprehensive coverage of tests. For this purpose the robot was adapted to send a trigger to the algorithm that controls the fault injection. The fault injection used an optical switch that interrupted the communication among the board's system components. The results show the effectiveness of fault injection in detecting bugs that were not detected during several months when other types of tests were applied.

Keywords: embedded software validation, fault injection, regression test, GPON

I. INTRODUTION

Telecommunications systems must operate without interruption and without loss of data, since these events may cause several financial losses to telecommunications operators and users.

According to Sommerville [1], reliance on computer systems is a property that reflects the degree of confidence that users may on the system and it depends on reliability and availability of telecommunication system. Moreover, according to Clark and Pradhan [2], availability is the ability to be operational at any given time without failures, while reliability is the ability to operate without failure for a certain period. However reliability cannot be expressed numerically; other terms can be used such as: "unreliable", "very reliable" and "ultra reliable" [1].

Disregarding reliability and availability during software development can result in the occurrence of several failures in the operational phase of the system. Software test can be applied to help in disclosing the faults responsible for these failures.

Software testing is time consuming, therefore the automation of software test is designed to conduct a large volume of test cases (TC) in a shorter time, and with greater reliability in results, also reduce costs in the testing process. Despite the challenge in dealing with embedded software is the use of tools to automate the tests, whereas each type of embedded software has a specific interface. Each embedded software requires a customization for automation to be successful.

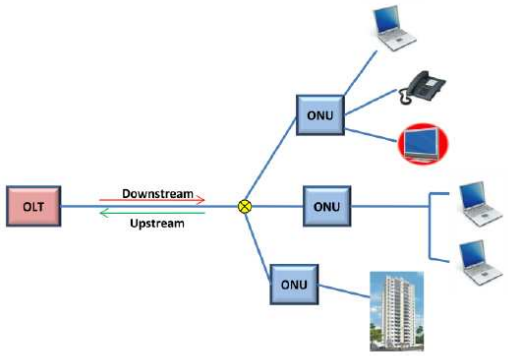

Figure 1. Operation of GPON network [4].

The optical networks are gaining new followers every day, mainly in European and Asian continents [3]. The increase of the number of users occurs due to high rates of transmission of these networks, which can reach 2.5 Gbit/s in the downstream direction and 1.25 Gbit/s in the upstream direction besides the possibility of their diversity of services [4].

Aiming to ensure the reliability and availability of embedded software for optical networks, two experiments will be discussed in this paper. The first one is the regression testing automation and the second one is the automatic fault injection, both applied in GPON (Gigabit Passive Optical Networking), a project of CPqD (Centre for Research and Development in Telecommunications).

GPON networks are access solutions for high-capacity triple-play services (voice, video and TV) using data transmission over fiber [3]. The connections are established between the OLT (Optical Line Termination), located at the service provider, and the ONU (Optical Network Unit), located at residential or corporate areas, as shown in Figure 1.

GPON consists of software units that are embedded on OLT and ONU's boards. Each OLT fiber must be connected to almost 128 ONU's and at a distance of up to 12 miles (The standard ITU-T sets 37 miles) [4].

At CPqD, a GPON pilot project has been placed on trial in conjunction with the Experimental Design High-Speed Network - GIGA, also developed at CPqD [3], and has been in operation for at least six months. This pilot also includes mobility network of Wimax, WI-Fi and Ad Hoc. In this project, the GPON network proved to be an efficient mode of transmission, and was able to carry data over long distances with high transmission capacity and quality [5].

Due to the large number of features of GPON system, a technique of regression testing was used. The regression tests are often used when some correction is made in the

software or when some functionality is inserted. It is a very popular technique, since all features can be reviewed in new versions. It is currently the best technique to be applied for this purpose. According to Rothermel [6], regression testing is a technique performed on a modified program to ensure that changes are correct, and without damage of unchanged portions of the program. In this experiment, the regression testing were automated and executed in the GPON system, ensuring reliable results in a shorter execution time. In order to succeed in it a customization was necessary and a robot was developed to perform the tests.

