MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-13264-TDI/1031

COFI: UMA ABORDAGEM COMBINANDO TESTE DE CONFORMIDADE E INJEÇÃO DE FALHAS PARA VALIDAÇÃO DE SOFTWARE EM APLICAÇÕES ESPACIAIS

Ana Maria Ambrosio

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Eliane Martins, Nandamudi Lankalapalli Vijaykumar e Solon Venâncio de Carvalho, aprovada em 1º de julho de 2005.

INPE
São José dos Campos
2005

681.3.06 : 620.1.004

AMBROSIO, A. M.
CoFI: uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em aplicações espaciais / A. M. Ambrosio. – São José dos Campos: INPE, 2005.
209p. – (INPE-13264-TDI/1031).

1.Metodologia de teste. 2.Teste de conformidade. 3.Injeção de falhas. 4.Verificação de telecomando. I.Título.

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

Computação Aplicada

Dr. Antonio Montes Filho

Presidente / INPE / SJCampos - SP

Dr. Solon Venâncio de Carvalho

Orientador(a) / INPE / SJCampos - SP

Dra. Eliane Martins

Orientador(a) / UNICAMP / Campinas - SP

Dr. Nandamudi Lankalapalli Vijaykumar

Orientador(a) / INPE / SJCampos - SP

Dr. Nilson Sant'Anna

Membro da Banca / INPE / SJCampos - SP

Dr. Clóvis Torres Fernandes

Convidado(a) / ITA / SJCampos - SP

Dr. Mario Jino

Convidado(a) / UNICAMP / Campinas - SP

Aluno (a): Ana Maria Ambrosio

São José dos Campos, 01 de julho de 2005

*“*Melhor é acender um fósforo que lamentar a escuridão*”*.

Drausio Varela.

à CÂNDIDA ALBANEZ e DIONINO AMBROSIO, meus pais,
ao MÁRIO CELSO PADOVAN DE ALMEIDA e
à LUANA, minha filha, dedico este trabalho.

AGRADECIMENTOS

Este trabalho não teria sido possível sem a orientação, o apoio e amizade da Prof. Dra. ELIANE MARTINS, a quem gostaria de expressar aqui minha gratidão.

Agradeço o orientador Prof. Dr. Solon Venâncio de Carvalho, que ao seu modo me passou o espírito de um pesquisador. Meus agradecimentos ao Prof. Dr. N. L. Vijaykumar por todos os incentivos. E a todos os professores do LAC que me possibilitaram crescimento como pesquisadora meus agradecimentos.

Às colegas de curso Adriana Mattedi e Silvia Maria F. S. Massruhá pelas palavras amigas e positivas nas minhas horas de fraqueza.

Em especial, agradeço à grande amiga Maria de Fátima Mattiello-Francisco pelas discussões técnicas, pela visão pragmática que sempre me passou, por todos os conselhos e principalmente pelo positivismo que me tem transmitido desde que nos conhecemos.

Ao Mario, à Luana e à minha mãe pelo afeto e por compreenderem (ou simplesmente aceitarem) minha ausência em tantas ocasiões.

Aos colegas da DSS que compreenderam o período de silêncio pelo qual passei.

Ao Instituto Nacional de Pesquisas Espaciais – INPE/Engenharia e Tecnologia Espaciais pela oportunidade de desenvolver este trabalho de doutorado e pela confiança no retorno que ele poderá proporcionar às missões especiais brasileiras.

RESUMO

Este trabalho propõe um processo e uma metodologia de teste de software visando reduzir custos e diminuir o número de incidentes em missões espaciais. Seguindo a tendência atual das padronizações de software em aplicações espaciais, o teste de conformidade será de interesse em tais aplicações em futuro próximo como foi o caso da padronização de protocolos em telecomunicações, que levou ao estabelecimento do padrão ISO-9646 de teste de conformidade, nos anos 80. Para softwares em aplicações espaciais a validação da conformidade é complementada com a técnica de injeção de falhas para considerar as conseqüências da radiação do ambiente sofridas pelo software. A proposta consta de um processo de teste baseado na IS-9646 que abriga uma metodologia para orientar o projeto de testes. Ambos, o processo e a metodologia são denominados CoFI, pois combinam teste de conformidade e validação por falhas injetáveis. A principal característica da metodologia é tratar falhas que imitam os problemas físicos causados pela radiação que afetam a comunicação computacional a bordo de satélites. Uma outra característica é o uso de métodos formais e artefatos da UML para facilitar a automação. Efeitos-colaterais, como explosão no número de testes, do uso de ferramentas de geração de testes a partir de especificações formais são tratados com a separação do comportamento normal e excepcional em modelos distintos. A seqüência de teste gerada com a CoFI é avaliada empiricamente em três aplicações: um serviço padronizado na norma ECSS-E-70-41A que aborda comunicação solo-bordo, um protocolo de bordo e um protocolo de solo para comunicação entre Centro de Controle e estação terrena. As duas últimas têm sido usadas atualmente em missões de satélites científicos no INPE. Uma análise da eficiência da seqüência de teste CoFI relacionada ao critério de adequação de falhas de mutação em máquinas de estado foi realizada. Os resultados mostraram que a metodologia é competitiva e eficaz para ser aplicada em projetos espaciais reais. Apesar da abordagem CoFI focar a validação de software de comunicação a bordo de satélites, ela não é restrita a esse tipo de aplicação.

COFI: AN APPROACH COMBINING CONFORMANCE TEST AND FAULT INJECTION FOR ESPACE APPLICATION SOFTWARE VALIDATION

ABSTRACT

In order to reduce costs and decrease the number of space mission incidents this dissertation proposes a software test process and methodology. Following the standardization trend of software in space application, conformance testing will be of interest in such applications in the near future; as was the case for telecommunications when, in 1980`s, the International Organization for Standardization (ISO) established the ISO conformance testing standard IS-9646. To complement the conformance validation and to take into account the consequences space applications software suffers from the environment radiation, the fault injection technique was adopted. In this dissertation, the test process is based on the IS-9646 and it embraces a test methodology that guides test designing. Both the process and the test methodology are named CoFI, as they combine conformance testing and fault injection. The main characteristic of the methodology is to deal with faults that mimic problems caused by the space radiation environment that affects computational communication on board of satellites. Another feature is to use formal methods and UML artifacts in order to allow automation. Side effects such as the test number explosion of using formal methods are mitigated with normal and exceptional behavior representation into distinct models. The CoFI test sequence is empirically evaluated in three real applications: a standardized service of the ECSS-E-70-41A for ground-board communication, an on-board communication protocol and a ground, Control Centre and ground station, protocol. The last two applications have been used in INPE`s scientific satellite missions. An analysis of the CoFI test sequence efficiency related to the adequacy criteria of finite state machine mutation faults was performed. Results have shown a competitive and efficient methodology for real space projects. Although the CoFI approach is focused on satellite on-board communication software validation, it is not restricted to this kind of application.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AF - *Active Ferry*

ANSI - *American National Standards Institute*

APEX - *Alpha, Próton and Electron Monitoring Experiment in the Magnetosphere*

APID - *Application Process IDentification*

ASP - *Abstract Service Primitive*

ATM - *Asynchronous Transfer Mode*

ATIFS - Ambiente de Teste com Injeção de Falhas por Software

CCSDS - *Consultative Committee for Space Data System*

CLCW - *Command Link Control Word*

CLTU - *Command Link Transfer Unit*

CNES - *Centre Nationalle D'Etude Spatialle*

CoFI - Abordagem de teste que emprega Conformidade e Falhas Injetáveis

DCI - Declaração de Conformidade da Implementação

DFT - Documento de descrição das Ferramentas de Teste

ECSS - *European Committee for Space Standardization*

ESA - *European Space Agency*

ETSI - *European Telecommunications Standards Institute*

EXP - Experimento científico inteligente

FIC - *Fault Injector Controller*

FIM - *Fault Injector Module*

FSM - *Finite State Machine*

IEEE - *Institute of Electrical and Electronics Engineers*

INPE - Instituto Nacional de Pesquisas Espaciais

ISO - *International Organization for Standardization*

ITU - *International Telecommunication Union*

IXIT - Informação eXtra da Implementação em Teste

LT - *Lower Tester*

MME - Modelador de Máquinas de Estados

MSC - *Message Sequence Chart*

NASA - *National Aeronautics and Space Administration*

OBDH - *On-Board Data Handling*

OCL - *Object Constraint Language*

OMG - *Object Management Group*

PCO - Ponto de Controle e Observação

PDU - *Protocol Data Unit*

PF - *Passive Ferry*

PLAVIS - *Platform for software Validation and Integration for Space systems*

PLUTO - *Procedure Language for User in Test and Operations*

PLUTO - *Product Lines Use case Test Optimization*

PUS - *Packet Utilization Service*

SAMSTAG - *SDL and MSC based test case generation*

SAT - Seqüência Abstrata de Teste

SATS - Seqüência Abstrata de Testes Selecionados

SCENT - *SCENario-based validation and Test of software*

SDL - *Specification Description Language*

SPET - Seqüência Parametrizada e Executável de Teste

SWIFI - *SoftWare Implemented Fault Injection*

TC - Telecomando

TM - Telemetria

TOTEM - *Testing Object-orienTed systEms with the unified Modeling language*

TTCN - *Testing and Test Control Notation*

UML - *Unified Modeling Language*

UT - *Upper Tester*

UUT - *Unit Under Test*

CAPÍTULO 1

INTRODUÇÃO

Aplicações espaciais dependem cada vez mais de sistemas computacionais para aumentar a autonomia de operações em satélite, balões, espaçonaves e foguetes. Esses sistemas requerem softwares dedicados, sejam embarcados na eletrônica de vôo dos veículos espaciais, sejam em solo contribuindo para a operação remota dos mesmos. Devido a sua natureza específica, i.e, diferente para cada nova missão, o custo de desenvolvimento de software para aplicações espaciais é alto (Larson e Werts, 1992, cap 15 e 16 ; Mattiello-Francisco, 2003a).

1.1 Tendência no Desenvolvimento de Aplicações Espaciais

Cientistas e técnicos responsáveis pelo desenvolvimento de sistemas espaciais acreditam que o custo de tais sistemas poderia ser reduzido se houvesse maior padronização. Nesse sentido, dois órgãos compostos por representantes de agências espaciais vêm trabalhando na padronização de sistemas dedicados a aplicações espaciais, que são, o *Consultative Committee for Space Data System* (CCSDS) desde 1985 (CCSDS, 2004) e o *European Committee for Space Standardization* (ECSS), desde 1993 (ECSS , 2003a).

Entretanto, o uso de padrões apropriados ao desenvolvimento de software em aplicações espaciais deve estar associado a processos, metodologias e técnicas de validação dos produtos de software para reduzir riscos de acidentes. A TABELA 1.1 (Mazza, 2000), apresenta os últimos incidentes em sistemas espaciais e aponta insuficiência na validação de software como uma das principais causas. Dadas as suas particularidades, a qualidade do software espacial depende muito dos testes realizados e da compatibilidade deles com os requisitos especificados (Levenson, 2001).

Com o estabelecimento recente de serviços de sistemas computacionais para aplicações espaciais em normas publicamente reconhecidas, a necessidade da realização de teste de

conformidade, com os quais busca-se validar a conformidade de implementações com relação à especificação normatizada, é uma conseqüência.

TABELA 1.1 - Falhas em Sistemas Espaciais.

PROJETO	INCIDENTE	CAUSA
ARIANE 501, junho 1996	Falha no Lançamento, perda de 4 satélites CLUSTER	*Insuficiência de Validação de software e Validação de sistema*
Mars Climate Orbiter set. 1998	Perda do *Orbiter*	Troca de unidades de medida. *Insuficiência de validação de software e validação de sistema*
Mars P. Lander dez.1998	Perda do *Lander*	Erro de software forçou desligamento prematuro da máquina de pouso: *insuficiência de validação de software*
Titan-4B abril 1999	Falha ao colocar o satélite USAF Militar na sua órbita final	Falha na ignição do estágio superior do Centaur. Erro no ponto decimal no sistema de guiagem. *Insuficiência de validação apropriada no software.*
Delta-3, abril 1999	Lançamento abortado	Falha do SW de bordo na inicialização da máquina principal. Insuficiência de validação apropriada de software
PAS-9 março 2000	Perda do satélite ICO *Global Communication* F-1	Erro na atualização do software de solo. Insuficiência de validação de software

FONTE: Mazza (2000).

1.2 Teste de Conformidade

Na década de 80, a *International Organization for Standardization* (ISO), responsável por padronizações em telecomunicações, após padronizar protocolos de comunicação, propôs um padrão para orientar a realização de *teste de conformidade* de implementações do protocolo (realizadas por diferentes empresas) que seguiam as especificações dos protocolos do padrão ISO. Este padrão de teste de conformidade de protocolos ISO encontra-se no conjunto de normas IS 9646 (ISO, 1991), aprovado também pela International Telecommunication Union (ITU-T, 1995). Os testes de

conformidade permitem não somente revelar erros que estão presentes no escopo do sistema, como também, demonstrar que o sistema em teste implementa todas as capacidades requisitadas (Binder, 2000, p 718).

Dentre as vantagens dos testes de conformidade pode-se citar que ele: (i) determina se uma implementação incorpora os requisitos da norma, (ii) provê métricas para julgar o progresso na direção da norma, (iii) permite reuso dos testes e dos procedimentos, (iv) facilita a integração de produtos de diferentes fabricantes desenvolvidos em épocas diferentes, pois, com o passar dos anos novas tecnologias são incorporadas e; finalmente (v) possibilita redução dos riscos na integração de novas tecnologias (Chestnutwood, 2004).

O teste de conformidade, na academia, tem motivado pesquisas em geração automática de casos de teste a partir de especificações em notação formal baseada em estados principalmente para validação de protocolos de comunicação (Chow, 1978 ; Bochmann e Petrenko, 1994 ; Cavali et al., 1996 ; Petrenko et al., 1996a ; Martins et al., 1999; entre outros). A vantagem da automação dessa atividade de teste é reduzir os custos com testes, que chegam a atingir até 50% do orçamento destinado ao desenvolvimento de um produto de software (Tretmans e Belinfante , 1999; Blackburn et al., 2004). Um panorama dos resultados das pesquisas em geração automática de testes pode ser encontrado em Ural (1992), Lee e Yannakakis (1996), Dssouli et al. (1999), Lai, (2002). Os trabalhos de Jagadeesan et al. (1997) e Lai (2002) constatam, entretanto, que os excelentes resultados com a tecnologia de geração automática de teste, obtidos pela academia, ainda não são amplamente adotados pela indústria. E os dois fatores mais evidentes dessa situação são: (i) a dificuldade em se obter a especificação formal de sistemas reais, (ii) a proliferação de eventos e ações a serem mapeadas e exercitadas no modelo.

No contexto em que: (i) as pesquisas indicam vantagens na automação de geração de testes e fraquezas no uso prático desta automação; (ii) os investimentos em teste de software e de sistemas espaciais são baixos (Mazza, 2000) mas; (iii) a padronização de serviços de software em aplicações espaciais é crescente, propõe-se nesta tese, uma

adaptação do processo de teste de conformidade da ISO e uma nova *metodologia de teste* aplicados a sistemas de comunicação em aplicações espaciais. A adaptação do processo de teste de conformidade ISO (ISO, 1991) vem da necessidade de incorporar atividades e artefatos para acomodar o uso da técnica de injeção de falhas. Uma vez que esta técnica completa a validação de sistemas espaciais que são executados a bordo de satélites, por permitir acelerar a ocorrência de falhas físicas que imitam as falhas provocadas pela radiação espacial (advindas do ambiente em que o software final será executado). A metodologia provê um guia para a criação sistemática de testes de conformidade e de experimentos de injeção de falhas

Os testes de conformidade aplicados a sistemas espaciais não seriam suficientes para assegurar a qualidade requerida, já que sistemas computacionais em aplicações a bordo de satélites sofrem influências da radiação[1] do ambiente espacial, que pode provocar falhas nos sistemas de comunicação, na memória e no processamento. Adicionalmente, os testes devem tentar reproduzir fielmente (tanto quanto possível) o ambiente final de operação: "*voe o que testar, teste o que voar*" (Levenson, 2001). Maior investimento na validação é imprescindível para esses sistemas que colocam milhões de dólares em risco se não funcionarem bem (Hagar, 1996 ; Ihara, 2004).

Na proposição da metodologia de teste, considerou-se a tendência atual no desenvolvimento de software em aplicações espaciais, na qual o software tem sido cada vez mais desenvolvido por indústrias, fora das agências espaciais. Conseqüentemente, a metodologia deve ser viável economicamente. A metodologia proposta orienta a pessoa responsável pelos testes a criar uma especificação formal do comportamento do serviço a ser testado, a partir da descrição informal, em passos que acrescentam formalismos, gradativamente.

[1] A região de mais intensa radiação no espaço é chamada cinturão de Van Allen, consta de uma zona de partículas altamente carregadas, localizada em altas altitudes no campo magnético da Terra. James A. Van Allen foi quem as descobriu em 1958. (Enciclopédia Britânica)

1.3 Validação por Injeção de Falhas

Uma técnica que tem sido bastante utilizada para avaliar e validar o comportamento de sistema, em presença de falhas, é a injeção de falhas. As falhas podem ser injetadas por radiação de íons pesados ao hardware, provocando falhas transientes em posições aleatórias nos circuitos integrados dos computadores. O artigo de Karlsson et al. (1994) e o trabalho da agência espacial francesa em Pignol (2004) ilustram o uso dessa técnica.

A injeção de falhas pode, também, ser realizada 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). A SWIFI foi explorada pela Agência Espacial Européia, no início da década de 90, para melhorar a prática de prevenção de falhas de software. Nesse projeto os mecanismos de tolerância a falhas do software de controle de órbita e atitude, foram avaliados com experimentos de injeção de falhas em posições aleatórias na região do código da memória do computador de bordo (Vardanega et al., 1995). Recentemente, a SWIFI tem sido usada para validar a tolerância a falhas requerida no uso de softwares comerciais em aplicações espaciais. Ngo e Harris (2001) e Madeira et al. (2002), relatam experiências com SWIFI na avaliação de sistemas comerciais (processadores Powerpc 750/LynxOS), para uso no processamento de dados científicos do projeto *Remote Exploration Experimentation* (REE) da NASA. Os resultados mostraram que técnicas adicionais ainda são requeridas nos sistemas comerciais, para uso em aplicações espaciais.

No INPE, investigações com injeção de falhas por software para validação de sistemas de comunicação em aplicações espaciais, vêm sendo realizadas no projeto de pesquisa denominado ATIFS (INPE, 2002 ; Martins et al., 2002 ; Martins e Mattiello-Francisco, 2003b ; Ambrosio et al., 2004).

Devido às vantagens da técnica de injeção de falhas por software para validação do comportamento de sistemas computacionais em aplicações espaciais, ela foi adotada

como um complemento aos testes de conformidade na abordagem Conformidade com Injeção de Falhas (CoFI), proposta neste trabalho.

A abordagem CoFI inclui uma metodologia para geração de testes e um processo de teste de conformidade, que agregam a técnica de injeção de falhas para validação de software em aplicações espaciais. O processo de teste acomoda o uso das ferramentas de testes desenvolvidas no projeto ATIFS, de forma a reduzir a distância entre o estado da arte e o estado da prática em teste de software (Lai, 2002), uma vez que a simples existência de novas ferramentas não é suficiente para a melhoria da qualidade do software (Hagar, 1996).

O processo de teste define um conjunto de atividades e os artefatos necessários para realização dos testes. A metodologia orienta a realização da atividade de criação de casos de teste prevista no processo.

1.4 Motivação para Metodologia de Teste

Baseadas nos fundamentos matemáticos dos autômatos finitos (ver Capítulo 2), encontram-se na literatura, vários algoritmos e ferramentas de geração automática de teste. Entretanto, o trabalho intelectual de elaboração de modelos do comportamento de um sistema em uma linguagem formal, mesmo que em sua forma gráfica, ainda está sob a responsabilidade do testador que se depara com dificuldades (Hagar, 1996 ; Merri et al., 1996 ; Jagadeesan et al., 1997 ; Lai, 2002) e questões do tipo:

- como obter a especificação completa do comportamento do sistema em teste?
- o que é pertinente e o que não é pertinente modelar para a geração dos testes?
- uma vez conseguido o modelo, como evitar a explosão do número de casos de teste? A solução para o problema de explosão combinatória tanto em número de estados do modelo do comportamento, quanto o número de casos de teste gerados automaticamente é uma questão aberta (Ural, 1992).

Adicionalmente, na validação de softwares em aplicações espaciais a atenção aos problemas físicos causados pelo ambiente externo é imprescindível. A adoção da técnica de injeção de falhas para cobrir este aspecto na validação, impõe outra questão:

- como derivar experimentos de injeção de falhas determinísticos, que validem o comportamento dos mecanismos de tratamento das falhas causadas pelo ambiente?

A metodologia CoFI propõe uma resposta às questões colocadas atendendo à necessidade de projeto de casos de testes daqueles serviços (ou funções), que descrevem um comportamento baseado em estados, isto é, reconhecem eventos e os tratam a qualquer momento e, ainda devem continuar funcionando mesmo quando eventos não previstos acontecem. Uma particularidade dessa metodologia, é que ela incorpora passos para geração de casos de teste de conformidade e passos para derivação de casos de falha (os quais vão apoiar experimentos de injeção de falhas).

A metodologia CoFI orienta o projeto de casos de teste e de casos de falhas de forma sistemática, obedecendo a um *critério* que permite quantificar e projetar os testes com antecedência no ciclo de vida do desenvolvimento do software. Um *critério de teste* permite quantificar os testes e aumentar a qualidade do conjunto de casos de teste. Além disso, aplicando-se a metodologia de teste CoFI, reduz-se a dependência da experiência e percepção da pessoa responsável pelos testes na qualidade do conjunto de casos de teste, sem impedir a valiosa contribuição intelectual do ser humano. A metodologia proposta desfruta dos resultados positivos de geração automática de teste, baseada em métodos formais: casos de teste são gerados a partir de modelos de estados, cuja base matemática são os autômatos.

1.5 Contribuições Desta Tese

A principal contribuição desta tese é a definição de uma metodologia de teste de software, que sistematicamente orienta a pessoa responsável pelos testes a projetar casos de teste e experimentos de injeção de falhas. Essa metodologia é incorporada a um

processo de teste de conformidade também proposto nesta tese. O desenvolvimento de software em aplicações espaciais sob a responsabilidade do INPE, carece de uma metodologia e um processo para apoio as atividades de teste de tais softwares. Na metodologia a geração de teste parte de uma especificação textual (descrição informal) e, de forma gradativa e a um custo praticável, leva à criação de um modelo formal (modelo de comportamento baseado em estado). Ela apoia-se no uso de diagramas UML, que vêm sendo usado no ciclo de desenvolvimento de software em aplicações espaciais. A metodologia integra ferramentas de teste, tanto para geração automática de teste baseadas em autômatos como para injeção de falhas, que vêm sendo exploradas em pesquisas no âmbito do projeto ATIFS (Martins, 1995 ; Stefani, 1997 ; Sabião, 1998 ; Araújo, 2000 ; Martins et al., 2003a ; Ambrosio et al., 2005a). A metodologia objetiva reutilização de testes e satisfaz as necessidades de validação de software em aplicações espaciais com o uso da técnica de injeção de falhas. Ao integrar vários artefatos UML, permite reduzir a distância entre o estado da prática e o estado da arte no que diz respeito ao uso de métodos formais em teste de software (Ambrosio, 2005a).

Ao incorporar a técnica de injeção de falhas ao teste de conformidade, a metodologia incluiu passos para a derivação determinística de experimentos de injeção de falhas por software (SWIFI) (Ambrosio, 2005b). Essa parte da metodologia, se considerada separadamente, caracteriza uma abordagem pragmática e nova no âmbito das pesquisas em torno da técnica de injeção de falhas.

A adaptação do *Processo de Teste de Conformidade* da **IS 9646** (ISO, 1991) para software em aplicações espaciais, na qual foram incorporados requisitos necessários para acomodar a técnica de injeção de falhas, como um complemento ao teste de conformidade, não encontra similar.

Com relação à problemática de teste de software em aplicações especiais, cuja carência de processos e métodos foi discutida, este trabalho contribui não apenas na proposição da abordagem CoFI, mas também na exemplificação da geração de testes e conseqüentemente, na modelagem dos serviços da Norma ECSS-E-70-41A (2003).

Esses serviços, recém-estabelecidos, especificam a comunicação entre aplicações de software em solo e a bordo (o *Packet Utilization Service* (PUS)) de uma espaçonave e vêm sendo amplamente adotados nas missões espaciais européias. Assim, o conjunto de testes CoFI e a modelagem desses serviços constituem contribuições à comunidade da área espacial (Ambrosio, 2004a).

Este trabalho está organizado nos capítulos:

- *CAPÍTULO 2 - CONTEXTO DO TRABALHO*: apresenta conceitos de validação e teste de software.

- *CAPÍTULO 3 - CoFI$_p$: UM PROCESSO DE TESTE* DE CONFORMIDADE INCLUINDO VALIDAÇÃO DE COMPORTAMENTO PERANTE FALHAS: apresenta o processo de teste proposto.

- *CAPÍTULO 4 - CoFI$_m$: UMA METODOLOGIA PARA GERAÇÃO DE TESTES PARA VALIDAÇÃO DO COMPORTAMENTO DE SOFTWARE EM CONFORMIDADE E EM PRESENÇA DE FALHAS*: descreve detalhadamente os passos da metodologia para geração de casos de teste e de casos de falhas, ilustrando-os com o protocolo OBDH-EXP.

- *CAPÍTULO 5 - AVALIAÇÕES EMPÍRICAS DA SEQÜÊNCIA DE TESTE CoFI:* ilustra a avaliação da metodologia em três estudos de caso reais de software em aplicações espaciais. No primeiro, a metodologia foi aplicada ao protocolo OBDH-EXP, que define comunicação entre duas aplicações a bordo, desenvolvido e implementado no INPE. No segundo, a metodologia baseou-se na descrição textual de um serviço de comunicação solo-bordo, da norma ECSS-E-70-41A. No terceiro, a CoFI foi aplicada a um protocolo de comunicação solo-solo, cuja especificação já se encontrava em um autômato. As seqüências de testes foram avaliadas quanto à adequação a falhas estruturais em máquinas de estados finitas. Uma comparação com a metodologia N+ também foi realizada para um modelo reduzido do primeiro estudo de caso.

- *CAPÍTULO 6 - TRABALHOS CORRELATOS:* esse capítulo apresenta os principais trabalhos relacionados com o conteúdo desta tese, que abordam metodologias voltadas para teste de protocolos, metodologias de teste em engenharia de software e aquelas que tratam validação por injeção de falhas.

- *CAPÍTULO 7 - CONCLUSÃO*: resume os resultados obtidos, discute as contribuições e analisa as perspectivas de trabalhos futuros.

- *APÊNDICES:*

    - Apêndice A - apresenta uma visão geral sobre teste de conformidade segundo a Norma IS-9646.

    - Apêndice B - mostra os modelos da especificação OBDH-EXP com apenas dois comandos, criados para comparação com a seqüência N+.

    - Apêndice C - apresenta os resultados dos passos da $CoFI_m$ criados a partir da especificação do serviço *Verificação de TC* da norma ECSS-E-7041A.

- *ANEXO A* - traz a especificação do protocolo solo-solo.

CAPÍTULO 2

CONTEXTO DO TRABALHO

Este capítulo apresenta alguns conceitos que fundamentam este trabalho. Dois assuntos são principalmente focados: teste a partir de modelos em máquina de estados finita e técnica de injeção de falhas.

Primeiramente são discutidos os conceitos de verificação e validação. Aparentemente estes termos são muito semelhantes. Sommerville (2001) cita que Boehm em 1979 caracterizou a diferenças entre eles ao expressar as questões:

> Validação: nós estamos construindo o produto certo?
>
> Verificação: nós estamos construindo de maneira certa o produto?

Nas definições mais recentes apresentadas no contexto do projeto de *Capability Maturity Model Integration* (CMMI, 2002), foram acrescentadas às perguntas de Boehm, as seguintes perguntas:

> Validação: o produto satisfaz o uso pretendido quando colocado em seu ambiente pretendido?
>
> Verificação: o produto satisfaz os requisitos especificados?

Neste texto usa-se sempre o termo *validação,* pois a abordagem CoFI visa projetar testes que avaliem o comportamento do software, em sua versão final quando executado em seu ambiente operacional final. Algumas vezes o conceito de verificação poderia ser mais adequado, principalmente quando se apresenta a definição de teste de conformidade, que avalia se o software satisfaz seus requisitos especificados.

Uma forma de saber se o software certo foi construído e se ele satisfaz o uso pretendido, é testá-lo. *Testar* é executar um programa com a intenção de encontrar erros (Myers, 1979). Assim, o teste tem sucesso se encontra erros. Entretanto, encontrar erros não é a única questão relacionada a teste de software, porque os testes têm que ser factíveis

dentro de um prazo e um orçamento. Além disso, os testes servem para verificar se ele responde conforme o requisitado. O teste é uma técnica dinâmica de verificação e validação.

Antes de executar os testes é preciso projetar as entradas que exercitarão o programa (os *casos de teste*). Na literatura definem-se dois tipos clássicos de teste: o *teste caixa-branca*, que se baseia na estrutura do código para o projeto (ou geração) de casos de teste e o *teste caixa-preta*, que não se baseia no código, mas sim em outros modelos, ou somente nas funções especificadas, que definem o software. O conjunto de testes submetido a uma aplicação é chamado *seqüência de casos de teste*. Existe uma grande variedade de métodos que orientam o projeto de uma seqüência de casos de teste. Esses métodos requerem um *modelo de teste* (notação), a partir do qual derivam-se os casos de teste. O modelo de teste influencia o tipo de erro que se pretende descobrir. Esses modelos podem ser:

- a implementação, isto é, o código do programa (testes caixa-branca);
- a especificação, uma descrição das funções, um esquema do comportamento do programa (testes caixa-preta);
- as falhas de hardware ou de software.

Nos testes baseados na especificação, os modelos podem ser uma especificação textual ou um modelo formal, aquele que apresenta uma descrição matemática. Exemplos de modelos formais são as máquinas de estados finitas, as redes de Petri, a lógica temporal, entre outros. O teste de conformidade checa se uma implementação está conforme sua especificação, é, portanto, considerado do tipo caixa-preta.

Os testes baseados em falhas de hardware são, geralmente, realizados pela técnica de injeção de falhas. Os baseados em falhas de software são inspirados no conceito de mutação, podendo esta ser no código fonte (estática) ou no código executável (dinâmica).

Independentemente do tipo de teste, testar requer a realização de uma série de atividades. Se as atividades de teste estiverem parcialmente ordenadas, inter-relacionadas, estabelecidas com métodos e práticas em um curso que transforma artefatos de entrada em artefatos de saída, esse curso é dito ser um Processo de Teste. Um processo é uma receita a ser seguida por um projeto (Paula-Filho, 2001). Um método de projetar testes é um conjunto de passos que se baseia na ação sistemática e não na improvisação; aqui usamos o termo metodologia de teste ao invés de método de teste, já que a literatura não é precisa quanto a esses termos.

Nas seções seguintes, primeiramente são descritos os testes baseados em modelos de estado, conceitos formais de autômatos e métodos de geração de teste a partir de modelos formais, que compreendem a maior parte desta seção. Mais resumidamente são apresentados os conceitos relacionados com a técnica de injeção de falhas. Apesar da tese não desenvolver extensões formais que façam uso desses conceitos, ao contrário, a metodologia proporciona um grau de abstração mais alto, esses conceitos fazem parte da base fundamental que justifica esta pesquisa.

2.1 Teste Baseado em Modelos de Estados

O comportamento de um sistema computacional tem sido representado, na literatura, através de diferentes modelos baseados em estados, sejam visuais, sejam puramente matemáticos, sejam via linguagens. Entre esses modelos podemos citar autômatos, cadeias de Markov, Redes de Petri, linguagens em lógica temporal, entre outros.

Os modelos de estados mais comumente usados para representar o comportamento de um sistema com objetivo de derivar teste são as máquinas finitas de estados de Mealy.

A máquina de estados é um modelo conveniente para geração de teste, pois representa um comportamento previsível; representa características testáveis (em termos de entradas e saídas a cada momento da execução de um sistema); permite abstrair detalhes, que tornariam o teste proibitivo; preserva detalhes essenciais para revelar falhas e demonstrar conformidade; representa todos os eventos aos quais o sistema em

teste deve responder e qual a resposta esperada. Para que possam ser usadas como fonte para geração de casos de testes elas devem estar corretas, consistentes e representar de forma precisa os requisitos do sistema que será testado (Binder, 2000). Muitos algoritmos para automação da geração de casos de teste, baseiam-se em métodos de varredura de grafos e requerem certas propriedades da máquina.

Uma máquina de estados finita (FSM, do inglês, *finite state machine*) é formalmente definida como um autômato dado pela 5-tupla $M = (\Sigma, S, \delta, s_0, F)$ (Hopkroft e Ullman, 1979 ; Menezes, 2001) onde:

- Σ é o alfabeto de símbolos de entrada (ou simplesmente, entradas)
- S é o conjunto finito de possíveis estados,
- δ é a função de transição δ: $S \times \Sigma \rightarrow S$

  δ significa que para um dado símbolo de entrada e o estado corrente da máquina, esta função define o próximo estado reconhecendo ou não a seqüência dos símbolos de entrada,
- s_0 é o estado inicial, $s_0 \in S$,
- F é o conjunto de estados finais, $F \subset S$.

Na concepção de Mealy, as transições da FSM sempre compreendem entradas e saídas (Lee e Yannakakis, 1996). Do ponto de vista matemático a máquina de estados inclui uma função de saída: λ: $S \times \Sigma \rightarrow O$, onde O é o conjunto de símbolos de saída. Do ponto de vista do comportamento do sistema representado, isto significa que para toda entrada, o sistema produz uma saída.

O formalismo diagramático do autômato, chamado diagrama de estados, permite uma visualização do comportamento do sistema. Neste diagrama, os estados são representados por círculos, as transições por arcos direcionados rotulados que ilustram as entradas que causam mudança de estado (ou não). Na teoria de grafos, os círculos são

vértices e os arcos arestas e assim, algoritmos de busca em grafos são usados para gerar seqüências de transições.

Uma FSM possui propriedades estruturais, que podem ser automaticamente verificadas (Dssouli et al., 1999 ; Lee e Yannakakis, 1996), tais como:

completeza – diz-se que a FSM é completamente especificada se ela trata todas a entradas em todos os estados; c.c., ela é parcialmente especificada. Para máquinas parcialmente especificadas a seqüência de teste gerada é dita ser de *conformidade fraca;* caso contrário, ela é de *conformidade forte*. Quando as transições não especificadas de uma FSM são completadas para fins de realização dos testes, diz-se que se assume uma *suposição de completeza*;

conectividade – a FSM é fortemente conexa se, para cada par de estados (s_i, s_j) existe um caminho por transições que vão de s_i a s_j. Ela é *inicialmente conectada* se a partir do estado inicial for possível atingir todos os demais estados da FSM;

minimalidade - ou irredutibilidade se, na FSM não existem quaisquer dois estados eqüivalentes. Dois estados são eqüivalentes se, para as mesmas entradas produzirem as mesmas saídas;

determinística – quando em um estado e para uma dada entrada neste estado, a FSM permite uma única transição para um próximo estado. Caso contrário ela é não-determinística.

Devido às suas características, as máquinas de Mealy são mais convenientes para geração automática de teste: as seqüências de entrada/saída geradas são usadas para validar a conformidade da especificação (resultado esperado) com o comportamento da implementação (resultado observado).

Uma *seqüência de teste*, nesse contexto, é uma seqüência de símbolos de entrada derivada da FSM. Estas entradas são submetidas à implementação correspondente para verificar a conformidade das saídas geradas pela implementação com as saídas especificadas na seqüência de teste. Relações formais de equivalência entre uma especificação e uma implementação foram trabalhadas em Tretmans (1992), mas não serão discutidas aqui.

Os métodos mais clássicos para geração de seqüências de teste a partir de FSMs são:

Método T – ou Transition Tour (Naito e Tsunoyama, 1981). Para uma dada FSM, o método T produz seqüências que incluem as transições que partem do estado inicial, atravessam todas as transições pelo menos uma vez e retornam ao estado inicial. Esse método permite detecção de erros de saída, mas não garante detecção de erros de transferência.

Método UIO – Unique input/output (Sabnani e Dahbura, 1988). Produz seqüências de entradas capazes de identificar cada estado da máquina. Não garante cobertura total dos erros, pois a seqüência de entrada leva a uma única saída na especificação correta, mas isso não é garantido para as implementações com erro.

Método DS – Seqüência de distinção (Gonnenc, 1970). Uma seqüência de entrada é uma seqüência de distinção se, aplicada a cada um dos estados produz uma seqüência de saída diferente. A desvantagem deste método é que a seqüência de distinção nem sempre pode ser encontrada para uma dada FSM.

Método W – (Chow, 1978). É um conjunto de seqüências que permite distinguir todos os estados e garante detectar os erros estruturais, sempre que a FSM for completa, mínima e fortemente conexa.

Outros métodos como Wp, UIO-v, método-E, etc, são discutidos em Lai (2002).

Os métodos requerem propriedades das máquinas finitas de estado para produzirem as seqüências de teste. Essas propriedades acabam por limitar o uso do método em muitos casos reais, seja porque o modelo não possui todas as propriedades requeridas, seja

porque ao aplicar o método, este causa explosão de estados ou explosão no número de testes. Uma alternativa para evitar as restrições impostas por estes métodos é usar métodos menos restritos que apenas cobrem a especificação ou caminhos da FSM e que combinados com outras técnicas, aumentam o poder dos testes com apoio factível no uso de automação na geração de casos de teste.

2.2 Injeção de Falhas

A técnica de teste por *injeção de falhas* consiste na inserção deliberada de falhas e erros no sistema, com o objetivo de acelerar a ocorrência das falhas para observação do comportamento do sistema. Ela é uma técnica de validação experimental e tem sido usada para:

- Previsão de Falhas: avaliação do comportamento do sistema com relação a falhas, visando obter medidas de confiança no funcionamento (*dependability assessment*).

- Eliminação de Falhas: verificação, diagnóstico e correção do sistema submetido à validação por injeção de falhas, visando detectar as falhas de projeto/implementação (Arlat et al., 1990).

As técnicas para injeção de falhas podem ser baseadas em:

- um hardware específico – capaz de provocar falhas físicas por radiação, através da emissão de íons pesados,

- simulação do sistema - um modelo de simulação do sistema alvo é usado para avaliar o comportamento de mesmo perante as falhas,

- software - falhas de hardware são emuladas por software. Essa técnica é conhecida como injeção de falhas implementadas por software, do inglês *SoftWare Implemented Fault Injection* (SWIFI) e tem sido mais intensamente utilizada.

Um dos aspectos importantes para utilização de injeção de falhas na validação é a determinação das falhas que se vão injetar. Para determiná-las, deve-se estabelecer:

- o critério de seleção: pode ser com base no fluxo de execução de um programa, com base na especificação. Exemplo de um critério de teste é: exercitar todas as condições de exceção;

- a forma de geração: estatística ou determinística. A forma determinística baseia-se na cobertura de um modelo e segue um critério de seleção;

- modo de injetar: a injeção pode ser por alteração no fluxo de controle ou por meio de perturbações nas entradas externas do sistema.

A caracterização de experimento com injeção de falhas, requer a definição das entradas normais ao sistema alvo, provocadas pelo ambiente onde o sistema é executado (*workload*), bem como, das falhas (*faultload*).

O *faultload* inclui o conjunto de falhas a serem injetadas e, as características das falhas, como localização (endereço de memória, identificação de pino, canal de comunicação, etc) e tempo (absoluto ou relativo a outro evento). Em geral o *faultload* está associado a um modelo de falhas (um modelo que representa as falhas que o ambiente onde o sistema será executado pode lhe causar).

CAPÍTULO 3

COFI$_p$: UM PROCESSO DE TESTE DE CONFORMIDADE INCLUINDO VALIDAÇÃO DE COMPORTAMENTO PERANTE FALHAS

Neste Capítulo é apresentado o processo de teste CoFI$_p$. Este processo estabelece um conjunto de atividades relacionadas ao teste de conformidade e à validação com injeção de falhas, além disso, integra uma metodologia de teste que promovesse o potencial uso de ferramentas de validação de software em aplicações espaciais, como aconselha Hagar (1996).

As seções deste Capítulo são organizadas da seguinte forma: primeiramente são discutidas abordagens em que o teste de conformidade pode ser usado; em seguida, é traçado um paralelo entre um processo de desenvolvimento de software e um processo de teste; o processo de teste de conformidade da ISO (ISO, 1991) e o processo de teste formal, recentemente publicado em Drabick (2004), são apresentados e comparados com o processo CoFI$_p$. Na seqüência, as atividades de teste descritas nas normas de desenvolvimento de software IEEE-12207 (IEEE, 1998) e ECSS-E-40 parte B (ECSS, 2003) são abalizadas. A primeira norma trata de processos de desenvolvimento de software em geral e a segunda é específica para desenvolvimento de software em aplicações espaciais. Finalmente, as atividades e os artefatos do processo CoFI$_p$ são definidos.

3.1 Abordagens de Uso de Teste de Conformidade

O teste de conformidade tem sido usado em duas abordagens:

- os testes são executados de forma independente, por terceiros; os métodos e requisitos de teste são desenvolvidos por uma organização competente como IEEE, ISO, ANSI, ou ainda por um consórcio como o ATM Forum, que controla a documentação de teste; a especificação, a partir da qual os testes são criados, é oficial e publicamente reconhecida, ou;

- os testes são executados localmente; os métodos e requisitos de teste são desenvolvidos a partir de uma especificação interna à empresa.

O processo $CoFI_p$ pode ser aplicado a qualquer uma das abordagens citadas. No primeiro caso, uma seqüência de teste é criada em um nível de abstração tal que independe de detalhes da implementação. Tal seqüência é reutilizada para o teste de várias implementações. No segundo caso, a geração de teste parte de uma especificação contendo detalhes específicos de uma aplicação particular. A seqüência atende a uma única implementação, mas ainda é útil, pois, orienta o trabalho do testador proporcionando um meio de quantificar os testes e avaliar a cobertura dos mesmos com relação à especificação.

3.2 Processo de Desenvolvimento de Software x Processo de Teste

A norma IEEE-12207 (IEEE, 1998) define um conjunto de processos relacionados com o ciclo de vida de um software. Dentre os processos fundamentais dessa norma está o *Processo de Desenvolvimento de Software*, cujas atividades, entre outras inclui: análise de requisitos, projeto arquitetural, projeto detalhado, codificação e teste de unidade. Um outro processo definido nessa norma é o *Processo de Validação*, classificado como apoio ao desenvolvimento de software (Rocha et al., 2001), que inclui atividades como: determinar se o projeto garante o esforço de validação e o grau de independência organizacional para validação; planejar e documentar a validação; preparar requisitos de teste, casos de teste, especificações de teste; conduzir os testes; validar que o produto de software satisfaz seu uso pretendido. O *Processo de Validação* não se restringe ao uso da técnica de teste. Esta pode ser substituída pelas técnicas de análise, modelamento, simulação, etc. Ambos os processos citados têm o teste como uma de suas atividades.

A norma IEEE-1008 (IEEE, 1986), vigente desde 1987, define *tarefas de teste* como: (i) *planejamento* (define as tarefas, os recursos, o cronograma); (ii) *projeto* (especifica os casos de teste); (iii) *implementação* (obtem as entradas e os recursos para execução dos testes); (iv) *execução* (executa os testes e determina os resultados); (v) *cheque para o término* (avalia a necessidade de novos testes) e (vi) avaliação (avalia o esforço do teste

planejado e realizado, relata as variações observadas). Essas tarefas não são caracterizadas como um *processo de teste*.

Mais recentemente, a Norma ECSS-E-40 *Part B* (ECSS, 2003) foi definida pela ECSS (contração de *European Cooperation for Space Standardization*), um grupo cooperativo composto pelas agências espaciais européias e indústrias relacionadas para padronização do desenvolvimento de software em aplicações espaciais. A Norma ECSS-E-40 adapta os processos definidos na IEEE-12207 à área de software espacial, mantendo o teste de software como uma atividade dentro do *Processo de Validação*.

Ao longo dos anos, as atividades de teste para verificar e validar software vêm ganhando cada vez mais importância. Hoje, o teste é apresentado como um processo, sob a ótica da engenharia de software. Drabick (2004) enfatiza que as atividades de teste requerem conhecimentos e técnicas específicas para serem executadas e têm sua própria complexidade, demonstrando a pertinência da definição e organização dessas atividades em um *processo de teste*. Este processo de teste consta das seguintes atividades: planejamento dos testes, projeto de testes, desenvolvimento de casos de teste, desenvolvimento de programas de teste, desenvolvimento de procedimentos de teste, execução de testes e atualização da documentação. A FIGURA 3.1 mostra um paralelo entre as atividades de um processo de teste e as atividades de um processo de desenvolvimento de software, requeridas ao longo do desenvolvimento.

Com relação à norma ECSS-E-40 Part B (ECSS, 2003), o processo $CoFI_p$ está contido no escopo da atividade "*Validação com relação à Especificação Técnica*", que por sua vez, faz parte do Processo de Validação.

Cabe notar que, as atividades do processo $CoFI_p$ foram definidas com base na norma IS 9646 (ISO, 1991). A essas atividades incluiu-se o apoio à validação do comportamento do software em presença de falhas. Uma visão geral do conteúdo da norma IS-9646, encontra-se no Apêndice A.

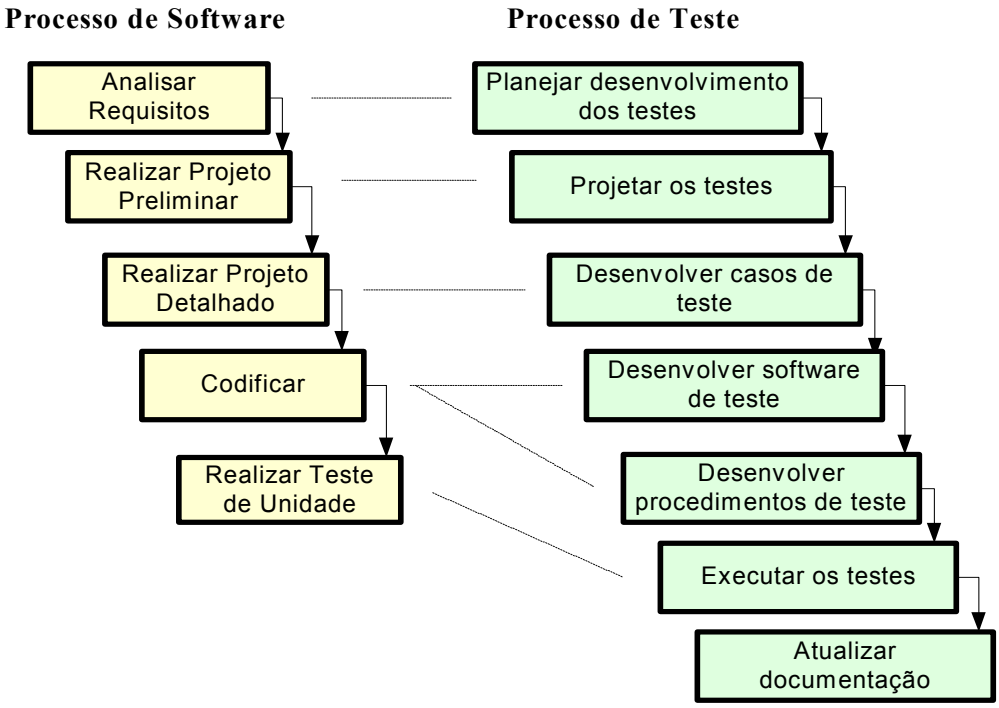


FIGURA 3.1 - Paralelo entre processo de software e processo de teste.