A complementary test technique is also presented, that is an experiment validation that automates the fault injection in an embedded software system of a telecommunication project. Fault injection is applied based on state machines that describe the transition among the possible system states, which are used to define TC's in order to achieve greater coverage of the system. The greater the amount of coverage of the code exercised, the greater the quality of software. However, increasing the code coverage results in improving software quality (less failures), while improving the fault coverage implies improving the mechanism for fault tolerance [7].

This later experiment has used finite state machines, which consist of a single set of states. They have an initial state, and one or more final states, depending on their execution flow. The state transitions occur when an event is generated, and when this happens, the states are updated [9].

The bugs found by these experiments are classified according to the following criteria: (i) the low priority bugs are trivial flaws or improvements to be made, which do not affect the operation of the system; (ii) the average critical bugs are defects that can affect network performance, but do not lead to crash or interruption of its operation; (iii) the highly critical bugs can interrupt the system or compromise network performance; (iv) very highly critical bugs can totally undermines the functioning of the system or some basic functionality.

The test automation experiments have brought significant results, requiring great effort to run only at the beginning of each implementation for the generation of TC's and test the adequacy of the robot. Later, this effort was rewarded by the possibility of running the tests on each new version, only updating TC's.

After this introduction, Section II discusses the regression tests executed manually and automatically through experiment executed in the GPON network. Section III comprehends the fault injection experiments and its automation. And Section IV discusses the advantages achieved in the implementation of the automation of regression testing and automated fault injection and the conclusions.

II. REGRESSION TESTING

The GPON project development has been taking four years, and at least once a month a new functionality is released. Due to the extensibility of its functionality, it is necessary to check each new version in order to verify if new errors were not inserted in areas of the system that were previously tested.

The development of TC's in GPON was created from a document of commands, and they are based on submitted inputs, and the outputs (black-box testing). The commands entries are sent to the software, and each command has its own respective parameters. The results of these commands are displayed to the operator. An example could be sending as an input command to enable the ONU, and with the output, the activation and deployment of this device will be expected.

The features of the GPON project can be represented by commands. Currently, the project has 110 commands with more than 850 TC's.

TC's are created based on their parameters and their preconditions. For instance, the "activate_link" command has two parameters, device_id and link_id, which can vary from 0 to 7. Based on these conditions, as shown in Table 1, TC's "activate_link" (al) command may be designed: within these limits, outside the limits, or error conditions, among other cases. These conditions were held for the creation of TC's of all available commands.

TABLE 1
Preparation of Test Cases

Test Type	Test
Within the limits (success)	al 0 0, al 7 7, al 1 5, ...
Outside the limits (error)	al -1 -1, al 8 8, al 8 4, ...
Errors test	Non-execution of pre-conditions, re-executing the same command, ...

A. MANUAL REGRESSION TESTING

In the execution of manual tests, the commands are sent by the operator to the OLT software through an interface, as shown in Figure 2. This interface displays the output, also events and alarms generated by the application can also be seen. All TC's stored in spreadsheets had: the preconditions, the test itself, the expected result and the priority of the test run.

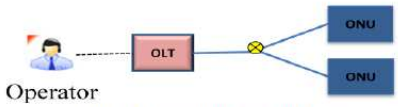

Figure 2. Execution of manual tests

A defect is observed when the result of TC is not equal to the expected output.

However, when regression testing is performed manually they are usually very repetitive and their conduction requires considerable effort. Due to the volume of TC's developed, the execution has become unfeasible. It required the development of tools that automate it, as discussed in the next topic.

B. REGRESSION TESTING AUTOMATION

Automatic execution of regression testing allows the reduction of the execution time of the TC's, and may lead to increased coverage of the software. The increase of coverage may occur once that it allows the testers to conduct a large number of tests, to focus their efforts on other types of test or tests that cannot be automated.

The test robot was developed at CPqD to perform regression testing. It was implemented in C language, Linux operating system and it communicates with the OLT equipment through TCP/IP sockets. The robot is responsible for collecting the outputs and comparing them with the expected results.

TC's that were manually developed were stored in a database consisting of text files. These files have the test script to be executed, and the expected results. Only one of the commands cannot be automated, due to the hardware characteristics.