3.3 Fluxo das Atividades do Processo $CoFI_p$

O processo $CoFI_p$ é fortemente baseado no processo de teste de conformidade da ISO (ISO, 1991) e no processo de teste, recentemente publicado em Drabick (2004), o qual foi produzido para melhorar a qualidade de software nas grandes companhias americanas como *Lockheed Martin, Federal Aviation Authority, Amtrak*, etc.

Este processo compreende em linhas gerais: (i) geração de testes baseados na especificação; (ii) ajuste dos testes à implementação, (iii) execução dos testes (iv) avaliação da implementação quanto à conformidade com a especificação e (v) avaliação do comportamento da implementação em presença de falhas do ambiente. As atividades deste processo são apresentadas na TABELA 3.1.

TABELA 3.1 - Atividades do processo $CoFI_p$.

Atividade	Descrição
1	definir os propósitos de teste
2	definir a Seqüência Abstrata de Teste, a qual inclui casos de teste e casos de falha
3	selecionar os casos de teste e os casos de falha de acordo com a implementação a ser testada e com o injetor de falhas disponível
4	estabelecer a seqüência parametrizada executável de teste
5	executar *campanhas de teste*[2]
6	analisar os resultados dos testes
7	gerar o relatório de conformidade, robustez e estatísticas dos testes

O fluxo das atividades do processo $CoFI_p$ e os artefados envolvidos são ilustrados na FIGURA 3.2. Nessa figura, os retângulos representam as atividades, as setas indicam os artefatos de entrada e os artefatos produzidos ao final de cada atividade e, as elipses destacam os artefatos e os documentos contendo definições geradas por entidades não envolvidas com os testes, mas requeridas pela atividade. A TABELA 3.2 define os conceitos e artefatos usados no processo.

Os casos de testes e a implementação baseiam-se na mesma especificação de serviços normatizados por um órgão competente da área espacial e na Especificação Técnica particular de uma missão espacial. O produto de software, ou implementação, a ser testado é referenciado aqui como *unidade em teste*, do inglês *unit under test* (UUT). O termo é usado em inglês por ser um jargão comum da área espacial.

[2] Uma *campanha de teste* consta da atividade de executar a seqüência de teste para uma unidade em teste particular e produzir os resultados de teste (ISO, 1991).

TABELA 3.2 - Conceitos e Siglas do Processo de Teste $CoFI_p$.

Propósito de Teste	Uma descrição em prosa de um objetivo de teste bem definido, focado sobre um único requisito de conformidade (um serviço) ou em um subserviço.
Notação de Teste	A notação usada para descrever os casos de teste/falha abstratos que compõem a SAT.
Arquitetura de Teste (Ferry-Injection)	A estrutura dos componentes necessários para a execução dos testes. Ela está relacionada com a testabilidade da UUT.
Seqüência **A**bstrata de **T**este **(SAT)**	Um conjunto de casos de teste/falha abstratos. Os casos constam de uma especificação completa e independente de ações requeridas para atingir um propósito de teste específico. O caso é completo no sentido de que é suficiente para habilitar um veredicto de teste de forma não ambígua para cada saída potencialmente observável. Ele é independente no sentido de que deve ser possível executar o caso de teste/falha isoladamente dos outros. Ele é abstrato, pois independe da implementação.
Declaração de **C**onformidade da **I**mplementação **(DCI)**	Uma declaração, preparada pelo fornecedor da UUT, que contém os serviços e suas capacidades implementados na UUT.
Documento de descrição das **F**erramentas de **T**este **(DFT)**	Um documento contendo a descrição das capacidades e detalhes de interface das ferramentas de teste e/ou dos componentes do Sistema de Teste.
Informação e**x**tra da **I**mplementação em **T**este **(IXIT)**	Uma declaração, preparada pelo fornecedor da UUT, contendo toda informação sobre os serviços especificados no DCI. Essas informações trazem detalhes da implementação da UUT e do ambiente de teste necessário para executar a seqüência dos testes selecionados.
Linguagem	Uma notação de programação, para a qual existem ferramentas para gerar um código executável.
Seqüência **P**arametrizada e **E**xecutável de **T**este **(SPET)**	Casos, de teste e de falha, selecionados e parametrizados de acordo com informações contidas no IXIT.
Ferramentas de Teste	Ferramentas em software que apóiam o teste: editores, geradores de dados, simuladores, analisadores, executores, injetores, etc.
UUT	*Unit under Test* (unidade em teste). Pode ser exclusivamente em software ou incluir software e hardware.
Log de Teste	Documento que contém todos os resultados de uma execução de uma seqüência de teste.
Relatório de Conformidade	Um documento que detalha a execução e define o veredicto de cada caso teste/falha abstrato selecionado para execução.

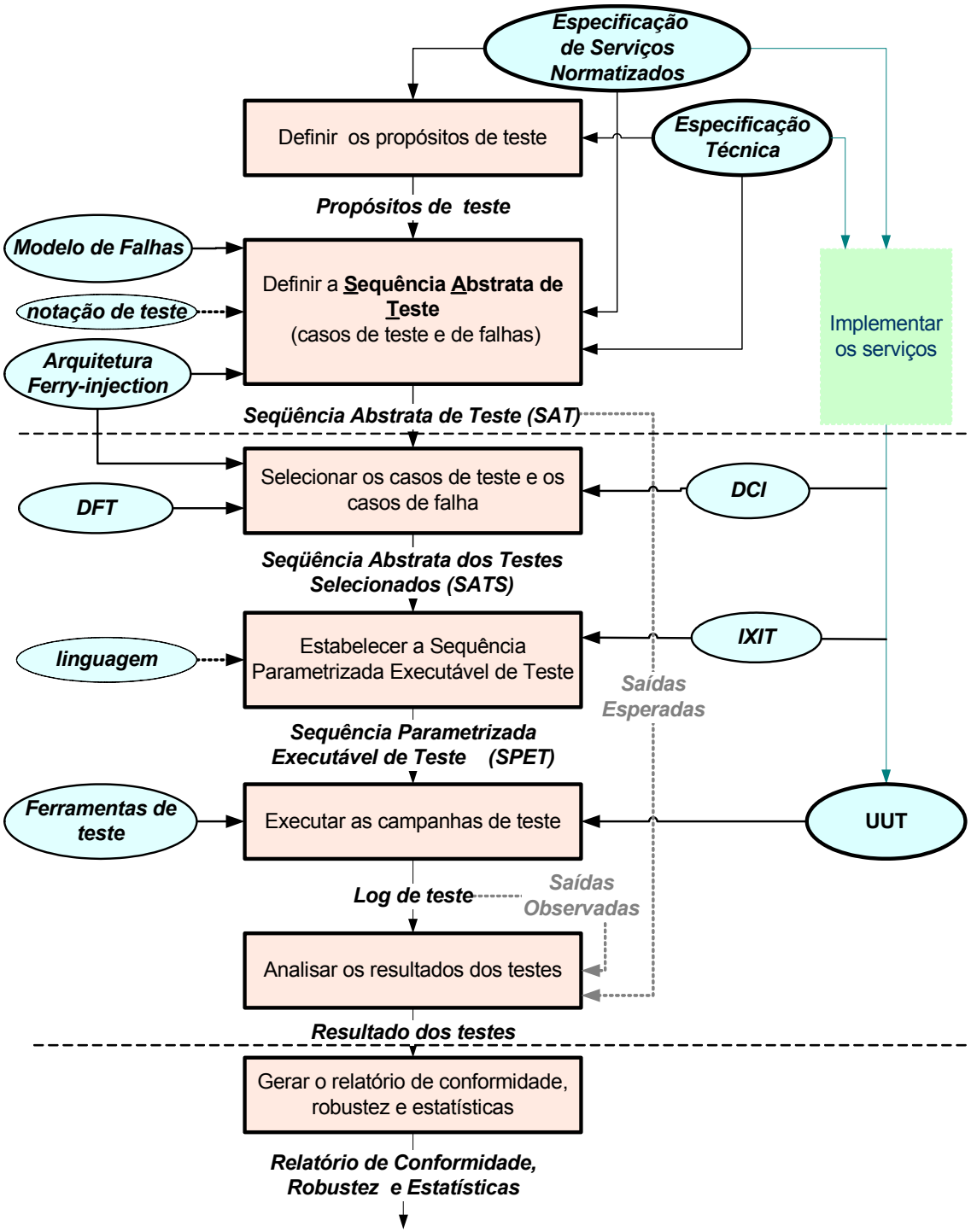


FIGURA 3.2 - Processo de teste $CoFI_p$.

3.4 Descrição das Atividades do Processo $CoFI_p$

3.4.1 Definir os Propósitos de Teste

O objetivo dessa atividade é definir os propósitos ou objetivos dos testes a serem realizados. Um *propósito de teste* consta de uma descrição precisa caracterizando um aspecto particular que deve ser testado. A definição dos propósitos de teste requer dois documentos: a norma que define os serviços normatizados que são aplicáveis à missão espacial em questão, chamada aqui de "Especificação de Serviços Normatizados" e, o documento chamado Especificação Técnica, específico de uma missão. Este último consta de um documento técnico específico da missão que descreve os requisitos do sistema a ser testado, no qual está baseado o primeiro. O artefato gerado ao final da atividade é um documento contendo os Propósitos de Teste para cada um dos serviços referenciados na Especificação Técnica. A FIGURA 3.3 mostra os artefatos de entrada e o artefato gerado nessa atividade.

FIGURA 3.3 - Atividade de Definição dos Propósitos de Teste.

Os artefatos de entrada dessa atividade são, supostamente, escritos em uma notação textual e podem estar organizados em serviços ou não. A Especificação de Serviços Normatizados é um documento público, enquanto que a Especificação Técnica é particular. Dentre as normas CCSDS, as especificações dos protocolos de telemetria e telecomando são apresentadas em notação formal, i.e, em máquinas de estados finitas desenhadas na forma de tabelas de estados (CCSDS, 2001c). Já as normas ECSS descrevem serviços em notação puramente textual. Como exemplo de uma especificação de serviços espaciais pública, citamos a norma ECSS-E-70-41A, que especifica serviços de utilização de pacotes de telemetria e telecomandos para

realização da comunicação entre aplicativos de bordo e de solo. Nas Especificações Técnicas de aplicações espaciais particulares que não estejam organizadas em serviços e que não estejam baseadas em serviços normatizados, os propósitos de teste podem ser ainda definidos com base na identificação de funções, de alto nível, a serem desempenhadas pelo sistema, devidamente apresentadas na Especificação Técnica. A especificação em serviços torna a definição dos propósitos de teste mais evidente, pois se atribui a cada serviço, pelo menos, um propósito de teste. Os requisitos de teste de conformidade podem ainda ser estáticos (realizados por comparação a partir de descrições textuais) ou dinâmicos (baseados no comportamento operacional do sistema em teste).

3.4.2 Definir a Seqüência Abstrata de Teste

Essa atividade do processo de teste consiste em projetar[3] uma *seqüência abstrata de teste* (SAT). Neste trabalho, a seqüência é composta de casos de teste e casos de falha (aqueles que cobrem o tratamento de uma falha externa à unidade em teste). A SAT é projetada de forma a cobrir todos os propósitos de teste.

Um *caso de teste* ou *caso de falha* é dito *abstrato* se consta de uma especificação independente e completa das ações necessárias para atingir um propósito específico de teste. A descrição do caso de teste/falha, também chamada de especificação de teste, é dada em um nível de abstração particular que independe da UUT, mas que leva em conta, tanto as características do ambiente como a *arquitetura de teste* (ver definições a seguir). Consta da especificação de teste a relação das *entradas* necessárias para ativar a UUT e para cada entrada a saída correspondente que deve (ou que pode) ser observada (também chamada, *saída esperada*) durante e após a execução do caso de teste ou do caso de falha. Cada caso tem saída esperada correspondente, que pode ser nula ou composta. Os casos de falha devem conter ainda uma descrição da *falha* a ser acionada.

A atividade de especificação da SAT requer:

[3] Do inglês, test design

- Definição dos Propósitos de Teste, gerados na atividade anterior.
- Especificação de Serviços Normatizados, a serem aplicados à missão.
- Especificação Técnica, específica para uma missão espacial.
- O modelo de falhas às quais a UUT será submetida quando em seu ambiente real.
- A arquitetura de teste *Ferry-injection*.
- A notação do teste – notação na qual os testes abstratos serão escritos, uma vez que, nessa atividade, os testes são independentes de uma implementação particular.

A FIGURA 3.4, ilustra os artefatos requeridos e aqueles gerados nessa atividade.

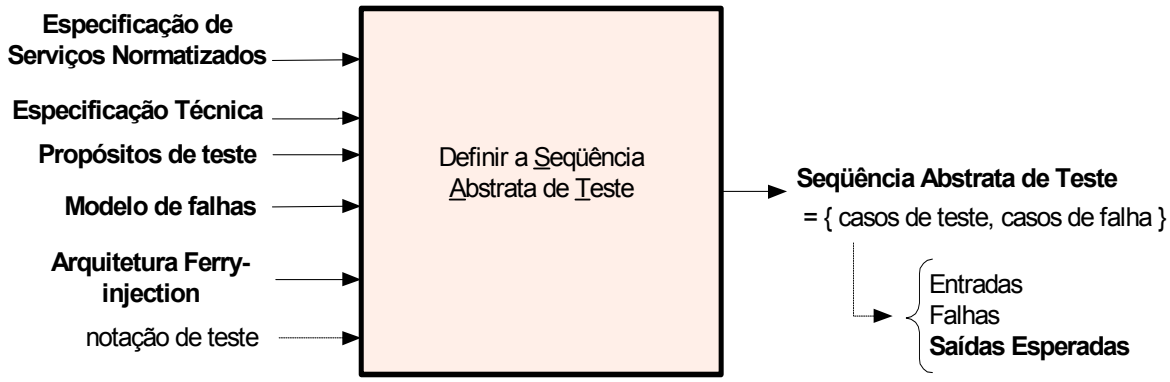


FIGURA 3.4 - Atividade de Definição da Seqüência Abstrata de Teste.

A especificação da seqüência abstrata de teste requer a definição dos seguintes conceitos:

- *Modelo de Falhas* – como citado no Capítulo 1, os sistemas espaciais estão sujeitos a radiações do ambiente espacial (Larson, 1992). Essa radiação pode causar falhas na memória, nos processadores e nos canais de comunicação dos sistemas computacionais em aplicações espaciais. O modelo de falhas

representa, no contexto estudado, um conjunto de falhas externas à UUT que é usado para projetar os casos de falha que exercitam os mecanismos de tolerância a falhas implementados na UUT. Trata-se de um modelo de falhas de hardware destinado a encontrar falhas de projeto e implementação do software. As características das falhas consideradas neste trabalho (Powell, 1991) são:

a) falhas acidentais – aquelas que ocorrem ou são criadas fortuitamente,

b) falhas físicas – aquelas causadas por um fenômeno físico adverso, em contraposição a falhas causadas por erros humanos,

c) falhas externas – as que resultam da interferência do meio físico (perturbações eletromagnéticas, radiações, temperatura, vibração, comandos errôneos, etc.) e não de partes do próprio sistema,

d) falhas operacionais – ocorrem durante a exploração (o uso) do sistema,

e) quanto à persistência, podem ser falhas permanentes (cuja ocorrência não está relacionada a uma condição pontual) ou falhas temporárias (aquelas relacionadas a uma condição pontual que acontece por um período de tempo limitado).

- *Arquitetura Ferry-injection* – a arquitetura de teste consta dos meios que caracterizam o ambiente no qual os testes são executados. A arquitetura *Ferry-injection* (Araújo, 2000) compreende componentes para ativar as *entradas,* para observar as *saídas* e injetores para acionar as *falhas*. A *Ferry-injection*, ilustrada na FIGURA 3.5, é baseada no Sistema de Teste proposto por Chanson et al. (1989), o qual é totalmente independente do Sistema em Teste (aquele que contém a UUT). Essa independência é proporcionada pelos mecanismos *ferry-clip* (Zeng, 1985), que compreende os componentes: *active-ferry*, *passive-ferry* e *ferry-channel* para lidarem somente com as informações dos testes. No Sistema de Teste, o componente *Test Engine* interage com o usuário e executa o roteiro

de teste (*test script*). O roteiro de teste controla o fluxo das *entradas,* das *falhas* e observa as *saídas* em cada ponto de observação e controle (PCO); (a definição de PCO é apresentada a seguir). A *Ferry-injection* prevê, no Sistema de Teste, um testador superior (UT), um inferior (LT) e o componente *Fault Injection Controller* (FIC) para controlar as falhas. O Sistema em Teste contém o componente *passive-ferry* que faz as *entradas* atingirem a UUT e o *Fault Injection Module* (FIM) que faz as falhas serem executadas. Cabe observar que a arquitetura *Ferry-injection* não impõe uma solução de implementação dos meios de testes, simplesmente sugere os componentes necessários. Estudos com o uso dessa arquitetura podem ser encontrados em Martins et al. (2002) e Martins e Mattiello-Francisco (2003b).

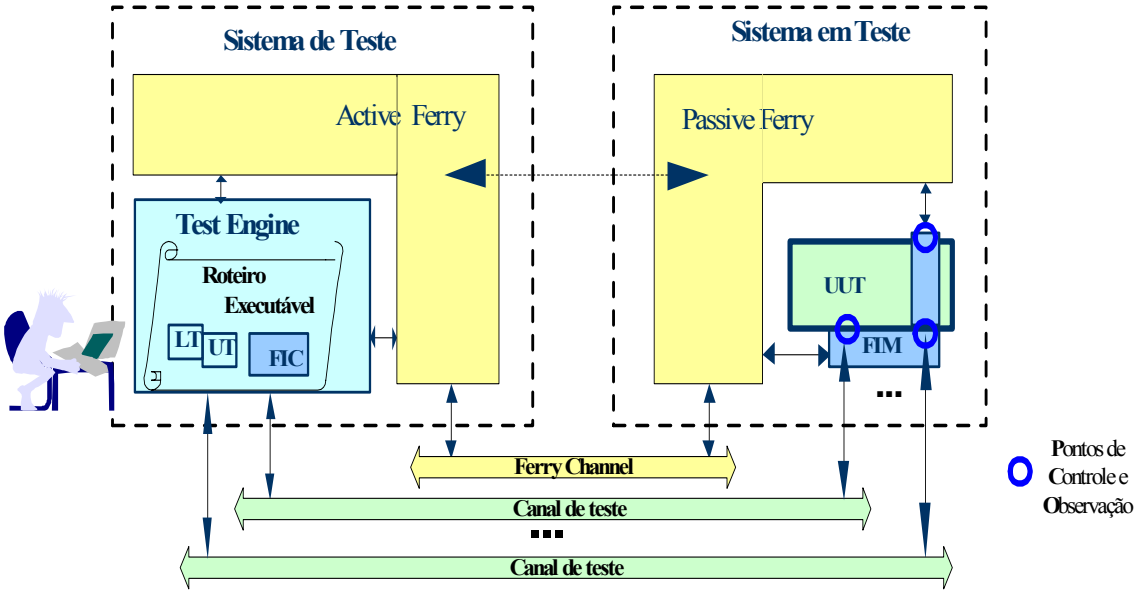


FIGURA 3.5 - Arquitetura de teste *Ferry-injection*.

A arquitetura afeta fortemente os requisitos de conformidade que podem ser efetivamente checados por meio de testes (Cavali et al, 1996); isso quer dizer que, somente os casos de teste e casos de falha que são possíveis de serem executados com os componentes disponíveis da arquitetura serão incluídos na SAT. Muito fortemente relacionados com a arquitetura, estão os pontos de controle e observação (PCOs), dos quais depende o projeto dos testes (ver Capítulo 4).

- *Pontos de controle e observação (PCO)* – correspondem ao local por onde fluem as *entradas* e *saídas* dos casos de testes na UUT que podem ser controladas e observadas pelo ambiente de teste. Segundo a IS 9646 (ISO, 1991), as entradas e saídas podem ser vistas como eventos de teste, os quais podem ser na forma de primitivas (Abstract Service Primitives - ASPs) e unidades de dados (Protocol Data Units - PDUs). Os eventos de teste podem ser alterados através de falhas injetáveis.

- *Notação de teste* – a SAT deve ser escrita em uma *notação de teste* que seja independente da implementação. A ISO definiu a notação de teste denominada Testing and Test Control Notation (TTCN[4]) (ETSI, 2003). O processo $CoFI_p$, não estabelece uma notação especial para documentação da SAT. No entanto, sob o ponto de vista de automação dessa atividade, uma linguagem que facilita a tradução da SAT para uma linguagem executável é discutida no Capítulo 4.

A Especificação dos Serviços Normatizados ou mesmo a Especificação Técnica, pode omitir a definição precisa de endereços, nome das mensagens, primitivas ou variáveis; entretanto, uma indicação clara dos PCOs é fundamental. A visibilidade dos eventos nos testes através dos PCOs está fortemente relacionada com a *testabilidade* da UUT; isto é, com o grau em que um sistema facilita o estabelecimento de critérios e desempenho dos testes para determinar se aqueles critérios foram satisfeitos.

A especificação da *Seqüência abstrata de teste (SAT)* para uma dada Especificação de Serviços Normatizados associada a uma Especificação Técnica é definida de forma que, para cada propósito de teste, um conjunto de casos de teste e de casos de falha abstratos é especificado para atingir esse propósito. Como mencionado anteriormente, a definição dos casos de teste e de falhas é limitada, pois depende da acessibilidade dos PCOs, da disponibilidade dos recursos da arquitetura de teste e, da definição do modelo de falhas a ser aplicado. Visando apoiar essa atividade, a metodologia de teste também

[4] Nas versões anteriores, TTCN abreviava Tree and Tabular Combined Notation.

denominada $CoFI_m$, apresentada no Capítulo 4, define uma série de passos que orientam a derivação da SAT.

Relacionada às atividades de "Definir os propósitos" e "Definir a SAT" está o conceito de critério de teste. O critério na primeira atividade é cobrir *todos os serviços especificados*. Na segunda atividade, diminui-se a granularidade, sendo o critério aplicado para cada serviço, o de cobrir *todo o comportamento normal e o comportamento excepcional incluindo os mecanismos de tratamento de falhas externas,* conforme estabelecido no modelo de falhas para todos os serviços.

3.4.3 Selecionar os Casos de Teste e os Casos de Falha

A SAT é projetada para todos os serviços especificados, sem levar em conta restrições da implementação. Dessa forma, uma seleção dos casos de teste e casos de falha da SAT, que estão relacionados com os serviços de fato implementados, faz-se necessária. Essa atividade requer informações sobre:

- todos os componentes disponíveis na arquitetura Ferry-injection, inclusive os injetores de falhas. Essas informações estão contidas no Documento de descrição das Ferramentas de Teste (DFT), produzido pelo grupo responsável pelos testes.

- as capacidades dos serviços implementados, inclusive os pontos de controle e observação disponibilizados na UUT e acessíveis pelos componentes da arquitetura Ferry-injection. Tais informações estão contidas na Declaração de Conformidade da Implementação (DCI), documento produzido pelo grupo responsável pela implementação da UUT.

A saída dessa atividade é um documento contendo *a seqüência abstrata dos testes selecionados* (SATS), isto é, os casos de teste e de falha que devem ser executados. A FIGURA 3.6 ilustra os artefatos relacionados a essa atividade.

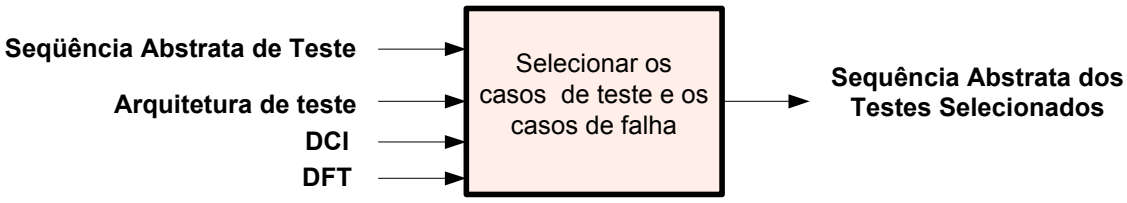


FIGURA 3.6 - Atividade de Seleção de casos de teste e casos de falha.

Nessa atividade, o critério de teste é específico para os serviços implementados na UUT.

3.4.4 Estabelecer a Seqüência Parametrizada Executável de Teste

Essa atividade é caracterizada pela parametrização e tradução da seqüência abstrata para uma notação executável. Essa tradução é feita somente para os casos de teste e de falha selecionados na atividade anterior (ver Seção 3.4.3).

Os testes selecionados, que constam da *Seqüência Abstrata dos Testes Selecionados*, são parametrizados com as informações extras sobre a UUT. As informações extras constam do documento Informações extras sobre a Implementação em Teste (IXIT), preparado e fornecido pelo grupo responsável pela implementação da UUT. Exemplos de informações extras são: valores de contadores, de temporizadores, endereços para acesso aos PCOs e outras variáveis importantes na execução do serviço.

Ao final dessa atividade, os casos abstratos acrescidos das informações extras são traduzidos para uma *linguagem* executável (uma linguagem de programação interpretada ou compilada). O conjunto de casos de teste e de falhas produzido aqui compõe o artefato denominado *Seqüência Parametrizada, Executável de Teste* (SPET). A FIGURA 3.7 mostra as informações e os artefatos requeridos e produzidos nessa atividade.

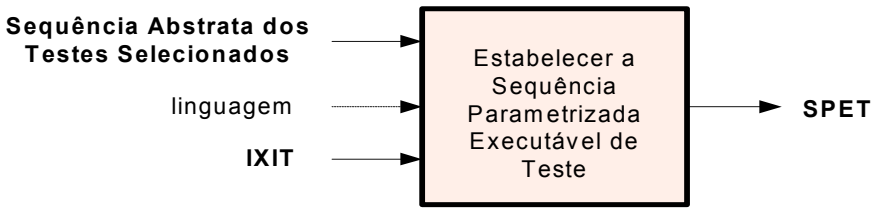


FIGURA 3.7 - Atividade de Estabelecimento da SPET.

No processo $CoFI_p$, nenhuma linguagem específica é recomendada desde que possa ser executada por computador. Entretanto, cabe observar que uma alternativa é o uso do perfil de teste definido pela OMG, o UML *Testing Profile Specification* (OMG, 2005). A ECSS recomenda a linguagem PLUTO para elaboração de procedimentos de teste e operações de satélites na norma ECSS-E-70-32A (ECSS, 2001), na tentativa de fazer convergir a indústria espacial européia em torno de uma linguagem única.

3.4.5 Executar as Campanhas de Teste

Essa atividade consiste em realizar uma campanha de teste, ou seja, executar todos os casos de teste e casos de falha contidos na SPET para uma UUT particular e produzir o *Log de Teste*.

A execução da campanha requer ferramentas de teste que implementam as funções dos componentes da arquitetura *Ferry-injection* (ver Seção 3.4.2). Durante a execução de uma campanha de teste, o Sistema de Teste envia para a UUT (através dos PCOs) as *entradas* normais especificadas nos casos de teste e casos de falha, como ilustrado na FIGURA 3.5. Com relação ao teste dos mecanismos de tratamento de erro, além das entradas, o sistema de teste envia ao módulo injetor, as informações que definem uma falha (o tipo de falha, quando a falha deve ser ativada, o número de ativações, etc) de acordo com o caso de falha especificado. A injeção da falha se dá de acordo com os mecanismos dos injetores do Sistema em Teste. Vale observar que os injetores de falha são responsáveis pela ativação das falhas, de forma independente da ativação das entradas normais.

Durante a campanha, todas as reações observadas e todos os resultados das execuções de uma seqüência de teste são registrados no documento denominado *Log de Teste*. Os artefatos de entrada e os produzidos nessa atividade são ilustrados na FIGURA 3.8.

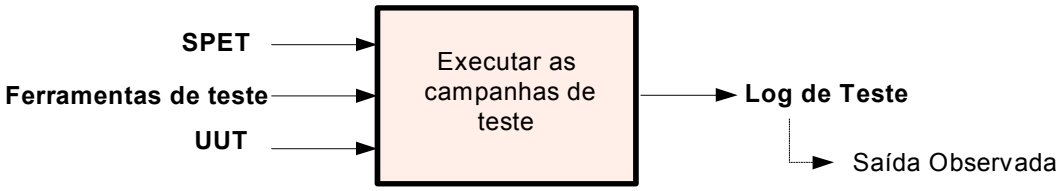


FIGURA 3.8 - Atividade de Execução das Campanhas de Testes.

Dentre as reações registradas no *Log de Teste* devem estar, as entradas de teste, as saídas observadas, assim como quaisquer outros eventos observados no Sistema em Teste. O conjunto particular de todas as saídas observadas que caracterizam o comportamento na presença de falhas são chamados *read-outs* (Powell, 1991). Toda informação armazenada nessa atividade serve não somente para a análise de resultados, como também, para referência futura e, portanto deve ser mantida pelos responsáveis pela execução dos testes.

3.4.6 Analisar os Resultados dos Testes

A especificação de cada caso de teste e cada caso de falha abstrato que compõem o artefato *Seqüência abstrata de teste (SAT)* é composta por entradas e *saídas esperadas* correspondentes (ver a atividade "Definir SAT" na Seção 3.4.2).

Na atividade de "Executar as campanhas de teste" (ver Seção 3.4.5.), cada caso de teste e cada caso de falha é associado às *saídas observadas* durante a execução do caso, informação essa registrada no *Log de Teste*. Dessa forma, na atividade de "análise dos resultados dos testes", a saída esperada de cada elemento da SAT é comparada com a saída observada.

O resultado da comparação permite atribuir a cada caso de teste e caso de falha executado um veredicto do teste. Todos os resultados dos testes são documentados no

artefato denominado *Resultado dos Testes*. A FIGURA 3.9 ilustra os artefatos e as informações requeridos e o artefato produzido nessa atividade.

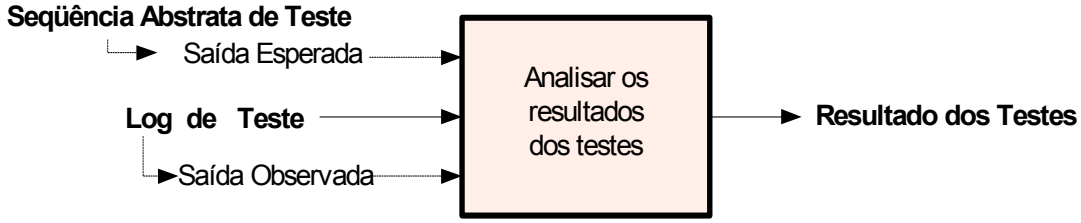


FIGURA 3.9 - Atividade de Análise dos resultados dos testes.

A classificação do veredicto dos testes, de acordo com as definições da IS 9646 (ISO, 1991) é: (i) *passou* - a saída observada evidencia a conformidade do requisito para o qual o propósito de teste do caso de teste ou do caso de falha foi definido; (ii) *falhou* - a saída observada demonstra não conformidade do requisito para o qual o propósito de teste do caso de teste ou de falha foi definido ou, contém pelo menos um evento de teste inválido com relação aos requisitos; (iii) *inconclusivo* - a saída observada é tal que não permite atribuir o veredicto de passou nem o de falhou; neste caso pode ter ocorrido um *erro durante a execução ou término anormal*.

3.4.7 Gerar o Relatório de Conformidade, Robustez e as Estatísticas dos Testes

Essa atividade consiste em elaborar o Relatório de Conformidade, robustez e as estatísticas dos testes. O relatório é baseado no Resultado dos Testes, gerado durante a análise dos resultados. A análise dos resultados permite a conclusão sobre o veredicto de cada caso de teste. O caso de teste está relacionado a um requisito de conformidade que fora associado aos propósitos de teste. A FIGURA 3.10 ilustra os artefatos requeridos e produzidos nessa atividade.

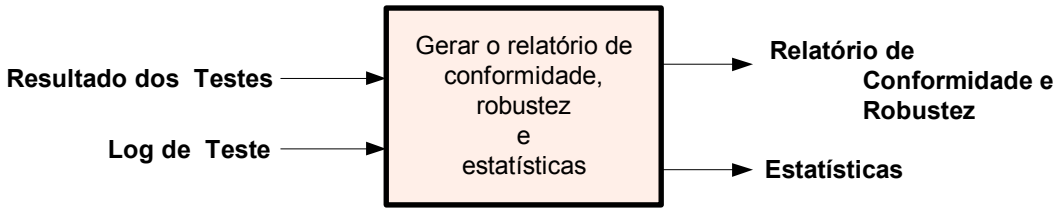


FIGURA 3.10 - Atividade de Geração do relatório de conformidade.

A atribuição do veredicto pode ser manual ou automática. Os veredictos são atribuídos tanto aos casos de teste como aos casos de falha. Para o segundo, o veredicto é atribuído somente se o comportamento esperado frente à falha fora devidamente determinado na definição do caso de falha. Esse veredito permite avaliar a robustez. O Log de teste pode também ser consultado para compor os resultados.

O artefato contendo as *estatísticas* sobre a execução dos testes inclui porcentagem de não conformidades, falhas injetadas com relação às que foram ou não percebidas pelo sistema. Os cômputos e registros no relatório de estatísticas são usados para diagnósticos.

3.5 Fluxo dos Artefatos do Processo $CoFI_p$

As atividades do processo $CoFI_p$ podem ser agrupadas em três fases: preparação dos testes, operação ou execução dos testes e análise dos resultados. As figuras FIGURA 3.11, FIGURA 3.12 e FIGURA 3.13 ilustram os artefatos do processo $CoFI_p$ usados respectivamente nas fases citadas.

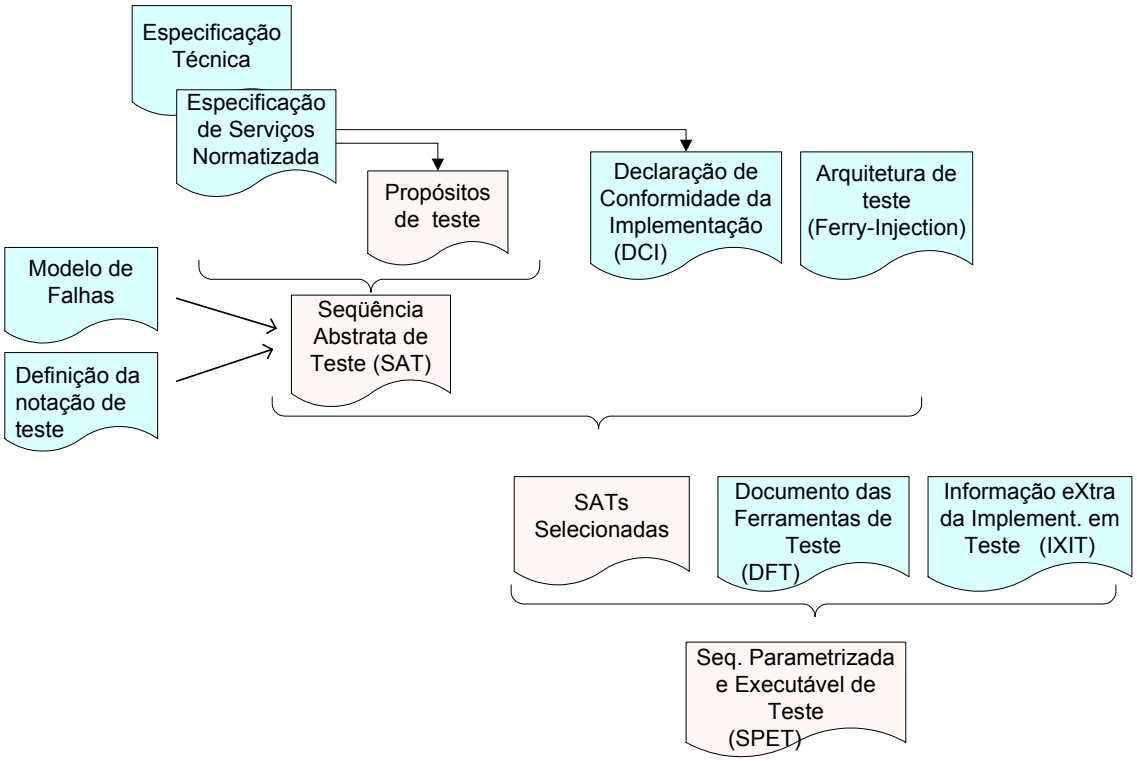


FIGURA 3.11 - Artefatos usados na preparação dos testes.

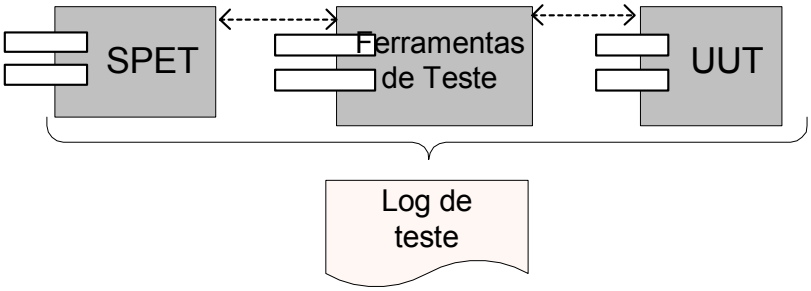


FIGURA 3.12 - Artefatos usados na operação dos testes.

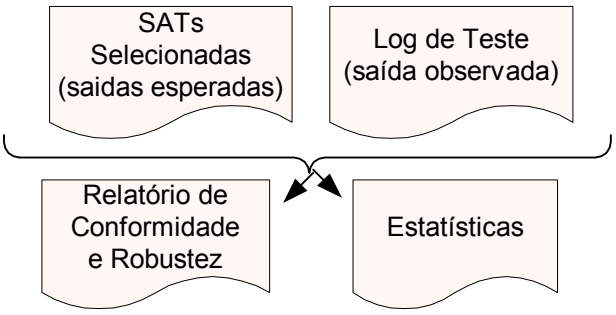


FIGURA 3.13 - Artefatos usados na análise dos resultados.

3.6 Comparação entre os Processos de Teste: $CoFI_p$, ISO e Drabick

Neste texto, o processo de teste descrito na Norma IS-9646 é referenciado como processo ISO, o processo de teste definido em Drabick (2004) é referenciado como processo Drabick.

O processo $CoFI_p$ modifica o processo ISO acrescentando artefatos e subatividades para que testes para avaliação do sistema em presença de falhas fossem considerados na avaliação de conformidade. Uma comparação entre as atividades do processo $CoFI_p$ (conformidade e injeção de falhas), do processo ISO (conformidade) e do processo Drabick (teste formal de software) é mostrada na FIGURA 3.14.

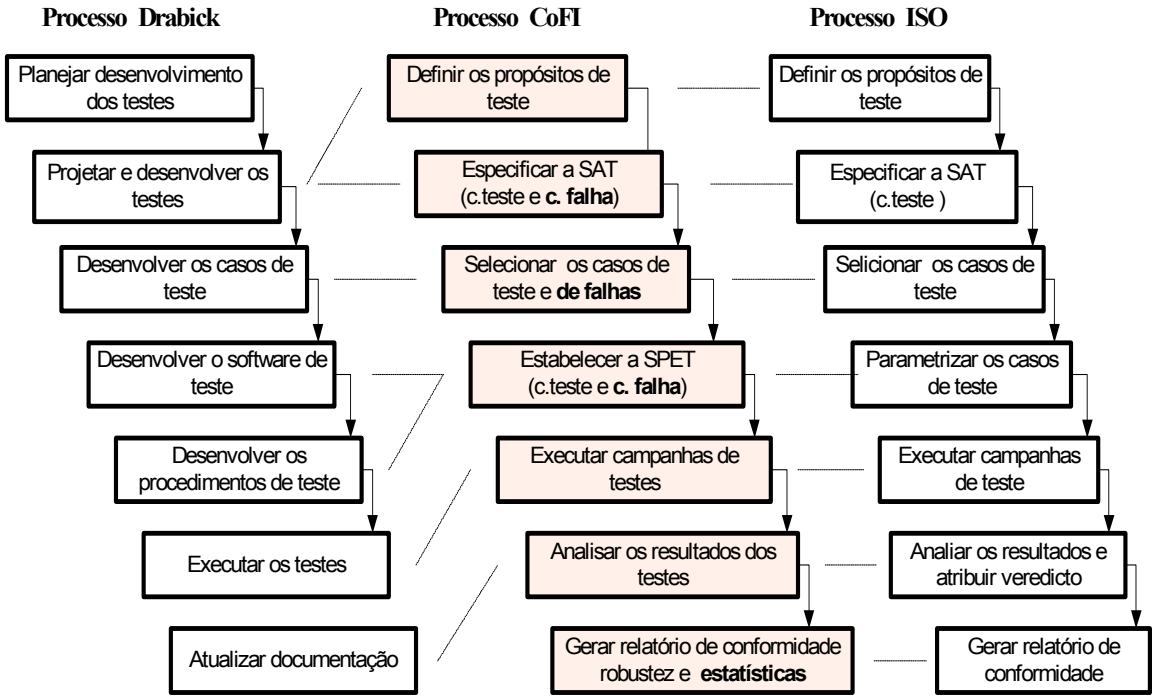


FIGURA 3.14 - Atividades dos processo de teste Drabick, $CoFI_p$ e ISO.

O processo de Conformidade e Falhas Injetáveis ($CoFI_p$) é mais especializado do que aquele apresentado por Drabick, cobrindo apenas as atividades de testes de conformidade voltadas para a validação de software de comunicação em aplicações espaciais. A TABELA 3.3 e a TABELA 3.4 comparam as diferentes características do $CoFI_p$ com relação ao processo Drabick e ao processo ISO, respectivamente.

TABELA 3.3 - Processo de teste $CoFI_p$ versus Processo Drabick.

	Processo $CoFI_p$	**Processo Drabick**
1	-	inclui atividade de planejamento dos testes incluindo cronograma, recursos, abordagem e o escopo dos testes.
2	específico para testes de sistema e teste de aceitação, teste caixa-preta	cobre todas as atividades de teste no ciclo de desenvolvimento do software. Na fase de teses de unidade, o código fonte do software é requisitado
3	não requer código fonte da unidade em teste	
4	inclui especificação de testes em uma forma abstrata de forma que eles podem ser reutilizados	-
5	inclui uma atividade de seleção de casos de teste em conseqüência do item 4 dessa tabela.	-
6	a criação do *roteiro de teste* é decorrente da tradução dos casos de teste para a forma executável na atividade *Estabelecer a SPET*	a criação do *programa de teste* está prevista na atividade *Desenvolver o software de teste*. A associação entre o programa de teste e os casos de teste a serem exercitados não é explícita
7	não prevê artefato com procedimentos de teste, pois supõe a automação da atividade de execução dos testes	prevê elaboração de *Procedimentos de teste* criados com base nos Casos de teste. Estes procedimentos constam de instruções passo-a-passo para executar os testes.
8	o projeto de casos de teste inclui a definição das saídas esperadas, que serão comparadas com as saídas observadas, para definição do veredicto de teste (passou/falhou/inconclusivo) na atividade Analisar resultados	o projeto de casos de teste inclui a definição do critério passou/falhou. Essa atribuição é deixada a percepção da pessoa responsável pelos testes
9	a análise de resultados aponta conformidade com os requisitos e desvios de comportamento na presença de falhas e estatísticas.	a análise de resultados aponta limitações e incidentes descobertos.

TABELA 3.4 - Processo de teste $CoFI_p$ versus Processo ISO.

	Processo $CoFI_p$	**Processo ISO**
1	provê um guia para a geração de casos de teste de conformidade	não orienta a geração dos casos de teste
2	provê um guia para a geração de casos de falha	não orienta a geração dos casos de falha
3	prevê uma SAT que contém casos de falha que permitem checar se a UUT suporta ou não influências ambientais que podem causar falhas	prevê várias SATs compostas por casos de testes que permitem checar a conformidade entre a UUT e a sua especificação
4	não determina uma linguagem para escrever a SAT, mas sugere o *test profile* da UML visando automação da execução	as SATs são escritas em TTCN
5	adota a arquitetura *Ferry-injection* para apoiar a execução dos testes	define métodos de teste para apoiar a execução dos testes locais, remotos, distintos
6	prevê automação na atividade de análise dos resultados	-
7	o relatório de teste aponta conformidade com os requisitos, desvios de comportamento na presença de falhas (robustez) e estatísticas	o relatório de teste indica a conformidade com os requisitos

3.7 Considerações Finais

Neste Capítulo foi apresentado o processo de teste proposto nesta tese, denominado $CoFI_p$. Uma visão geral do processo é mostrada no fluxo de atividades e uma descrição detalhada mostra os artefatos de entrada e aqueles produzidos nas atividades. O processo foi comparado com o processo de teste de conformidade dado na norma IS-9646, que inspirou o $CoFI_p$, visando apontar as diferenças, bem como, com o processo de teste de software, recentemente publicado em Drabick (2004). O processo de teste $CoFI_p$ também é situado nos processos de engenharia de software de acordo com a norma IEEE-12207 (IEEE, 1995) e com a ECSS-E-40 Parte B (ECSS, 2003c). Esta última, consta de uma versão da anterior adaptada para o desenvolvimento de software em aplicações espaciais, proposta pela Agencia Espacial Européia.

O processo ISO foi ajustado às necessidades de validação de software em aplicações especiais, as quais requerem além de conformidade, uma avaliação do comportamento da unidade em teste na presença de falhas. Os ajustes incluíram projeto de casos de falha a serem acrescentados na seqüência abstrata de teste. Ao invés de métodos abstratos de teste, foi adotada a arquitetura *Ferry-injection* para orientar a definição dos meios necessários para execução dos testes, pois incluem componentes para acelerar a ocorrência de falhas. A geração de uma seqüência abstrata de teste a partir de especificações padrões foi mantida, porém, as normas que especificam protocolos foram substituídas pelas normas que definem serviços de software em aplicações espaciais que vêm sendo definidas pelos órgãos de padronização de sistemas espaciais, como o CCSDS e o ECSS. A vantagem do processo $CoFI_p$ é ainda possuir alternativas para derivação de testes a partir de uma norma, estar associado a uma metodologia para geração de testes a partir de uma descrição textual.