After the execution of TC's, the robot generates a report composed by these executed TC's and the execution statistics, such as TC's that were successful and the present errors. This report can be sent to the entire team automatically.

C. RESULT OF REGRESSION TESTING AUTOMATION

During the one year of automatic execution of regression tests in GPON, we found 206 failures (distributed during the time), as shown in Figure 3. In this chart, one can see that the period of deployment of the robot (June-October, 2009) a large number of bugs were found. Another period when the number of bugs increased was the one between March and June 2010, which was the time when critical new features were implemented. In the Figure 4, the reported bugs were separated by criticality.

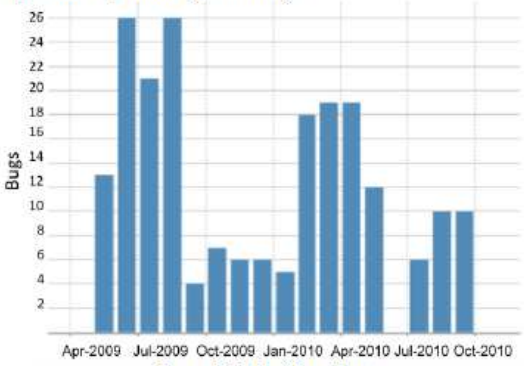

Figure 3. Total of found bugs

The execution time of regression testing has been reduced from 4 days to 10 hours. This resulted in time saving once the tests are executed overnight. During business hours it is only necessary to check the generated report, the registration bugs and updating TC's. With the time saving it was possible to think of our features for the robot, allowing the creation of another experiment that will be addressed in Section III.

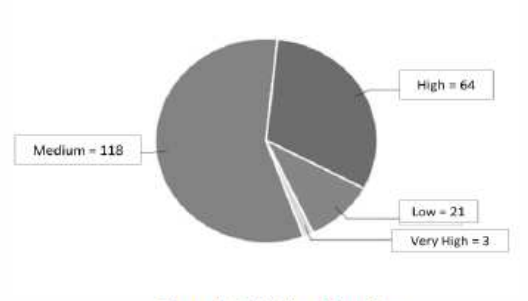

Figure 4. Criticality of found bugs

III. FAULT INJECTION

Telecommunications systems must have high availability, they must be able to provide, even in adverse conditions, to provide the requested services. Thus, one way to validate these systems is to verify if they are fault tolerant, i.e., they must deliver the service correctly even in the presence of faults [8]. Fault-tolerance is one of the essential characteristics for systems that need to ensure high dependability.

By developing systems that require high dependability, just implementing fault-tolerance mechanisms is not enough. It is also equally important to validate them in order to ensure they are correctly implemented, i.e., that all the services offered by the system are provided according to their specifications. To validate the implementations one can use any means designed to achieve dependability, such as: prevention, tolerance, removal and fault forecasting [8].

A technique that can be used to check whether the system is fault-tolerant or not is the application of fault injection, which aims to observe the behavior of the system in the presence of faults that were deliberately included in order to validate the system under analysis.

A. MANUAL FAULT INJECTION

The tests performed attempted to emulate the interruption of communication between the OLT and the ONU's. In an operational system, these failures may occur due to breakage of optical fibers or loss of signal, for instance. In an ideal system, when such events occur, after the fibers are replaced or the signal is recovered, the components must be reconnected. The OLT and the ONU's connected to this OLT should return to their previous state without human interaction.

To conduct this test manually, the fiber connected to the ONU's and the OLT was removed and the behavior of the system was observed. However there were two main difficulties: when a failure was discovered; it was difficult to reproduce it since the exact moment of the interruption is unknown. The second difficulty is to ensure that all these moments are being covered by tests.

To minimize these difficulties and increase test coverage, it was decided to perform the fault injection based on state machines of classes for embedded software, which will be detailed in the next subsection.

B. AUTOMATIC FAULT INJECTION

The experiment is detailed in Figure 5, it is composed by OLT and ONU's, a Test Robot containing OXC (Optical Cross Connection) equipment and controller software, and a switch. The switch performs the connections between network devices.