As definições do processo $CoFI_p$ apresentadas aqui não incluem formato e conteúdo detalhado dos artefatos de teste, nem, contudo, detalham as possibilidades de retorno às atividades anteriores. Entretanto, elas contribuem para a melhoria dos processos de teste que podem ser úteis para redução dos riscos de falhas nos sistemas espaciais.

CAPÍTULO 4

CoFI$_m$: UMA METODOLOGIA PARA GERAÇÃO DE TESTES PARA VALIDAÇÃO DO COMPORTAMENTO DE SOFTWARE EM CONFORMIDADE E EM PRESENÇA DE FALHAS

Neste Capítulo é apresentada a metodologia CoFI$_m$, cujo objetivo é amenizar a natureza *ad hoc* para criação de casos de teste, introduzindo uma sistemática para a atividade de *projetar teste* que proporciona reuso dos mesmos. Ela está associada à atividade de "Definir a Seqüência Abstrata de Teste (SAT)" do processo descrito no Capítulo 3. Através de seus passos, em uma abordagem *top-down*, a especificação, em notação textual, é traduzida para uma notação formal para que casos de teste e casos de falha possam ser gerados automaticamente.

As seções deste Capítulo apresentam uma descrição detalhada dos passos da metodologia, uma visão geral de como a metodologia CoFI$_m$ lida com a complexidade do modelo de teste; as falhas que se espera descobrir, o relacionamento dos passos, um resumo das vantagens dos passos e, finalmente, considerações sobre a automação da metodologia.

4.1 Abstração e Uso de Diagramas UML na Atividade de Teste

A metodologia CoFI$_m$ apóia-se na técnica mais efetiva de lidar com a **complexidade** de uma atividade, a *abstração* (Alagar e Periyasamy, 1998). Alagar discute vários aspectos da complexidade relacionados a um sistema de software: complexidade de tamanho, estrutural, ambiental, do domínio da aplicação e de comunicação.

Em teste, a complexidade de tamanho está relacionada com a especificação e um conjunto de casos de teste gerado. Na CoFI$_m$ essa complexidade foi tratada de duas formas: na primeira identifica-se a abstração *serviço* e geram-se testes para cada serviço isoladamente (conceito de decomposição para geração de teste); na segunda separa-se o comportamento normal e excepcional em modelos distintos.

A complexidade estrutural foi tratada com o uso de diferentes diagramas de UML, uma vez que não há um modelo ideal para modelar toda complexidade do comportamento de um software. Três diagramas diferentes da UML foram usados: casos de uso, diagrama de seqüência e diagramas de estados; cada um deles abstrai um aspecto da realidade. A cada passo são requeridas novas informações que são acrescentadas gradativamente na construção dos diagramas. A partir de diagramas de estados, casos de teste são gerados com ferramentas baseadas em métodos formais.

4.2 Contexto das Falhas na Abordagem *COFI$_m$*

Segundo Binder (2000), as não conformidades que podem existir entre a especificação de requisitos e a unidade em teste (UUT) são causadas por falhas de: (i) omissões - capacidade requerida ausente na implementação; (ii) comissões - capacidade errônea, podendo ter sido requerida ou não, (iii) extras - capacidade não requerida, podendo causar uma surpresa maligna ou benigna na UUT. Falhas de hardware, que podem levar à ativação ou não dos mecanismos de tolerância a falhas, foram acrescentadas no contexto da abordagem CoFI. O Diagrama de Venn da FIGURA 4.1 ilustra essas possíveis não conformidades.

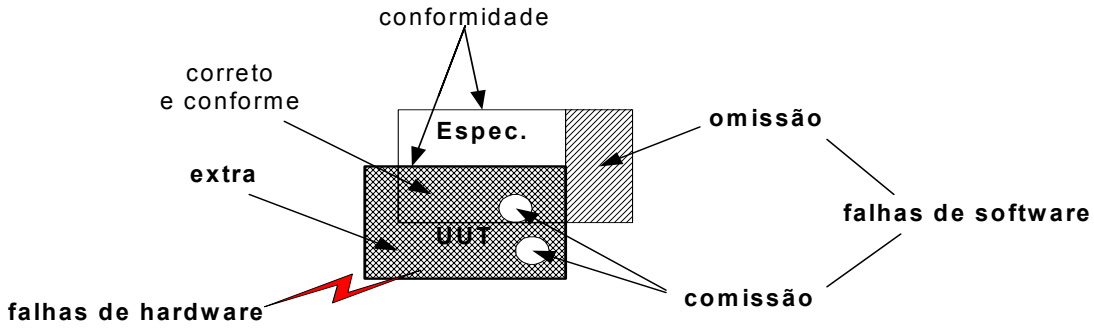


FIGURA 4.1 - Esquema de falhas da unidade em teste com relação a especificação.

O teste de conformidade encontra essas inconsistências entre a especificação e a UUT, submetendo à UUT, entradas corretas e errôneas em seqüências previstas. A validação com experimentos de injeção de falhas facilita a detecção de falhas nos mecanismos de

tolerância a falhas de hardware. Aqui, a UUT deve ser submetida a entradas corretas e a falhas de hardware, as quais podem acontecer independentemente da seqüência prevista, em qualquer momento durante a execução.

Na $CoFI_m$ o objetivo é analisar o comportamento de uma unidade em teste, sob entradas normais e excepcionais esperadas, sob entradas normais em momentos não esperados, bem como sob entradas alteradas pelas condições externas. A TABELA 4.1 resume os tipos de entradas errôneas que podem causar não conformidade e o modelo de falhas usado para derivação dos casos de falha pela metodologia $CoFI_m$.

TABELA 4.1 - Tipos de entradas errôneas cobertas pela $CoFI_m$.

Tipo de entradas errôneas	***Modelo de falha***
explicitamente especificadas	Especificação textual + Diagrama de estados
causadas pelo ambiente e supostamente tratadas pelos mecanismos de tolerância a falhas.	Modelo de falhas físicas + Diagrama de seqüência
entradas corretas explicitamente especificadas que chegam em momentos não esperados, também chamados *eventos furtivos* (Binder, 2000)	Matriz de Transições

As entradas errôneas causadas pelo ambiente são importantes para disparar os mecanismos de tolerância a falhas implementados em software de aplicações espaciais, os quais devem resistir as falhas provocadas pela radiação do ambiente espacial.

4.3 Descrição Geral da Metodologia *COFI$_m$*

A metodologia CoFI consiste de uma série de passos que transforma uma descrição textual de serviços (da área espacial) em uma especificação formal (em máquina de estados finita), a partir da qual uma seqüência abstrata de teste é gerada sob as condições e restrições da arquitetura de teste *Ferry-injection*. Os documentos que descrevem textualmente os serviços constam de normas e especificações técnicas (definindo especificidades de uma missão espacial particular). Os passos intermediários fazem uso de artefatos UML, tais como, caso de uso, diagrama de seqüência e diagrama

de estados. A metodologia prevê a geração de uma seqüência abstrata de teste apoiada por algoritmos de geração automática de testes a partir de especificações em máquinas de estados finitas. A seqüência de teste gerada é composta tanto por casos de teste (de conformidade) quanto por casos de falha (aqueles que serão executados pela técnica de injeção de falhas). A FIGURA 4.2 apresenta uma visão geral da metodologia $CoFI_m$.

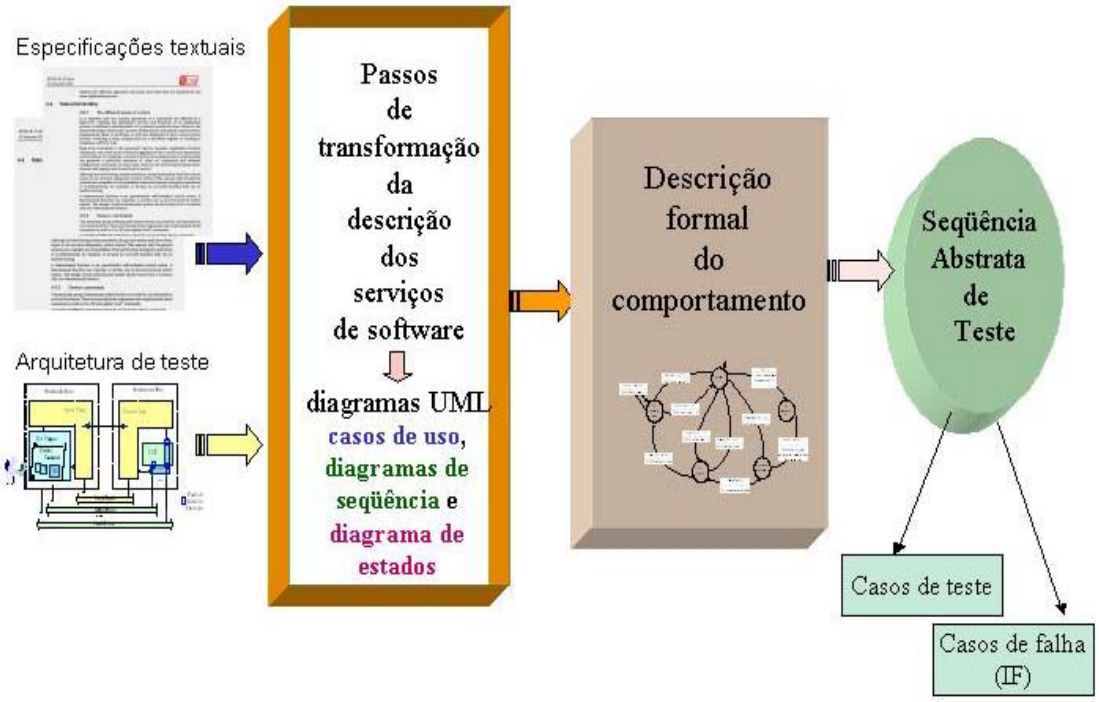


FIGURA 4.2 - Visão geral da metodologia $CoFI_m$.

A metodologia $CoFI_m$ consiste de treze passos, os quais são resumidamente descritos na TABELA 4.2.

A ordem e a dependência dos passos da metodologia são apresentadas no diagrama de atividades ilustrado na FIGURA 4.3.

TABELA 4.2 - Passos da metodologia $CoFI_m$.

Passo	Descrição
p1	identificar um serviço, para o qual derivar os testes e associá-los aos propósitos de teste
p2	identificar usuários do serviço e o meio físico de comunicação usado
p3	identificar as entradas, as saídas, a arquitetura de teste, os pontos de controle e observação e as variáveis operacionais relacionados ao serviço
p4	descrever o serviço como *casos de uso*. Cada caso de uso é composto por cenários (base, alternativos e excepcionais). Este passo inclui: • p4.1.) definir um *cenário base*, • p4.2.) derivar *cenários alternativos* a partir do cenário base, mapeando as situações normais, • p4.3.) derivar *cenários excepcionais*, mapeando as situações de exceção especificadas
p5	traduzir os cenários normais (base e alternativos) em *diagramas de seqüência*
p6	derivar *diagramas de estados normais* eqüivalentes aos diagramas de seqüência do passo 5
p7	gerar automaticamente *casos de teste* de conformidade a partir do diagrama de estados
p8	criar uma *matriz de transição*. Este passo inclui: • p8.1.) transcrever o diagrama de estados do passo 6 para essa matriz, • p8.2.) completar os campos da matriz com saídas esperadas para as transições não especificadas, sempre que possível
p9	escolher o *modelo de falhas* que representa as conseqüências que anomalias do ambiente podem causar no sistema em teste e associar primitivas de falhas de acordo com a arquitetura de teste
p10	criar cenários de exceção em *diagramas de seqüência* para os: • p10.1.) cenários das exceções especificadas (passo 4.3) • p10.2.) cenários da suposição de completeza da matriz de transições (passo 8.2) • p10.3.) cenários dos mecanismos de tolerância a falhas (identificados ao combinar-se as primitivas do modelo de falhas com os cenários normais do passo 5)
p11	derivar *diagramas de estados excepcionais* equivalentes aos digramas de seqüência do passo 10, indicando as transições de falha
p12	derivar *casos de falha*, a partir dos cenários de exceção, seja via diagramas de seqüência (passo 10), seja via os diagramas de estados excepcionais (passo 11). Ao derivar os casos de falha define-se o *workload* e *instâncias de falhas* a serem executadas por injeção de falhas
p13	compor a Seqüência Abstrata de Teste com os casos de teste (passo 7) e os casos de falha (passo 12)

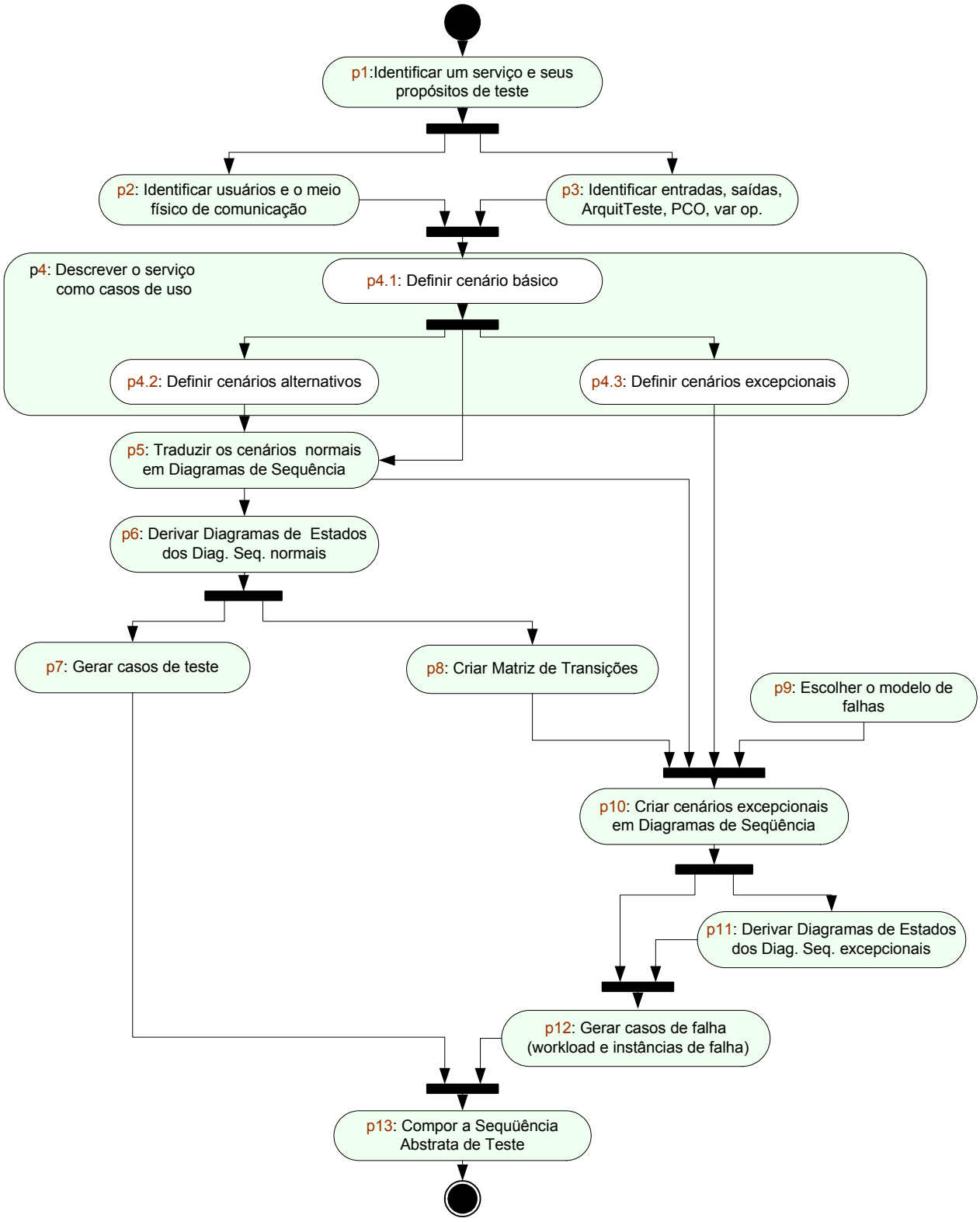


FIGURA 4.3 - Passos da metodologia $CoFI_m$.

A metodologia $CoFI_m$ manipula vários elementos, entre artefatos UML e conceitos, entre eles estão: propósito de teste, descrição do serviço, usuário, caso de uso, variável operacional, cenários, entradas e saídas, PCO, modelo de falhas, diagrama de seqüência: normal e excepcional, diagrama de estados: normal e excepcional, matriz de transição, caso de teste, caso de falha, instância de falha, arquitetura de teste, seqüência abstrata de teste. Esses elementos são apresentados no diagrama de classes da FIGURA 4.4, sendo cada elemento uma classe e as ligações entre elas representam o relacionamento entre si.

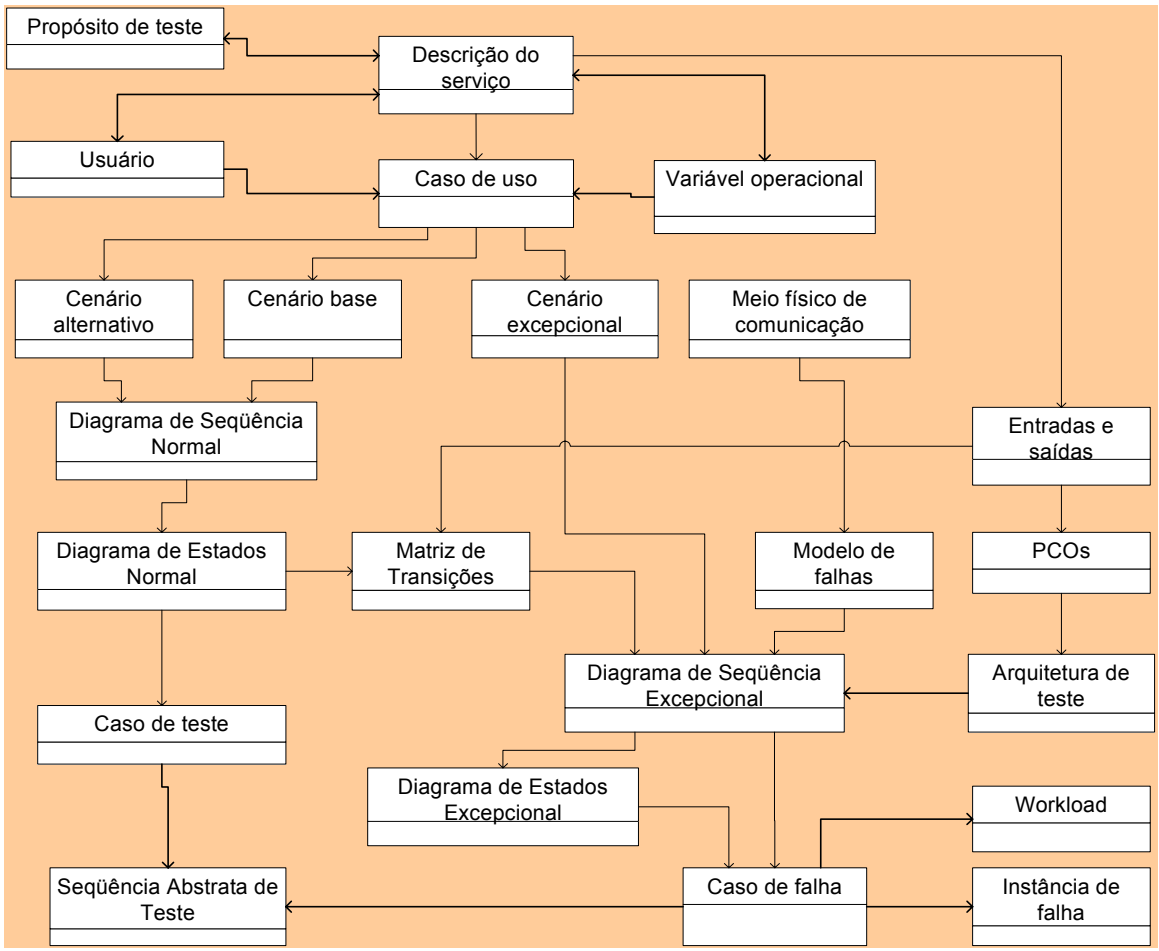


FIGURA 4.4 - Relacionamentos entre elementos manipulados pela $CoFI_m$.

4.4 Descrição Detalhada do Passos

Esta Seção descreve e ilustra cada passo da metodologia. O protocolo chamado OBDH–EXP, definido e desenvolvido no INPE, foi usado para ilustrar a descrição da metodologia $CoFI_m$. De fato, somente a função de reconhecimento de comandos, parte do protocolo, foi usada nas ilustrações. Esse protocolo foi definido para atender a comunicação entre dois equipamentos a bordo de satélites científicos, o computador principal (OBDH) e os experimentos inteligentes dos satélites científicos (INPE, 2005), como esquematizado na FIGURA 4.5.

O protocolo OBDH–EXP, implementado em um experimento científico com processador inteligente é a unidade em teste (UUT). A função de reconhecimento de comandos consta da identificação de um comando enviado pelo computador principal a bordo do satélite (do inglês, *on-board data handling*, OBDH) a um experimento científico, o qual é chamado, de forma genérica, de EXP. O EXP envia dados e respostas ao OBDH somente sob demanda. Ao reconhecer o comando, o EXP o executa e responde ao OBDH. Os comandos que podem ser reconhecidos e executados pelo EXP são: reinicialização (*reset*), reconfiguração, transmissão de dados, carga (*load*) de memória, descarga (*dump*) de memória, obtenção do tempo (*clock*), inicialização e interrupção de aquisição de dados, carga de parâmetros. Sempre que um comando não é reconhecido toda a mensagem é descartada. Se um byte do comando não chega dentro de 500 ms toda a mensagem é descartada e o EXP volta a aguardar novo comando. Outros detalhes do protocolo são ilustrados nas seções seguintes.

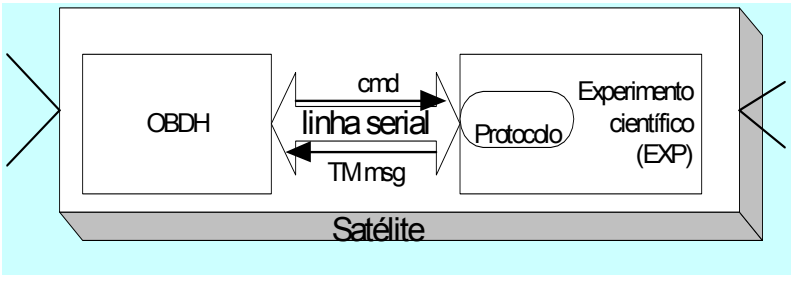
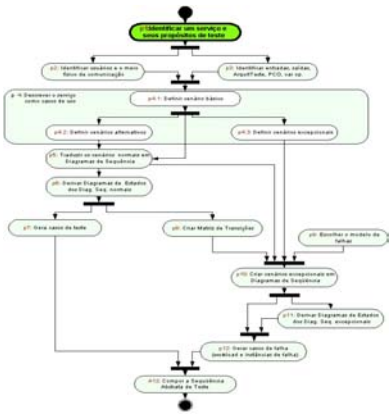


FIGURA 4.5 - Esquema de comunicação OBDH-EXP.

4.4.1 Passo 1: Identificar um Serviço e seus Propósitos de Teste



O primeiro passo da metodologia $CoFI_m$ consiste em identificar na Especificação de Serviços Normatizada e na Especificação Técnica, um serviço a ser testado. A este seviço é associado um ou mais propósitos de teste, dentre aqueles definidos na atividade 2 do processo $CoFI_p$, conforme apresentado na Seção 3.4.2.

Os passos da metodologia devem ser aplicados a cada um dos serviços, um de cada vez, de forma que os testes sejam gerados para todos os serviços.

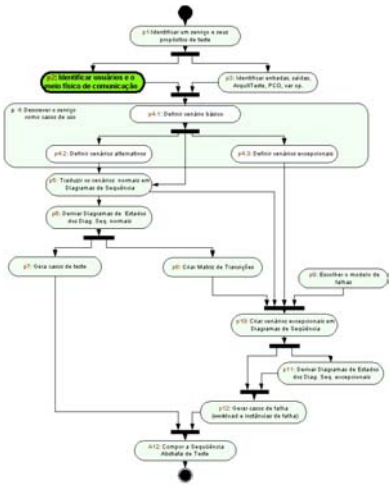
A vantagem de tratar cada serviço isoladamente é tornar factível o uso de métodos formais para a geração de casos de teste, evitando-se a explosão de estados na especificação formal ou a explosão no número de casos de teste.

Exemplo

O protocolo OBDH-EXP não contempla o conceito de serviço; entretanto, duas funções distintas são identificadas: *verificação do comando recebido* e *execução do comando*. A primeira função foi, então, selecionada para ilustrar os passos da metodologia. Para essa função apenas um propósito foi associado. O quadro a seguir mostra o nome do serviço e lista os propósitos de teste associados ao serviço.

Serviço: Verificação do comando recebido
Propósito (s) de teste: Checar se o serviço é capaz de reconhecer um comando recebido do OBDH.

4.4.2 Passo 2: Identificar Usuários e o Meio Físico de Comunicação



O segundo passo delimita o contexto do serviço a ser testado. Este passo consiste na identificação tanto dos usuários do serviço como, também, do meio físico de comunicação pelo qual o serviço recebe comandos e envia respostas.

Os usuários do serviço são subsistemas ou processos de aplicação que fazem uso do serviço, solicitando ou passando informações. Os usuários definidos neste passo corresponderão aos atores dos casos de uso no passo 4.

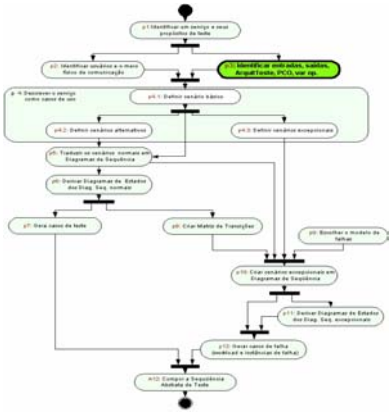
O meio físico de comunicação é o meio através do qual usuários fazem acesso ao serviço. Esse meio pode ser uma interface serial ou paralela, equipamentos e protocolos de uma rede de comunicação, um canal em rádio freqüência, no caso de comunicação entre uma espaçonave e os sistemas em solo, entre outros. A definição do meio de comunicação dará subsídios à definição do modelo de falhas a ser definido no passo 9.

Exemplo

Os usuários e o meio de comunicação do serviço de Verificação do comando recebido são apresentados no quadro abaixo.

Usuários: Processo aplicativo do OBDH , serviço de execução no EXP, relógio do sistema.
Meio de comunicação: linha serial.

4.4.3 Passo 3: Identificar as Entradas, as Saídas, a Arquitetura de Teste, os Pontos de Controle e Observação e as Variáveis Operacionais



O passo 3 consiste, primeiramente, na identificação das entradas e saídas que caracterizam as interfaces do serviço em teste. A descrição das interfaces é encontrada principalmente no documento Especificação de Serviços Normatizada; entretanto, o documento Especificação Técnica complementa informações do primeiro com relação a controle e observabilidade das interfaces. Apenas as entradas e saídas que podem ser observadas durante a execução do serviço são consideradas para teste.

As entradas e saídas de um sistema de comunicação, em geral, são mensagens que podem ser recebidas ou transmitidas em uma *unidade de dado* via um canal de comunicação ou uma *primitiva*. Neste passo define-se então: (i) mensagens de solicitação do serviço e (ii) mensagens de resposta geradas pelo serviço, incluindo mensagens de erro e de alerta.

Em seguida, são definidas as *variáveis operacionais* manipuladas pelo serviço. Estas *variáveis* são fatores cuja variação determinam diferentes respostas do sistema. Elas podem representar endereço, tipo de serviço, condições ambientais que indicam o estado do serviço como, em *crash*, pronto, aguardando evento, *time-out*, *checksum*, etc.; ou ainda valores de parâmetros ou de campos de mensagens de solicitação ou de resposta do serviço, a freqüência das entradas, valores limites aceitáveis e rejeitáveis; tempo de retardo aceitável e *time-out,* além de fórmulas para cálculo de *checksum,* entre outros. O conceito de variáveis operacionais foi definido em Binder (2000, p.282 e 724), para teste de sistemas orientados a objetos e foi adaptado para sistemas de comunicação neste trabalho.

Entradas, saídas e variáveis operacionais auxiliam a definição e caracterização dos cenários nos casos de uso (passo 4) e das interações (estímulos e interrupções) nos

diagramas de seqüência (passos 5 e 10). Neste passo, elas são organizadas em tabelas conforme ilustrado no exemplo a seguir.

Outra definição fundamental neste passo é a *arquitetura de teste* que apóia a realização dos testes e comporta os pontos de controle e observação (PCO). As definições da arquitetura Ferry-injection e de PCOs foram explicadas no Capítulo 3 (ver Seção 3.4.2).

A importância deste passo é que o levantamento de informações para teste a partir das especificações do serviço, permite reduzir discrepâncias entre os requisitos documentados e os requisitos necessários para o correto funcionamento. A discrepância de requisitos e o mau entendimento das interfaces do software são considerados problemas críticos nos sistemas aeroespaciais (Levenson, 2001); portanto, merecem toda atenção e custo de um levantamento cuidadoso.

Exemplo

As entradas, as saídas e as variáveis operacionais levantadas na definição do protocolo OBDH-EXP são apresentadas na TABELA 4.3.

TABELA 4.3 – Entradas, saídas e variáveis operacionais.

Entradas		
	Nome	*Descrição*
	Mensagem de comando	A mensagem de comando recebida pelo EXP (ver Figura 4.5.)
Saídas		
	Nome	*Descrição*
	Mensagem de telemetria do EXP (resposta)	Há 3 mensagens de telemetria que o EXP pode enviar ao OBDH em um pacote TM: *rate-data*, *memory-dump*, *signal*.
Exceção	Time-out	Retorno ao seu estado de espera pelos bits de sincronização.

(Continua).

TABELA 4.3 - Conclusão.

Variáveis Operacionais		
	Nome	*Descrição*
Campos msg cmd	SYNC	bits com valor fixo = EB H; caracteriza o início de uma mensagem. Variável contida no primeiro campo da mensagem de comando recebido pelo EXP.
	EXPID	bits que identificam o experimento. Para o experimento EXP = APEX esse valor é 92H. Variável contida no segundo campo da mensagem de comando recebido pela EXP.
	TYPE	byte que indica o comando solicitado pelo OBDH ao EXP, os quais podem ser: reset, clock, inic acquisition, stop, datatx, reconf, mdump, mload, pload.
	to	tempo máximo de espera de próxima parte de mensagem de comando, 500ms

A mensagem de comando recebida pelo APEX possui seis campos conforme mostra a FIGURA 4.6.

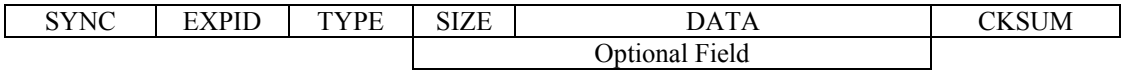


FIGURA 4.6 - Mensagem de comando ao EXP.

A arquitetura *Ferry-injection* adaptada para teste da unidade em questão é ilustrada na FIGURA 4.7. Observam-se na arquitetura de teste, os pontos de observação, o meio físico de comunicação e os injetores de falha.

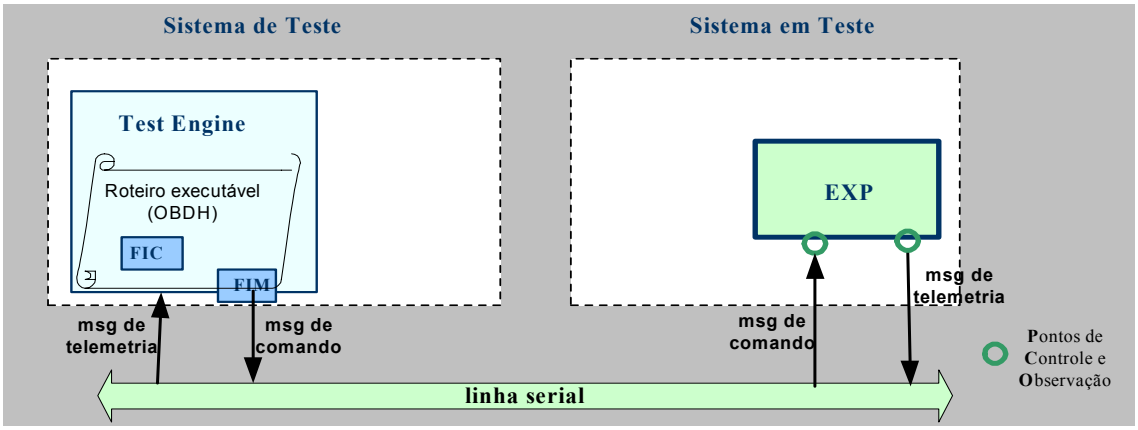
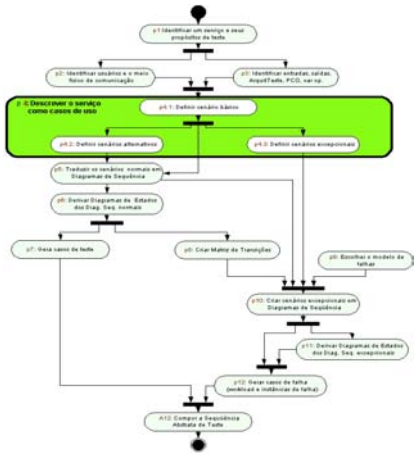


FIGURA 4.7 - Arquitetura *Ferry-injection* para teste do OBDH-EXP.

4.4.4 Passo 4: Descrever o Serviço como Casos de Uso



Neste passo o serviço é descrito como casos de uso. A abstração caso de uso da UML é usada nessa metodologia de teste, para resgatar os requisitos funcionais, sob a expectativa dos usuários do serviço, relacionados com a comunicação externa do serviço.

Um caso de uso representa uma seqüência de interações executadas por um *ator* para atingir um objetivo. Um ator representa um papel desempenhado por um usuário quando interage com o sistema sendo especificado. Um usuário pode ser um ser humano, um dispositivo de hardware ou outro sistema computacional (Ryser e Glinz, 2000). O objetivo a ser atingido pelo ator, traduzido no caso de uso, deve estar associado a um propósito de teste. A descrição do caso de uso, na metodologia $CoFI_m$, segue o esquema definido em Reed (2000), devendo conter: (i) nome do caso de uso; (ii) ator; (iii) descrição resumida das principais características do caso de uso; (iv) condição(ões) de entrada: pré-condição necessária para que o serviço seja inicializado; (v) condição(ões) de saída: pós-condição deixada após a execução do serviço e; (vi) fluxos de ações.

Na descrição de caso de uso, os usuários identificados no passo 2 são os atores e as entradas. As variáveis operacionais levantadas no passo 3 servem como catalisadores de novos cenários. Os fluxos de ações caracterizam cenários.

Um *cenário* é uma realização específica de sucesso ou insucesso de um caso de uso descrito com um número finito de ações. No trabalho de Ryser e Glinz (2000) os termos casos de uso e cenário são usados indistintamente. Aqui eles têm significados diferentes. Três tipos de cenários são balizados na $CoFI_m$: o cenário base, os cenários alternativos e os cenários de exceção.

Este passo consiste dos subpassos, conforme ilustrados na FIGURA 4.3:

- p4.1 – criação do cenário base – esse cenário consta do fluxo das ações que melhor caracteriza o serviço;

- p4.2. – criação dos cenários alternativos – os cenários *alternativos* são derivados do cenário base pela análise de cada ação já definida. Se a ação analisada permitir uma variação normal, um novo cenário alternativo é criado;

- p4.3. – criação dos cenários de exceção - esses cenários também são derivados do cenário base pela análise de cada ação. Caso a variação da ação trate de um erro ou represente uma exceção, um cenário de *exceção* ou *excepcional* é criado. Todos os tratamentos de exceções definidos na especificação dos serviços são mapeados em algum cenário excepcional.

As ações que constituem um cenário são numeradas seqüencialmente. As ações do cenário base são numeradas de 1 a n. As ações dos demais cenários iniciam com o número da ação do cenário (base ou não) que a originou, seguida de uma letra, seguida de um número que indica ordem das ações nesse novo cenário. A letra é usada para indicar mais de um cenário derivado de uma mesma ação. Aninhamentos são possíveis, i.e., criação de um cenário alternativo a partir de outro alternativo, seguindo-se o mesmo princípio de numeração.

Os cenários de exceção seguem as mesmas regras de formação dos cenários alternativos. As regras de formatação para descrição do cenário seguem as definições da metodologia Scent (Ryser e Glinz, 2000).

Todas as entradas, saídas e variáveis operacionais definidas no passo 3 devem ser contempladas em algum cenário ou, eventualmente podem ser excluídas da lista do passo 3. Os casos de uso definidos pela equipe de desenvolvimento, podem ser usados para teste como uma base para comparação com os casos de usos definidos pela equipe de teste, como uma estratégia para aumentar a qualidade do software.

Exemplo

No exemplo do protocolo OBDH-EXP apenas um caso de uso foi suficiente para descrever a função *Verificação do comando recebido*. Esse caso de uso é denominado *Reconhecer um comando* e contém um cenário base, dezenove cenários alternativos e cinco cenários de exceção. A TABELA 4.4 mostra a definição do caso de uso com alguns de seus cenários.

TABELA 4.4 - Caso de uso do protocolo OBDH-EXP.

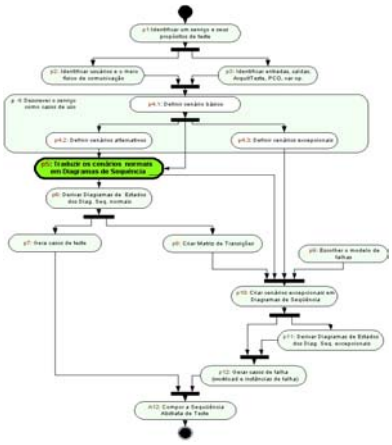
Nome: Reconhecer um comando ***Ator***: OBDH ***Descrição*****:** O SERVIÇO recebe uma mensagem de comando, valida-a e reconhece o comando requisitado. ***Condição de entrada***: ***Condição de saída***: **Cenário Base (1):** 1. o SERVIÇO reconhece os bits de sincronização na mensagem do OBDH, 2. o SERVIÇO reconhece os bits de identificação do experimento na mensagem do OBDH, 3. o SERVIÇO reconhece os bits do comando transmissão de dados (5) na mensagem do OBDH, 4. o SERVIÇO reconhece o byte de checksum na mensagem do OBDH. 5. o SERVIÇO retorna uma indicação de comando recebido ao serviço de execução do EXP

Continua.

TABELA 4.4 - Conclusão.

***Cenários alternativos**:* **Cenário 2:** 3.a.1. o SERVIÇO reconhece o comando Reset (1) na mensagem do OBDH, 3.a.2. retorna ao passo 4 **Cenário 3:** 3.b.1. o SERVIÇO reconhece o comando Clock (2) na mensagem do OBDH, 3.b.2. retorna ao passo 4, ... **Cenário 7 :** 3.c.1. o SERVIÇO reconhece o comando Memory Dump (1A) na mensagem do OBDH, 3.c.2. o SERVIÇO reconhece o tamanho na mensagem do OBDH, 3.c.3. o SERVIÇO reconhece o endereço na mensagem do OBDH, 3.c.4. retorna ao passo 4. ... (20 cenários normais)
***Cenários de exceção**:* **Cenário 21:** 1.a.1. o SERVIÇO não reconhece os bits de sincronização na mensagem do OBDH, 1.a.2. o SERVIÇO não executa ação alguma e continua esperando pelos bits de sincronização, **Cenário 22:** 2.a.1. o SERVIÇO não reconhece os bits de identificação do experimento na mensagem do OBDH 2.a.2. o SERVIÇO retorna ao seu estado inicial e espera por bits de sincronização **Cenário 23:** 3.a.1. o SERVIÇO não reconhece os bits do comando na mensagem do OBDH 3.a.2. o SERVIÇO retorna ao seu estado inicial e espera por bits de sincronização **Cenário 24:** 4.a.1. o SERVIÇO não reconhece o byte de checksum na mensagem do OBDH 4.a.2. o SERVIÇO retorna ao seu estado inicial e espera por bits de sincronização **Cenário 25:** 6.a.1. excede o tempo de espera do SERVIÇO pelos próximos bits da mensagem de comandos do OBDH 6.a.2. o SERVIÇO retorna ao seu estado inicial e espera por bits de sincronização (5 cenários de exceção)

4.4.5 Passo 5: Traduzir os Cenários Normais em Diagramas de Seqüência Normais



No passo 5, as informações do cenário base e dos cenários alternativos são traduzidas em um ou mais diagramas de seqüência, chamado diagrama de seqüência normal. A seqüência ordenada de eventos representada no diagrama de seqüência é também chamada traço de eventos em Rumbaugh et al. (1991). Esse diagrama representa a ordem cronológica de ocorrência das interações em cada entidade.

As entidades representam os usuários e/ou atores, definidos nos passos 2 e 4, que se relacionam com o serviço. Pelo menos três entidades são mapeadas: a UUT, um usuário e o meio de comunicação. Cada entidade é representada em uma coluna. A UUT pode ser composta por vários componentes; nesse caso, cada componente é representado como uma entidade. Algumas entidades são visíveis para os testes e outras não. As entidades não visíveis fazem parte do *contexto* e apresentam comportamento supostamente correto. Definições mais detalhadas sobre teste em contexto e pesquisas relacionadas podem ser encontradas em Petrenko et al. (1996b) e Anido et al. (2003).

As interações correspondem a eventos externos provocados ou sentidos pelas entidades. As interações podem estar associadas a *condições* que, por sua vez, habilitam ou desabilitam o evento e/ou a parâmetros (de uma mensagem ou de uma primitiva). As interações, suas condições e parâmetros são definidos com base na tabela de entradas, saídas e variáveis operacionais (passo 3). Entretanto, a caracterização das entradas ou variáveis operacionais, com seus valores possíveis, em interações é realizada neste passo. A ordem das interações é derivada dos cenários (passo 4). Cada interação é representada em uma linha horizontal. Interações podem representar mensagens/primitivas diferentes ou uma mesma mensagem/primitiva com combinação

distinta de valores de mensagens, ou ainda valores de um campo ou de um parâmetro e não a mensagem como um todo.

O uso de tabela da verdade é indicado para facilitar a identificação de todas as combinações de valores. A caracterização de uma interação depende do nível de abstração do serviço sendo modelado. Cabe ressaltar que apenas as interações observáveis no contexto do teste a ser realizado devem ser evidenciadas.

A importância da tradução dos cenários dos casos de uso em diagramas de seqüência é o estabelecimento da ordenação temporal das interações na colaboração entre entidades externas e a entidade que implementa o serviço em teste, proporcionando uma visão dinâmica do comportamento do serviço dentro dos limites de visibilidade dos testes. A entidade que implementa o serviço em teste é a unidade em teste (UUT).

Se o serviço está relacionado a mais de um propósito de teste, cria-se pelo menos um diagrama de seqüência normal para cada propósito.

Exemplo

As entidades relacionadas à função *verificação de um comando recebido* são: por um lado, o processo aplicativo do OBDH e a linha serial (meio de comunicação), e por outro o serviço de Verificação, o serviço de Execução, o relógio do EXP. A entidade Serviço Verificação, pertencente ao EXP é a unidade em teste. As interações são caracterizadas pelos valores dos campos da mensagem de comando recebida pela função a ser testada no EXP. De fato, esses valores correspondem ao domínio de valores atribuídos às variáveis operacionais. A TABELA 4.5 apresenta tais valores que neste passo vão caracterizar as interações do diagrama de seqüência no exemplo OBDH-EXP; isto é, cada interação recebe o nome igual ao valor da variável operacional que é recebida pelo OBDH-EXP. Na primeira coluna encontram-se os nomes dos campos da mensagem e na segunda os valores e os respectivos significados que são usados para indicar a interação no diagrama de seqüência.

TABELA 4.5 - Caracterização das interações para o protocolo OBDH-EXP.

Nome do campo da mensagem	Valores possíveis do campo
SYNC	EB
EXPID	92
TYPE	1 = reset, 2= clock, 3= inic acquisition, 4 = stop, 5 = datatx, 7= reconf, 1A = mdump, 1B = mload, 1F = load parameter.
SIZE	1, 4, datab
DATA	1, 0, w1w2, adr, 30,31, 32, 33, 3E, 3F, 40, 45, 46, 47, 48
CKSUM	Cs

A ordem dos valores que chegam à função em teste corresponde à sintaxe da mensagem; conseqüentemente, o teste para validação dessa ordem é chamado *teste de sintaxe*. Neste exemplo, não foi necessário associar condições às interações. As interações com o relógio e com o Serviço Execução não são visíveis ao teste. Elas fazem parte do *contexto* do teste, mas foram representadas para indicar o tempo de espera pela próxima mensagem. A entidade "meio de comunicação" é representada para salientar o meio que interfere no fluxo dos bytes entre um sistema e outro; além disso, ele é uma entidade importante para a realização de injeção de falhas a ser tratada no passo 10. É importante destacar que apenas as interações que saem da entidade Processo Aplicativo OBDH são observáveis no teste. Dois cenários são ilustrados no diagrama de seqüência normal na FIGURA 4.8. Os demais cenários são semelhantes, sendo um cenário para cada comando possível. O EXP reconhece 9 comandos (indicados no campo TYPE), sendo que o comando *reconf* pode ter uma variação e o comando *pload* onze. Assim, o número de cenários normais previsto é 20, sendo um para cada tipo de comando.