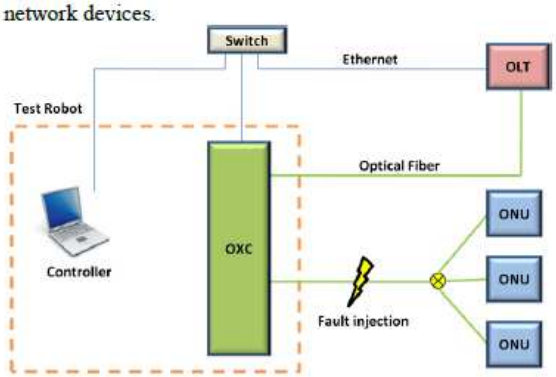

Figure 5. Operation of the experimental tests

The OXC was inserted into the environment to control the communication between the ONU's and the OLT. It is composed by optical switches, it is able to connect and disconnect the fibers by commands, and it was used to inject faults in the test environment.

The robot used in regression testing has been adapted to run this experiment. In this case it is responsible for:

- sending commands to the OLT;
- receiving the system logs (the logs have communication information between the OLT and the ONU's);
- sending commands to the OXC and;
- Running the TC's and analyzing their results.

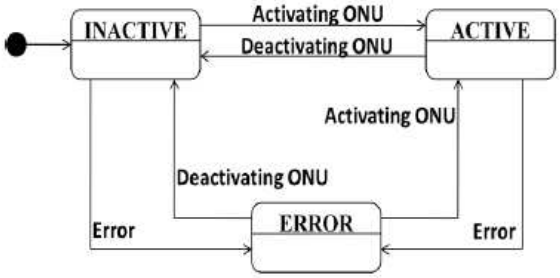

Figure 6. Example of ONU state machine

The TC's were derived from the OLT's state machine, therefore it is in the state transition that a greater likelihood of unexpected events normally takes place. In Figure 6, there is an example of the ONU state machine. Specific commands are required to enable or disable an ONU. In case of failure during activation or deactivation of the ONU, its status can be changed to Error. In the case the failure is tolerated, the ONU returns to the state prior to the failure event. Another instance of a state machine is presented in Figure 7, due to confidentiality reasons; the state machine was modified in this paper.

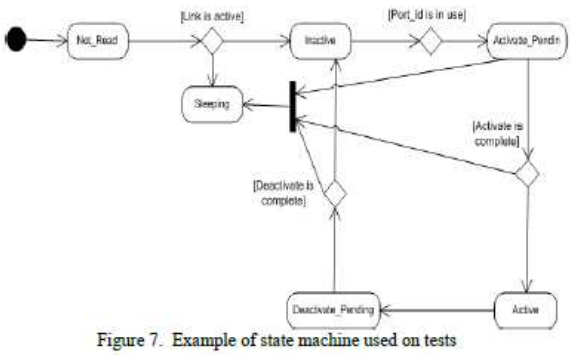

Figure 7. Example of state machine used on tests

Automated tests will run from the receipt of logs by an instance of the tests robot. As shown in the sequence diagram in Figure 8, the robot sends commands to an instance of GPON. When a specific state log is received, the instance of the robot will send a command to the instance of the OXC to interrupt the communication between the OLT and the ONU, injecting the fault in the system. After a period of time, the communication is re-established, and the logs are analyzed in order to verify the behavior of the OLT and the ONU's. While the robot does not receive these logs, the system continues processing the normal execution flow.

The period that the application was inoperative, i.e., with no critical event, since test results obtained were the same if interruption duration is 2 seconds or 2 minutes.

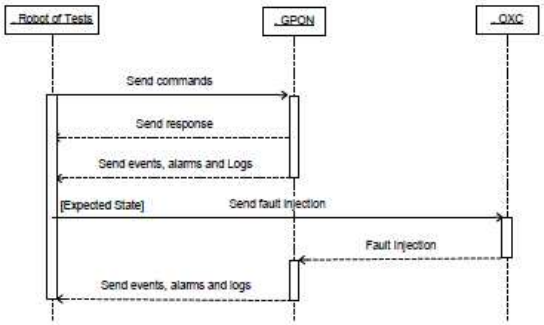

Figure 8. Sequence Experiment Diagram

We used nine state machines with from 3 to 24 transitions states, as shown in Table 3. In this way, some tests were performed with the interoperation of the states, when two or more state machines operate concurrently by threads, being processed in a seemingly simultaneous way, as shown in Figure 9.