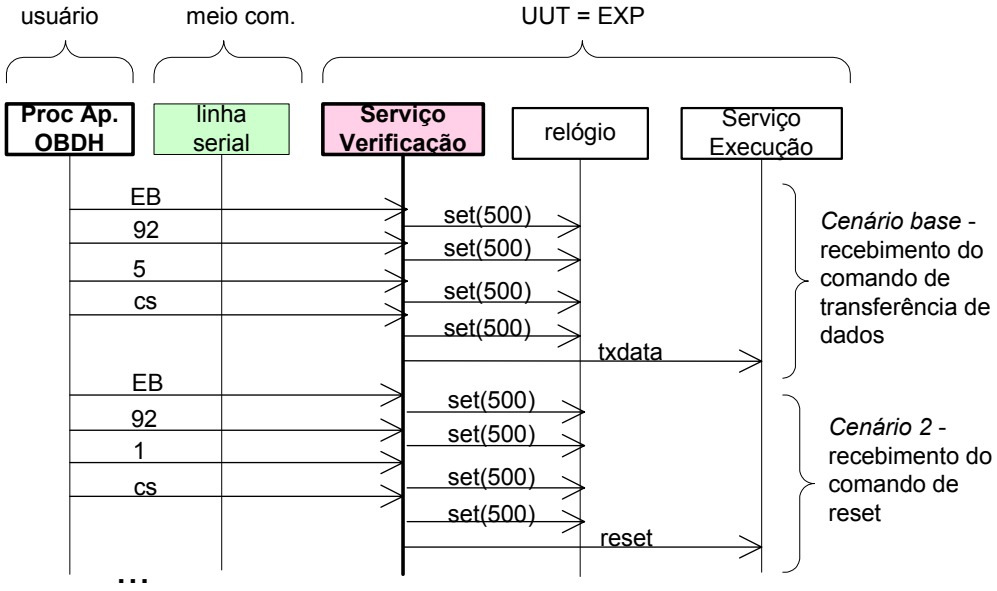
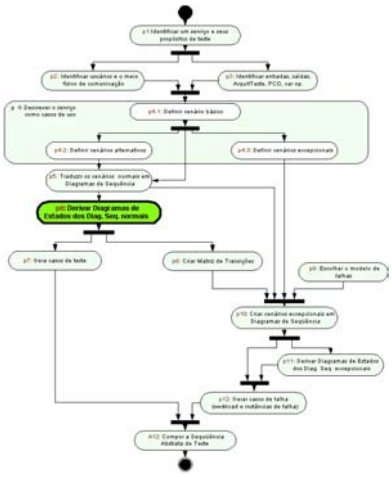


FIGURA 4.8 - Diagrama de Seqüência *Normal* do protocolo OBDH-EXP.

4.4.6 Passo 6: Derivar Diagramas de Estado Normais dos Diagramas de Seqüência Normais



Este passo consiste na geração de um diagrama de estados que represente as informações do diagrama de seqüência normal, relativas à entidade provedora do serviço em teste. Toda interação presente no diagrama de seqüência, que chega à entidade provedora do serviço, é mapeada para o diagrama de estados como um evento de entrada. Os eventos caracterizam uma transição de estado. As interações de saída correspondem às saídas ou ações de uma transição de estado. As *transições* no diagrama de estados são definidas como: *entrada* [*condição*]/ *saída*. As *entradas, saídas* e c*ondição* são compatíveis com aquelas das interações definidas no passo 5. A definição dos estados é inspirada nos intervalos entre as interações de entrada do diagrama de seqüência. A tradução de diagramas de seqüência em diagramas de estados tem sido alvo de

pesquisas, como pode ser encontrado em Rumbaugh et al. (1991, p. 175), Krüger et al. (1999) e Damm e Harel (2001).

O diagrama de estados mostra a seqüência de estados, pelos quais o serviço passa em seu ciclo de vida, bem como os eventos que o fazem mudar de estado e as saídas ou ações produzidas por um evento. Caso exista mais de um propósito de teste, pode-se criar um diagrama de estados normal para cada um deles ou um diagrama único que modele todos os propósitos.

As vantagens da criação do diagrama de estados são: (i) o comportamento desse modelo é previsível; (ii) permite representar controle determinístico de grande quantidade de eventos; (iii) existem ferramentas para simulação, verificação da completeza e consistência do modelo e, principalmente, derivação de casos de teste e; (iv) fundamentada na teoria de autômatos (Binder, 2000).

Exemplo

O diagrama de estados normal traduzido do diagrama de seqüência da FIGURA 4.8 é ilustrado na FIGURA 4.9. As interações entre o *serviço Verificação* e o relógio não foram representadas, pois não são visíveis ao teste. As interações entre o *serviço Verificação* (o serviço alvo dos testes) e o *serviço Execução* do comando são traduzidas para o diagrama de estados. Cada interação de entrada do diagrama de seqüência (EB, 92, 5, cs, 1, etc) é uma entrada em alguma transição de estado. Cada entrada deve disparar uma resposta. As respostas serão enviadas ao OBDH via o serviço *Execução do comando,* de acordo com a saída indicada pelo *serviço Verificação*. As saídas *set(500)* são geradas pelo *serviço Verificação* para o relógio. *Time-outs* serão tratados no passo 10.

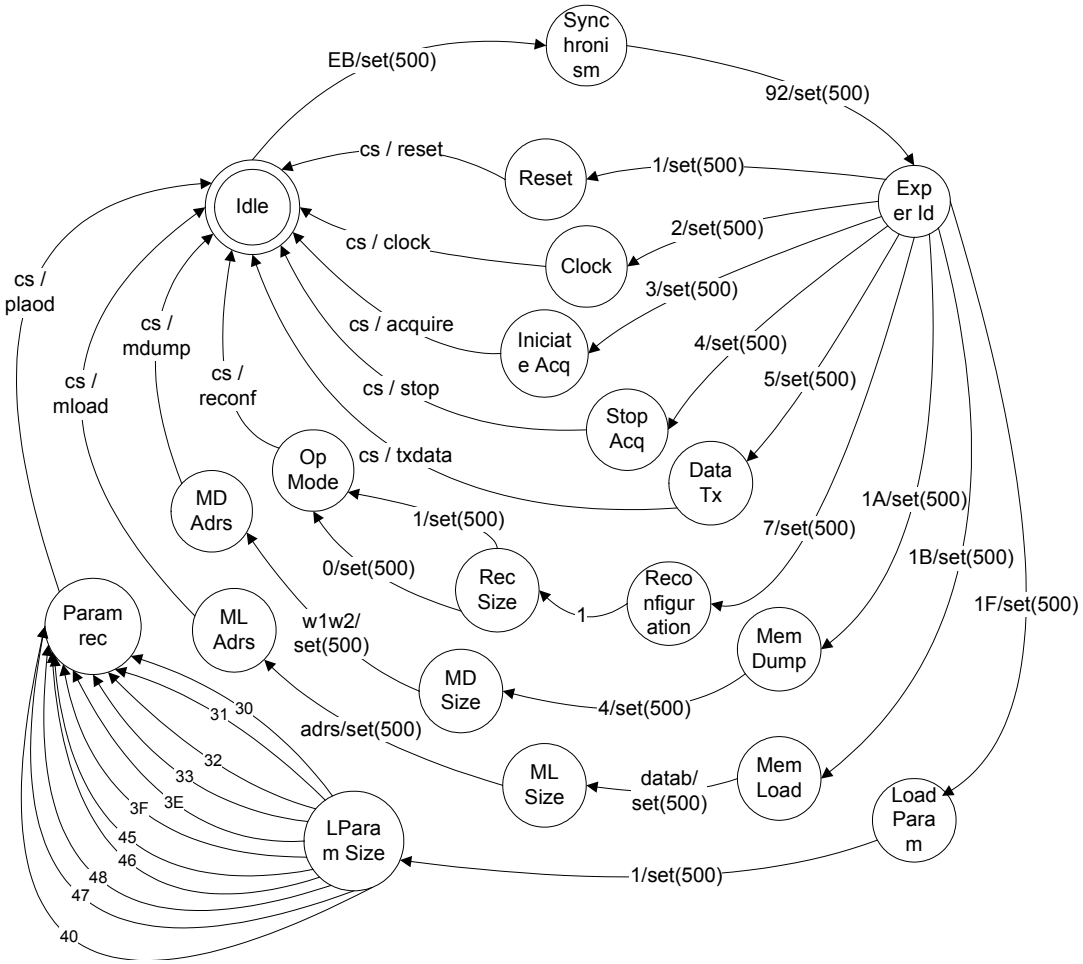
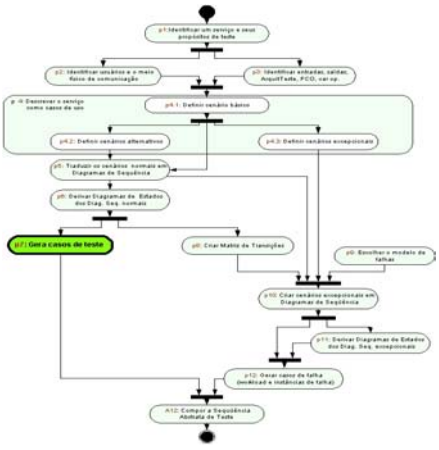


FIGURA 4.9 - Diagrama de Estados *Normal* do protocolo OBDH-EXP.

4.4.7 Passo 7: Gerar <u>casos de teste</u>



Uma vez que o comportamento normal do serviço em teste encontra-se representado em um diagrama de estados, esse diagrama é submetido a uma ferramenta capaz de gerar automaticamente uma seqüência de *casos de teste,* também chamados *casos de teste de conformidade*. Caso haja mais de um diagrama, cada um deles é submetido à ferramenta de teste de forma que o conjunto dos casos de teste total é a união de todas as

seqüências geradas.

A metodologia $CoFI_m$ conta com a ferramenta Condado (Sabião, 1998 ; Martins at al., 1999), mas não se limita a ela. A Condado requer que a máquina de estados seja mínima (para evitar a geração de testes redundantes), de Mealy (defina entradas e saídas nas transições), inicialmente conexa (tenha sempre um caminho a partir do estado inicial para alcançar os demais estados), o estado inicial seja indicado na descrição da máquina. Essa ferramenta implementa o algoritmo de busca em grafo, denominado método *switch-cover* (Chow, 1978), o qual realiza intercalação exaustiva entre todas as transições da máquina, de forma que os casos de teste dão uma cobertura exaustiva da especificação. Segundo Binder (2000), essa técnica permite, se todos os casos forem aplicados, revelar estados inválidos e as falhas em transições. Na Condado, uma transição é um par $<i_1, o_1>$, de uma entrada e uma saída. A seqüência de teste produzida consta do conjunto = $\{(<i_1, o_1>, <i_2, o_2>, ...<i_n, o_n>), (<i_1, o_1>, <i_4, o_4>, ...<i_m, o_m>), ...\}$, onde, cada elemento desse conjunto representa um caminho completo na máquina, o qual começa em um nó inicial da máquina e termina em um nó final (que pode ser o próprio nó inicial caso a máquina não tenho o nó final). Cada caminho completo livre de laço constitui um *caso de teste*.

Seqüências de teste geradas com base nos métodos citados no Capítulo 2, que se baseiam em algoritmos de busca em grafos, podem detectar erros do tipo**:** (i) transições incorretas ou ausentes (falha de transferência); (ii) saídas incorretas ou não especificadas (falha de saída); (iii) estados incorretos e; (iv) eventos corretos que produzem saídas esperadas, em momentos errados. As referências Bochmann et al. (1992) ; Dssouli et al. (1999); Binder (2000), para citar alguns, discutem os erros estruturais em máquinas de estados. A ferramenta Condado não garante cobertura de todas as falhas estruturais da especificação, no entanto, garante cobertura de caminhos da especificação do comportamento.

Exemplo

A máquina apresentada na FIGURA 4.9, satisfaz os requisitos exigidos pela Condado e quando submetida a ela, derivou 20 casos de teste. A TABELA 4.6 ilustra dois casos de

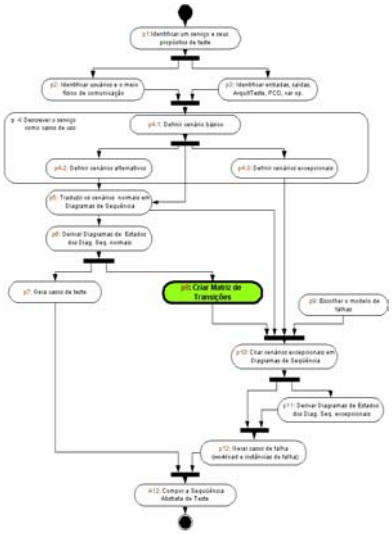
teste para o serviço de verificação do comando no formato gerado atualmente pela Condado. Os casos de teste são escritos em uma notação que facilita a transformação da seqüência de teste em uma notação executável, mas ainda genérica o suficiente para ser abstrata. Ela pressupõe a existência de primitivas de envio (*senddata)* e recepção (*recdata)* de mensagens pelo meio de comunicação. As letras L e T vêm do conceito dos métodos de teste de conformidade da IS-9646, que correspondem aos canais com o testador inferior (*lower*) e o testador superior (*upper*), respectivamente. No caso de aplicações espaciais, esses canais não são, necessariamente, superior e inferior (Ambrosio et al., 2004a), eles são referenciados aqui simplesmente por PCOs. Entretanto, a saída da Condado mantém a terminologia de canal superior e inferior (U e L). O conjunto de casos de teste derivado pela ferramenta Condado é dado pelo conjunto a seguir:

{(<EB, >, <92, >, <5, >, <cs, txdata>), (<EB, >, <92, >, <1, >, <cs, reset>), (<EB, >, <92, >, <2, >, <cs, clock>), (<EB, >, <92, >, <3, >, <cs, acquire>), (<EB, >, <92, >, <4, >, <cs, stop>), (<EB, >, <92, >, <7, >, <1, >, <1, >, cs, reconf>),.(<EB, >, <92, >, <7, >, <1, >, <0, >, cs, reconf>), (<EB, >, <92, >, <1A, >, <4, >, <w1w2, >, <cs, mdump>), (<EB, >, <92, >, <1B, >, <datab, >, <adrs, >, <cs, mload>), (<EB, >, <92, >, <1F, >, <1, >, <30, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <31, >, <cs, pload>),..(<EB, >, <92, >, <1F, >, <1, >, <32, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <33, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <3E, >, <cs, pload>), (<EB, >, <92, >, <1F, >, <1, >, <3F, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <40, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <45, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <46, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <47, >, <cs, pload>),.(<EB, >, <92, >, <1F, >, <1, >, <48, >, <cs, pload>),...}.

TABELA 4.6 - Casos de teste de conformidade para o OBDH-EXP.

senddata(L, EB) recdata(U, set(500)) senddata(L, 92) recdata(U, set(500)) senddata(L, 05) recdata(U, set(500)) senddata(L, cs) recdata(U, txdata)
senddata(L, EB) recdata(U, set(500)) senddata(L, 92) recdata(U, set(500)) senddata(L, 01) recdata(U, set(500)) senddata(L, cs) recdata(U, reset) .....

4.4.8 Passo 8: Criar a Matriz de Transições



Este passo consiste de dois subpassos, denominados p8.1 e p8.2, que incluem as tarefas de:

- p8.1 - transcrever a máquina de estados gerada no passo 6 para uma matriz de transições e,
- p8.2 - estabelecer a suposição de completeza da máquina de estados, isso significa definir uma saída e o próximo estado para todas as entradas contempladas no diagrama de estados do passo 6.

Na matriz, cada linha representa um evento e cada coluna um estado. Cada elemento resultante da intersecção evento-estado contém uma saída (ou ação) esperada normal explicitamente-especificada, mais o estado destino. A matriz resultante da transcrição é esparsa. Sob o ponto de vista da conformidade com a especificação, quaisquer saídas nos elementos vazios representam não conformidade (saídas não especificadas). Contudo, as transições não especificadas não necessariamente são ilegais.

A *suposição de completeza* consiste em preencher os elementos vazios da matriz, com objetivo de identificar cenários de exceção. Em Binder (2000, p. 225), a *suposição de completeza* é realizada em uma tabela chamada Matriz Resposta que acrescenta uma tabela da verdade. Nesta tabela verdade, os eventos e as expressões que caracterizam as

condições para sua ocorrência denominam as linhas e os estados caracterizam as colunas. Na intersecção das linhas e das colunas estão as ações que podem ser acionadas. Essas ações são indicadas por: um código de uma ação de exceção, uma indicação de que nenhuma ação deve ser aplicada, uma indicação de que os valores dos eventos são mutuamente exclusivos e portanto, nenhuma ação é associada, ou, uma indicação da ação explicitamente especificada. Esses elementos são referenciados no texto como (*evento$_c$, estado$_c$*). Ao atribuir-se uma saída aos elementos vazios (*evento$_c$, estado$_c$*) da matriz, o testador depara-se com transições:

- legais, indefinidas nessa fase por dependerem de informações da implementação,
- nulas, ilegais, representam situações mutuamente exclusivas,
- desconhecidas; neste caso, a UUT pode ter uma das seguintes saídas: (i) um término anormal: *crash* ou aborto; (ii) nenhuma aparente, isto é, o evento é ignorado e o estado mantido; (iii) uma exceção é disparada, e; (iv) inconsistente, o evento é aceito e uma saída produzida (Cristian, 1991).

Os cenários originados deste passo cobrem a ocorrência de eventos corretos em estados errôneos[5] (ver Seção 4.2). Caso exista mais de um propósito de teste e, conseqüentemente, mais de um diagrama de estados, deve-se criar uma matriz para cada diagrama.

Exemplo

A matriz de transições completa é apresentada na TABELA 4.7. Por simplificação dessa tabela, nomes dos estados, dos eventos e das ações são substituídos por mnemônicos listados na TABELA 4.8. Os elementos (*evento$_c$, estado$_c$*) em cinza, completados nesse passo, totalizam (16 + 16 + 8 + (6x 16) + 15 + (10 x 16) = 311.

[5] Binder (2000) define caminhos furtivos *(sneak paths*) como sendo a aceitação de um evento incorreto por algum estado ou de um evento correto por um estado errôneo.

TABELA 4.7 - Matriz de transição completa do protocolo OBDH-EXP.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
EB	p/2	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
92	d/1	p/3	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
01	d/1	d/1	p/4	d/1	d/1	d/1	d/1	d/1	p/ 10	p/ 11	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 19	d/1	d/1
02	d/1	d/1	p/5	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
03	d/1	d/1	p/6	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
04	d/1	d/1	p/7	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 13	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
05	d/1	d/1	p/8	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
07	d/1	d/1	p/9	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
1A	d/1	d/1	p/ 12	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
1B	d/1	d/1	p/ 15	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
1F	d/1	d/1	p/ 18	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
cs	d/1	d/1	d/1	r/1	r/1	r/1	r/1	r/1	d/1	d/1	r/1	d/1	d/1	r/1	d/1	d/1	r/1	d/1	d/1	r/1
00	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 11	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1
dt	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 16	d/1	d/1	d/1	d/1	d/1
w w	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 14	d/1	d/1	d/1	d/1	d/1	d/1	d/1
ad	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 17	d/1	d/1	d/1	d/1
pr	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	d/1	p/ 20	d/1

TABELA 4.8 - Descrição dos estados, eventos e saídas da Matriz Completa.

Estados (Número e Nome)					
1	Idle	8	Data tx	15	mem load
2	Synchronism	9	Reconfiguration	16	ml size
3	Exper id	10	Rec size	17	ml adrs
4	Reset	11	op mode	18	load param
5	Clock	12	mem dump	19	lparam size
6	Iniciat & Acq	13	md size	20	param rec
7	Stop Acq	14	md adrs		
Saídas ou ações (código e descrição)					
p	ler próximo byte e disparar temporizador				
d	descartar os bits recebidos e re-inicializar variáveis				
r	Indicar recepção correta e o comando				
Eventos (código e descrição)					
cs	Valor correto de *checksum* – exige cálculo e comparação com valor recebido				
pr	Parâmetro do comando *load param*. Pode conter: 30,31,32,33,3E,3F,40,45,46,47,48.				

4.4.9 Passo 9: Identificar o Modelo de Falhas

Este passo consiste em identificar o modelo de falhas que representa as conseqüências que anomalias do ambiente podem causar no sistema em teste. O modelo consta de uma suposição sobre os tipos de falhas que podem acontecer, ele pode ser baseado na experiência, na desconfiança, na análise, na experimentação (Binder, 2000). O objetivo deste passo é apoiar a geração de cenários de exceção que cubram os mecanismos de tolerância a falhas (a serem definidos no passo 10.3).

No contexto deste trabalho, foi usado o modelo de falhas de comunicação, definido por Cristian (1991) e Holzmann (1991), para manter compatibilidade com as ferramentas existentes no projeto ATIFS. Esse modelo inclui falhas de perda, atraso, duplicação e corrupção de mensagens que podem ser provocadas pelo meio de comunicação durante a operação (execução) do serviço.

A caracterização do modelo de falhas de comunicação requer a definição dos tipos exatos de falhas que o meio de comunicação físico pode provocar, bem como a capacidade dos injetores de falhas da arquitetura de teste, a *Ferry-injection* apresentada na Seção 3.4.2.

Visando facilitar a atividade de execução de teste foram definidas aqui, as *primitivas de falhas* apresentadas na TABELA 4.9. Essa solução segue a filosofia da ferramenta Condado no uso das primitivas *senddata* e *recdata*.

As primitivas de falhas são usadas na definição dos cenários de exceção no passo 10.3. Elas tornam explícita a relação entre o projeto dos casos de falha e a execução dos casos de falha com o apoio dos injetores de falha. Na tabela são indicados parâmetros requeridos em cada primitiva. O PCO pode também ser um parâmetro das primitivas de falhas.

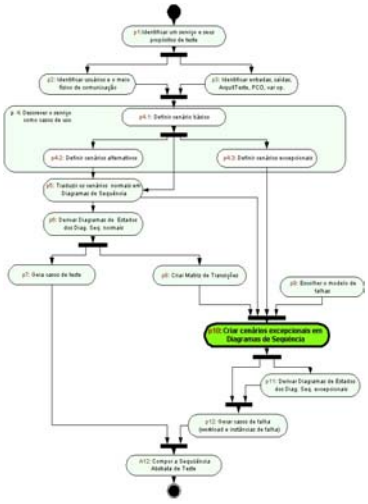
TABELA 4.9 - Primitivas de falhas de comunicação.

Falha	Mnemônico	Primitiva de falha	Descrição	Parâmetros
corrupção	f_1	f.corr	Valores em campos da mensagem são adulterados	Id da mensagem; Valor novo corrompido ou símbolo ($\neq$) para alterar o corrente para um qualquer
atraso	f_2	f.atraso	A mensagem é entregue após o tempo máximo de espera	Id da mensagem, tempo de atraso
duplicação	f_3	f.duplic	Duas mensagens iguais são entregues	Id da mensagem ,
perda	f_4	f.perda	A mensagem não é entregue	Id da mensagem,

Exemplo

O meio físico de comunicação entre o OBDH e o EXP é uma linha serial, a RS422. Para essa linha, falhas Bizantinas (arbitrárias) podem ocorrer na presença de radiação, portanto, os comandos podem ser corrompidos, perdidos, duplicados ou atrasados. Para a validação da função Verificação do comando recebido, implementada no EXP, o modelo de falhas consiste de: perda, atraso, duplicação e corrupção de bytes transmitidos do OBDH ao EXP. Para a validação da implementação do EXP, uma falha de corrupção a um byte significa que ele recebe, por exemplo, um 91h ao invés do 92h. O mapeamento de falhas de comunicação, nesse contexto, pode representar ou falhas no meio de comunicação ou falhas de mau funcionamento do próprio OBDH.

4.4.10 Passo 10: Criar Cenários Excepcionais em Diagramas de Seqüência Excepcionais



Este passo consiste em criar cenários de exceção em diagramas de seqüência, chamados *diagramas de seqüência excepcionais*. Os diagramas de seqüência excepcionais devem conter as entidades definidas no passo 5, quais sejam: a UUT, os usuários e o meio de comunicação; mais os injetores de falhas da arquitetura *Ferry-injection*, o FIC e o FIM (ver Seção 3.4.2). Os injetores apóiam a execução das primitivas de falha a serem aplicadas para caracterização dos experimentos de injeção de falhas caracterizados nos cenários excepcionais.

Os cenários excepcionais são descritos com seqüências de interações que chegam e saem da UUT, bem como, por interações trocadas entre os injetores de falha (FIC e FIM). Essas últimas são chamadas *interações de falha*. A definição das interações deve ser de acordo com o tipo do cenário excepcional que se pretende mapear. Há três tipos de cenários excepcionais: exceções explicitamente especificadas, exceções por caminhos furtivos e exceções para robustez (os softwares que vão a bordo de satélite, implementam mecanismos de tolerância a falhas para prover a robustez ao software. Por esse motivo, esses cenários são referenciados aqui como cenários de tolerância a falhas ou TF). A criação dos cenários excepcionais é descrita nos três subpassos 10.1, 10.2 e 10.3, os quais são mutuamente exclusivos e independentes.