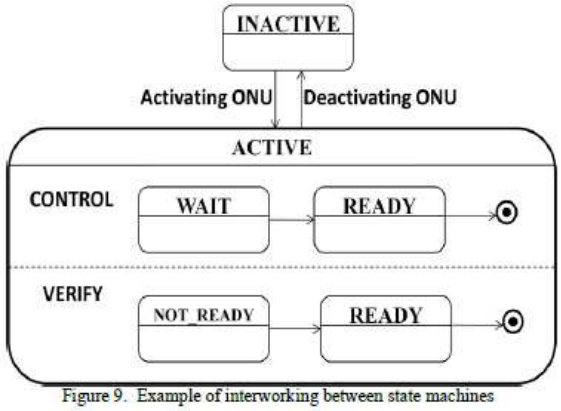

Figure 9. Example of interworking between state machines

In Figure 9, the ACTIVE state of Figure 6 is detailed. This state is composed by other state machines as CONTROL and VERIFY. The execution of these "sub-state machines" occurs concurrently until all state machines reach the final state. When the CONTROL and VERIFY states reach their final stages, the controls of the two sub-states competitors come together again in a single stream, and the state is updated to ACTIVE.

The model states represent the possible behaviors of the system, and the test scenarios derived from them. For the tests execution, 95 TC's that comprehended all the transitions of states machines listed. In each execution of TC's, the scenario was changed, the number of ONU's connected and the number of these flows connected to ONU's could be altered, as shown in Table 2.

TABLE 2
Scenario Explored on Tests

Scenario Testing	Situation
Scenario 1	1 ONU connected without flow
Scenario 2	3 ONU's connected without flow
Scenario 3	1 ONU connected with 1 flow
Scenario 4	2 ONU's connected with 1 flow
Scenario 5	1 ONU connected with 5 flows
Scenario 6	2 ONU's connected with 5 flows

C. RESULTS OF THE AUTOMATIC FAULTS INJECTION

After assembling the scenario shown in Figure 5, the three TC's created for the experiment were applied in different scenarios (Table 2), and the results are shown in Table 3. The first column of Table 3 shows the state machines. The number of transitions of each state machine is presented at the second column. The number of bugs found in medium criticality is shown in the third column and the number of bugs with high criticality is in the fourth column. The last column shows the total number of bugs found regardless of their criticality. Most of the problems found were bugs of medium criticality. However, high criticality bugs were also found, which could have stopped the system or compromised its normal performance, making the system work partially, if the faults had been inactivated.

TABLE 3
RESULTS OBTAINED WITH EXPERIMENT

State Machine	Number of transitions	Medium criticality bug	High criticality bug	Total bugs
A	12	2	0	2
B	4	3	0	3
C	3	0	0	0
D	22	6	2	8
E	12	11	0	11
F	4	2	0	2
G	6	0	0	0
H	8	0	0	0
I	24	0	0	0

In Table 3, the state machines identified as D and E presented more bugs than any other state machines. A possibility is that they have a higher frequency of use, and therefore are the most critical. In the state machine D, highly critical were bugs found, leading to system crashes.

It is noteworthy that the system has been operating in a pilot project for six months, and prior to this experiment, white box tests were also carried out in the code of implementing each state machine, in order to check if the algorithm was consistent with the proposed diagrams during the development project. Moreover, during previous phases of testing during the development, manual fault injection tests were performed, in which the fibers were simply removed and replaced in the equipment at randomly.

All bugs found in the fault injection experimental tests reported in this paper had not been previously observed. These faults in a commercial system in operational phases put at risk the reliability and availability of the system and must be corrected. Besides the system reliability, the organization credibility is also at stake as the system is critical for the client.

IV. CONCLUSIONS

With the aim of achieving greater reliability and availability of systems, the use of automation in testing optical networks is essential. Test automation can improve the coverage of tests, reduce redundant manual test execution, maximize the accuracy of test results and increase repeatability.

The results of the experiments provided from automation of regression test revealed a large number of bugs. Moreover, automation of regression testing saves time that allowed the test team to think of making new types of tests. This resulted in the automation of fault injection testing to be included in the process.

The use of automation of fault injection validation in the state machines brought improvements to the development and debugging process of the organization. Without using these techniques developers would have much more work trying to reproduce manually failures found by testers. Besides, the preparation of TC's from state machines allowed a better coverage once all states of the OLT.

The joint use of the techniques to automate tests and to inject the faults allows validating the operation of state machines in the presence of unexpected situations, improving the quality of the tests performed.

The errors reported by the tests that used automatic fault injection technique could not be identified with the manual techniques, once this technique has a wider breadth of coverage of state machines when compared to previously use the techniques. After the failures detection, their removal was facilitated by their knowledge of the exact location of the bug, and the exact time the crash occurred. This information is a precious one for the development team and greatly facilitated the system traceability.

This technique increases the probability of finding faults that are difficult to reproduce manually. The transitions from one state to another are short-lived (few milliseconds) and due to the short transition interval the manual test is impracticable, causing an inadequate coverage of the system.

The bugs found had not been reproduced manually and are more critical than those who had already been found by manual testing.

For the team responsible for the GPON system, the use of this technique was very suitable as failures could be reproduced as they occur in the field. Before the use of this technique it was not known if the system was able to treat them. The results of these experimental tests showed that the system must still be improvement to meet the needs of systems with high dependability. Thus, one of the next steps is to inject faults in the communication among components, in order to adjust the system so that it does not behave in unexpected ways, making it tolerant to such failures.

The fault injection experiment can be applied in any software application that can be represented by a state machine, and where communication can be interrupted.

Also, in future work, the objective is to observe the behavior of the GPON system when faults are inserted in the communication protocol called OMCI (ONU Management and Control Interface), which controls communication between the OLT and ONU's.

ACKNOWLEDGEMENTS

This research was conducted with the support of the Graduate Program of FT/UNICAMP - School of Technology and the Centre for Research and Development in Telecommunications (CPqD).

REFERENCES

[1] Sommerville, I.: Software Engineering. Sixth Edition. Pearson Addison Wesley. (2003).

[2] Clark, J., Pradhan, D.: Fault Injection: A Method for validating computer–system dependability. IEEE. Computer Society Press, Los Alamitos, CA, USA (1995).

[3] Tendências em Redes Ópticas de acesso e Tecnologia GPON – "Trends in Optical Networks and Technology Access GPON" – CPqD. Available at: <http://www.cpqd.com.br/file.upload/p-3_cpqd-giga_atilio-e-regiane_14-05-08.pdf>. Last accessed on August 1, 2010.

[4] ITU-T G.984.1: Gigabit-capable passive optical networks (GPON): General characteristics (03/2008). Available at: <http://www.itu.int/rec/T-REC-G.984.1-200803-I/en>. Last accessed on August 1, 2010.

[5] Martins, L.; Pozzuto, J.; Mokarzel, M.; Giolo, F.; Bonon, E.; Freire, M.; Junqueira, I.: Fornecimento de Acesso em Banda Larga com Solução Híbrida GPON, WiMAX, WiFi-Adhoc e Mesh CPQD – "Provision of Broadband Access Solution with Hybrid GPON, WiMAX, WiFi-Mesh and Adhoc CPQD". Infobrasil. (2010).

[6] Rothermel, G., Harrold, M.: Framework for evaluating regression test selection techniques. Proc. of the 16th Int'l. Conference on Software Engineering, Sorrento, Italy, p. 201-210. (1994).

[7] DeMillo, R., Li, T., Mathur, A.: Architecture of TAMER: A Tool for Dependability Analysis of Distributed Fault-Tolerant Systems (1994).

[8] Avizienis, A., Laprie, J. C., Randell, B.: Fundamental Concepts of Dependability. UCLA CSD Report n. 010028, LAAS Report n. 01-145, Newcastle University Report n. CS-TR-739 (2001).

[9] Thomas, D.; Hunt, A.: State Machines, IEEE Software, v.19 n.6, p.10-12 (2002).