Passo 10.1 - Gerar os Cenários de Exceção a Partir dos Cenários Excepcionais dos Casos de Uso:

Este passo consiste em traduzir os cenários de exceção definidos no passo 4.3 em um diagrama de seqüência; cobrindo os eventos de exceção explicitamente especificados que deixaram de ser mapeados no diagrama de seqüência normal (passo 5). O

procedimento para definição deste tipo de cenário consiste em tomar todos os cenários de exceção especificados no passo 4.3 e para cada um deles:

- definir as entradas para o serviço em teste que traduzem o cenário,
- se necessário, escolher uma primitiva de falha que altere uma entrada normal para caracterizar o cenário de exceção;
- definir as saídas esperadas correspondentes a cada entrada, inclusive após as falhas, se existirem.

A estimativa do número de cenários de falha nessa etapa é de um cenário no diagrama de seqüência excepcional para cada cenário de exceção dos casos de uso criados no passo 4.3.

***Passo 10.2 - Gerar os Cenários de Exceção a Partir da Completeza da Matriz de Transição**:*

Este subpasso consiste em identificar os cenários em que eventos corretos ocorrem em estados errôneos (ver Seção 4.2). A representação desses eventos na máquina de estados é chamada, em Binder (2000) de caminhos furtivos (*sneak paths)*. Para identificar os cenários de exceção de caminhos furtivos, deve-se analisar os elementos da matriz de transição (*evento$_c$, estado$_c$*), inseridos quando realizada a suposição de completeza realizada no passo 8.2.

O procedimento para definição dos cenários de caminhos furtivos consiste em tomar cada elemento (*evento$_c$, estado$_c$*), observar, caso-a-caso e decidir se é possível criar um cenário de exceção que contemple o elemento. Se possível, criar um cenário que o exercite. A primitiva de falha corrupção, em geral, é necessária e conveniente na formação desse tipo de cenário. Para cada elemento que pode ser exercitado, o cenário de exceção é criado da seguinte forma:

- definir uma seqüência de eventos, que parte do estado inicial e vai até o estado "*estado$_c$*",

- incluir o evento "$evento_c$" à seqüência.

- definir as saídas esperadas, para cada evento da seqüência de eventos,

- desenhar o cenário no diagrama de seqüência.

Para alguns elementos ($evento_c$, $estado_c$) não é possível identificar um cenário de exceção, além disso, a representação de todos cenários é uma tarefa cansativa e repetitiva, pois a única diferença entre muitos deles é apenas uma interação. Outro fator negativo dos cenários de caminhos furtivos é o grande número de cenários. Uma forma de contornar esse último ponto é usar o mnemônico <DIF, *ev*> para indicar todos os eventos diferentes de *ev* e, futuramente, criar uma ferramenta que apóie a realização deste passo.

***Passo 10.3 - Gerar os Cenários de Exceção Usando o Modelo de Falhas**:*

Este subpasso consiste em associar as primitivas do modelo de falha aos cenários normais criados no passo 5 (Seção 4.4.5). Mais especificamente, a cada entrada de um cenário normal, aplica-se uma falha do modelo de falhas para originar um novo cenário excepcional. Neste subpasso identificam-se os cenários próprios para validar a robustez da UUT contra os erros externos à implementação. Verifica-se se entradas incorretas são aceitas pela UUT. Essa etapa agrega valor ao teste alertando o testador para situações errôneas advindas do ambiente, do contexto, enfim, externas à implementação em teste.

O procedimento para criação desses cenários consiste em:

- selecionar um cenário normal definido no passo 5 e gerar novos cenários de exceção da seguinte forma:

    a) obter uma das entradas que interagem com a UUT e combinar um tipo de falha,

    b) desenhar uma interação de falha, entre os injetores, identificando a falha a ser aplicada e explicitando a primitiva de falha correspondente,

c) identificar qual a reação esperada do serviço sob essa falha e desenhar as interações de saída, assim encerrando um cenário,

d) aplicar o mesmo tipo de falha para todas as entradas do cenário normal (evitar repetições), repetindo os passos a, b e c para obter novos cenários,

e) repetir os passos a,b,c,d para todas as falhas do modelo de falhas definido no passo 9,

- repetir o procedimento para todos os cenários do passo 5.

A indicação da primitiva de falha nos diagramas de seqüência excepcionais será utilizada quando da geração de experimentos de injeção de falhas. O procedimento para criação de cenários de exceção não se limita ao modelo de falhas de comunicação. A mesma filosofia pode ser seguida se outros modelos de falhas forem utilizados. Alguns modelos de falha usados para validar software em aplicações espaciais são modelo de falhas de memória e o modelo de falhas de processadores.

A estimativa do número de cenários de falha nessa etapa é no máximo o produto de

$$primitivas\_de\_falhas \times \left(\sum cenários\_normais \times entradas\_distinas\right).$$

Os cenários criados conforme descrição deste passo são chamados de cenários de tolerância a falhas ou simplesmente TF.

Exemplo

Passo 10.1: o diagrama de seqüência de exceções especificadas, mostrado na FIGURA 4.10, representa os cinco cenários de exceção definidos no passo 4.3. As primitivas de falhas usadas foram de corrupção e atraso.

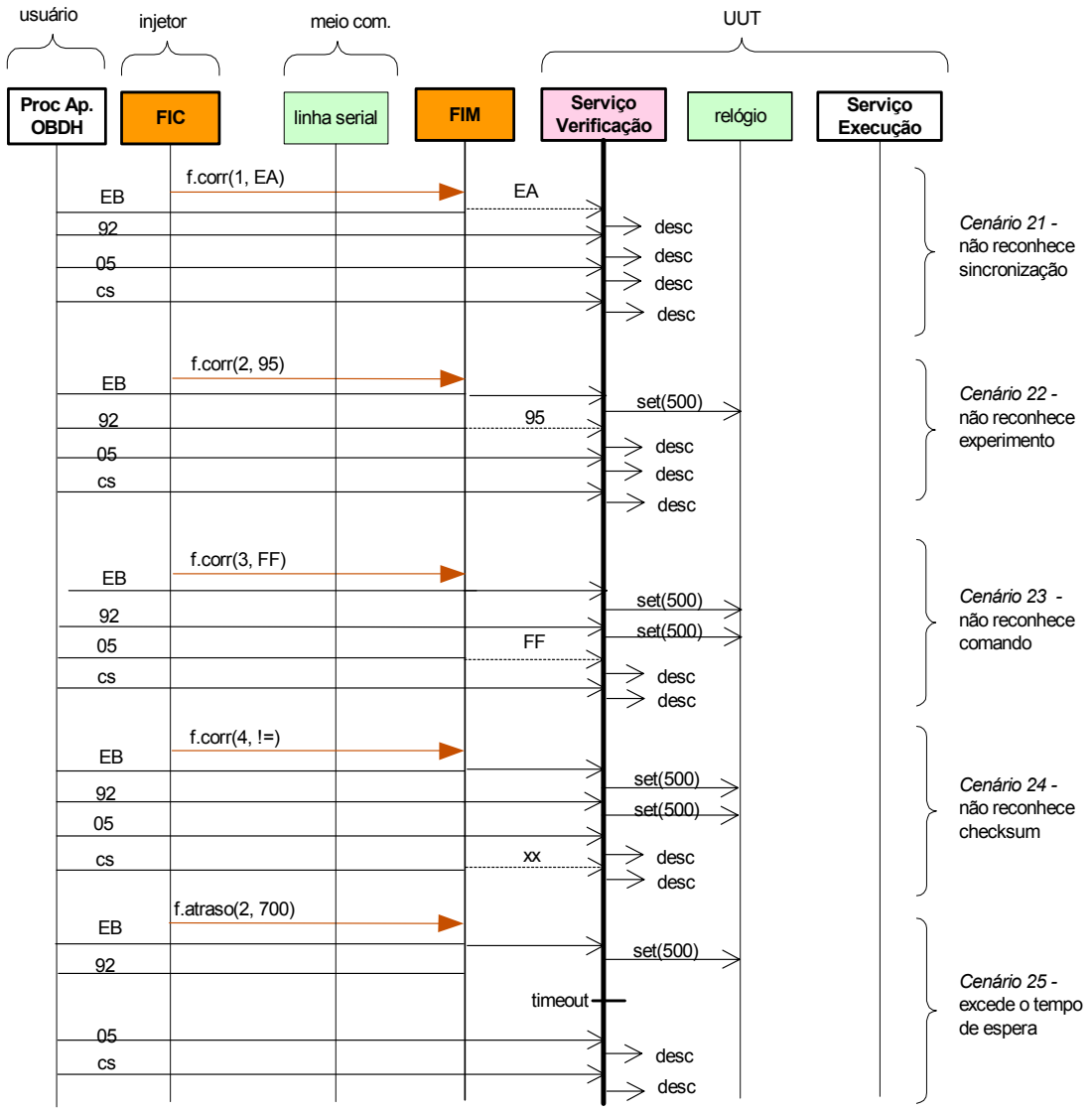


FIGURA 4.10 - Diagrama de Seqüência de *Exceções Especificadas* do OBDH-EXP.

Passo 10.2: A FIGURA 4.11 ilustra o diagrama de seqüência para alguns cenários de exceção de caminhos furtivos, mapeados a partir da matriz de transições mostrada na TABELA 4.7. Por exemplo, o cenário furtivo 17 é criado da seguinte forma: o elemento da matriz a ser exercitado é (EB,2); partindo-se do estado inicial, para se chegar no estado 2 é requerida a entrada EB. Se a entrada 92 for corrompida, então se chega no elemento (EB,2). Quanto ao número de cenários a serem criados, sabendo-se que a

Matriz de Transições tem (17 x 20) 340 elementos, sendo que 29 deles já foram contemplados nos cenários normais, restam 311 para serem contemplados nos cenários de exceção, se nenhuma estratégia de minimização for aplicada.

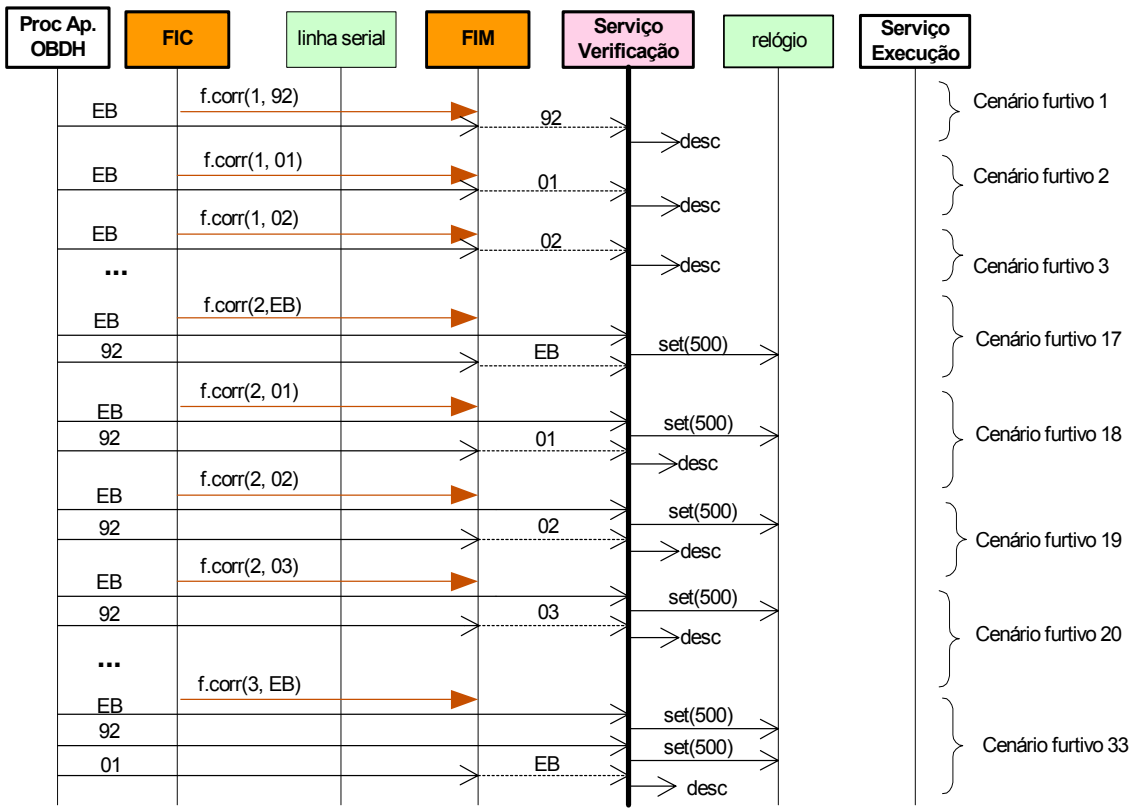


FIGURA 4.11 - Diagrama de Seqüência *Exceções Furtiv*as OBDH-EXP.

Ao analisar os cenários resultantes do mapeamento dos elementos ($evento_c$, $estado_c$) podemos observar que o cenário 21 (ver FIGURA 4.10) sintetiza os cenários furtivos de 1 a 16, o cenário 22 resume os cenários furtivos de 17 a 32, também os cenários 23 e 24 têm suas correspondências nos cenários furtivos.

Passo 10.3: a FIGURA 4.12 ilustra alguns cenários no diagrama de seqüência de *exceção de tolerância a falhas.* Estes cenários originaram-se do primeiro cenário normal definido no passo 5 através da aplicação das primitivas de falha f.atrasa(1,800), f.atrasa(2,800), f.perde(1), f.duplica(3) a segunda, primeira e terceira entradas,

respectivamente. O cenário com a primitiva de corrupção é ilustrado aqui para evidenciar a duplicação.

O conjunto completo das primitivas a serem aplicadas no primeiro cenário normal é mostrado na TABELA 4.10. Observa-se nos cenários excepcionais criados aqui, aqueles originários de falha de corrupção já foram cobertos no passo 10.2 para todas as entradas possíveis, portanto, eles podem ser excluídos. No cômputo geral, o número total de cenários de tolerância a falhas (TF) para o exemplo seria de 4 x (5x4 + 1x7 + 13x6) = 420. Excluindo-se a primitiva de falha de corrupção tem-se 3 x (5x4 + 1x7 + 13x6) = 315. Outra redução ainda pode ser feita nesse exemplo: dado que as duas primeiras entradas, EB e 92, são repetidas, nessa ordem, em todos os cenários normais, torna-se redundante aplicar as falhas a essas entradas nos demais cenários. Assim, o número de cenários será 3 x (1x4 + 4x2 + 1x5 + 13x4) = 207.

TABELA 4.10 - Primitivas de falhas para o primeiro *cenário normal* OBDH-EXP.

f.corr(1,!EB)	f.corr(2,!92)	f.corr(3,!05)	f.corr(4,!cs)
f.atrasa(1,800)	f.atrasa(2,800)	f.atrasa(3,800)	f.atrasa(4,800)
f.perde(1)	f.perde(2)	f.perde(3)	f.perde(4)
f.duplica(1)	f.duplica(2)	f.duplica(3)	f.duplica(4)

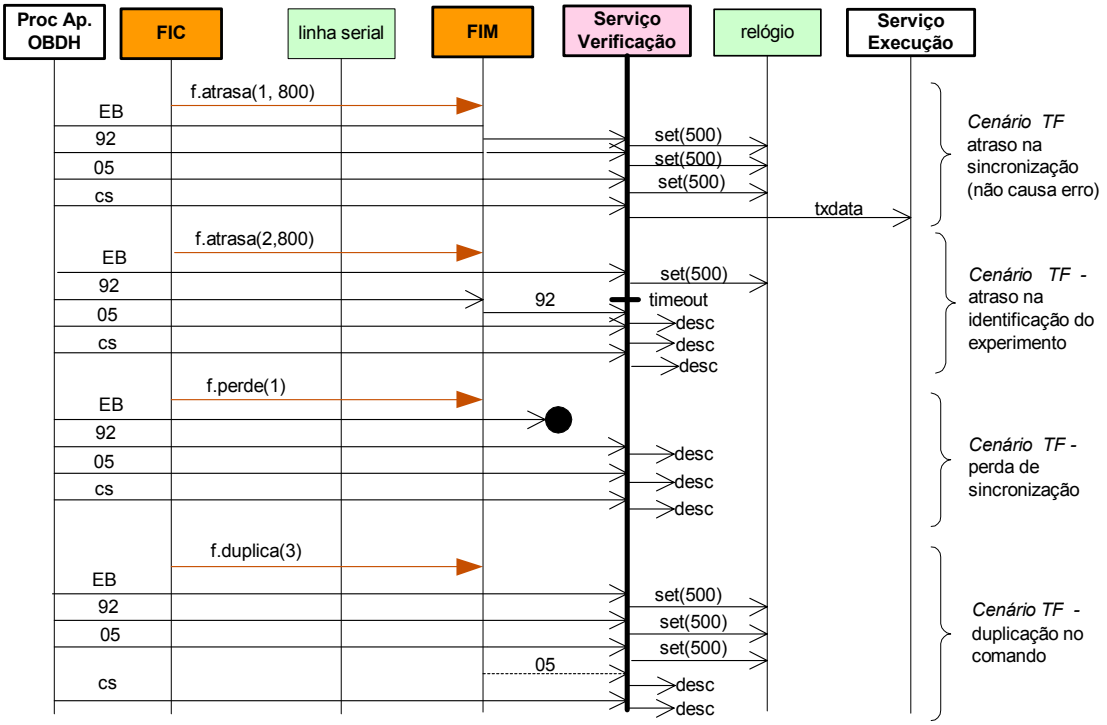
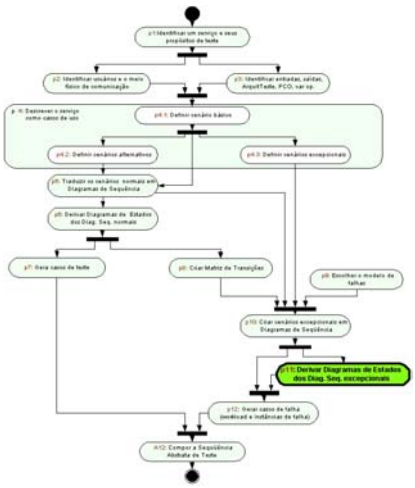


FIGURA 4.12 - Diagrama de Seqüência *Exceções de TF* OBDH-EXP.

4.4.11 Passo 11: Traduzir os Diagramas de Seqüência Excepcionais em Diagramas de Estados Excepcionais



Este passo consiste em traduzir os diagramas de seqüência excepcionais em diagramas de estados eqüivalentes, utilizando os mesmos procedimentos aplicados no passo 6, exceto para a interação de falha e para a forma de representar as transições. Uma transição no diagrama de estados excepcional é dada como $<i_1, o_1, f_1>$, onde i é uma entrada normal, o uma saída e, f a indicação de um tipo de falha relacionada à entrada. O símbolo f_0 é usado para indicar nenhuma falha.

A interação de falha não é diretamente representada no diagrama de estados. A falha representada na interação de falha do diagrama de seqüência é traduzida para uma entrada ou evento no diagrama de estado seguida de uma indicação do tipo da falha. A

indicação pode ser, <f_1>, <f_2>, <f_3>, <f_4>, respectivamente para corrupção, atraso, duplicação e perda (ver TABELA 4.9).

Diagramas de estados são criados para os cenários de exceções especificadas, para os cenários de caminhos furtivos e para os cenários de tolerância a falhas. A máquina de estados de exceções por caminhos furtivos possui, geralmente, muitas transições repetitivas, para simplificar pode-se fazer uso de transições com a marca <DIF, *ev*>, para indicar todos os eventos diferentes de *ev como seguido em* (Stefani, 1997).

Exemplo

Os diagramas de estados excepcionais equivalentes aos diagramas de seqüência gerados no passo 10 são apresentados, respectivamente, nas figuras FIGURA 4.13, FIGURA 4.14, FIGURA 4.15 e FIGURA 4.16, para exceções especificadas, exceções de caminhos furtivos, exceções de tolerância a falhas de atraso e perda, e exceções de tolerância a falhas de duplicação. Para este exemplo, o comportamento da aplicação é o mesmo para as falhas de atraso de perda; ambos são representados em um mesmo diagrama. Alguns estados e transições foram omitidos para simplificação visual, mas a filosofia de eventos é semelhante à dos demais eventos.

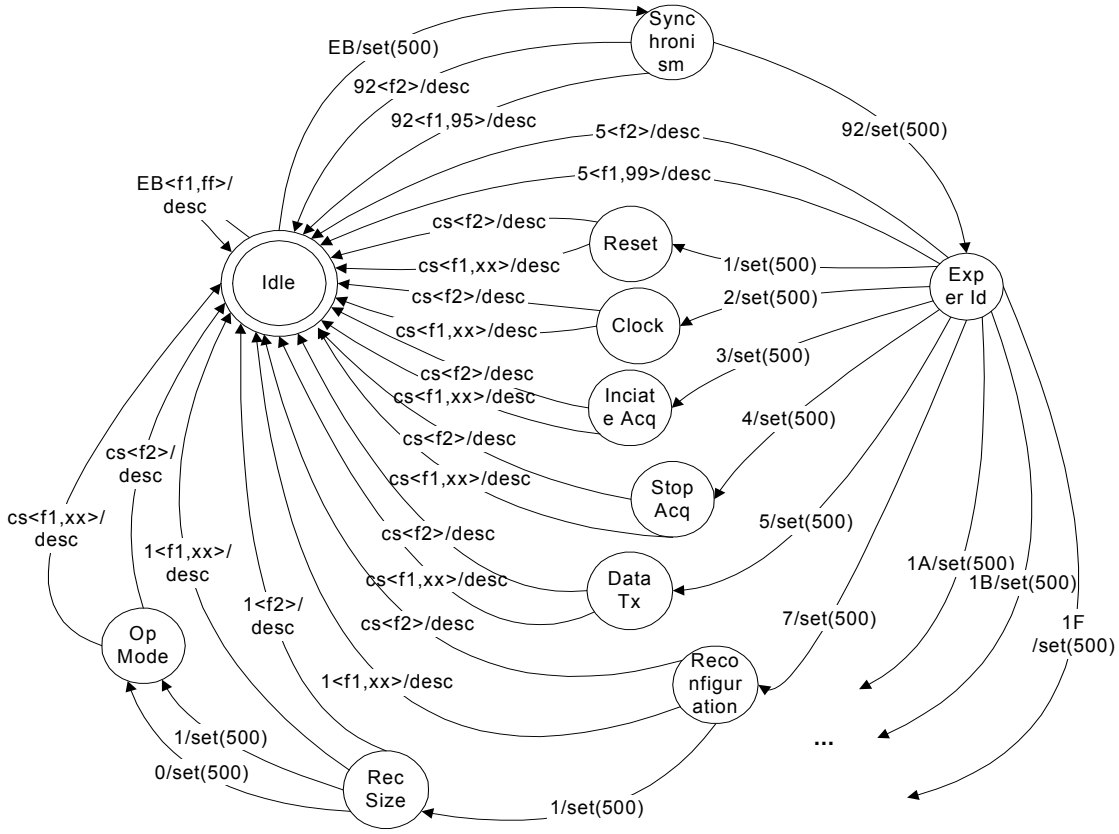


FIGURA 4.13 - Diagrama de Estados *Exceções Especificadas* OBDH-EXP.

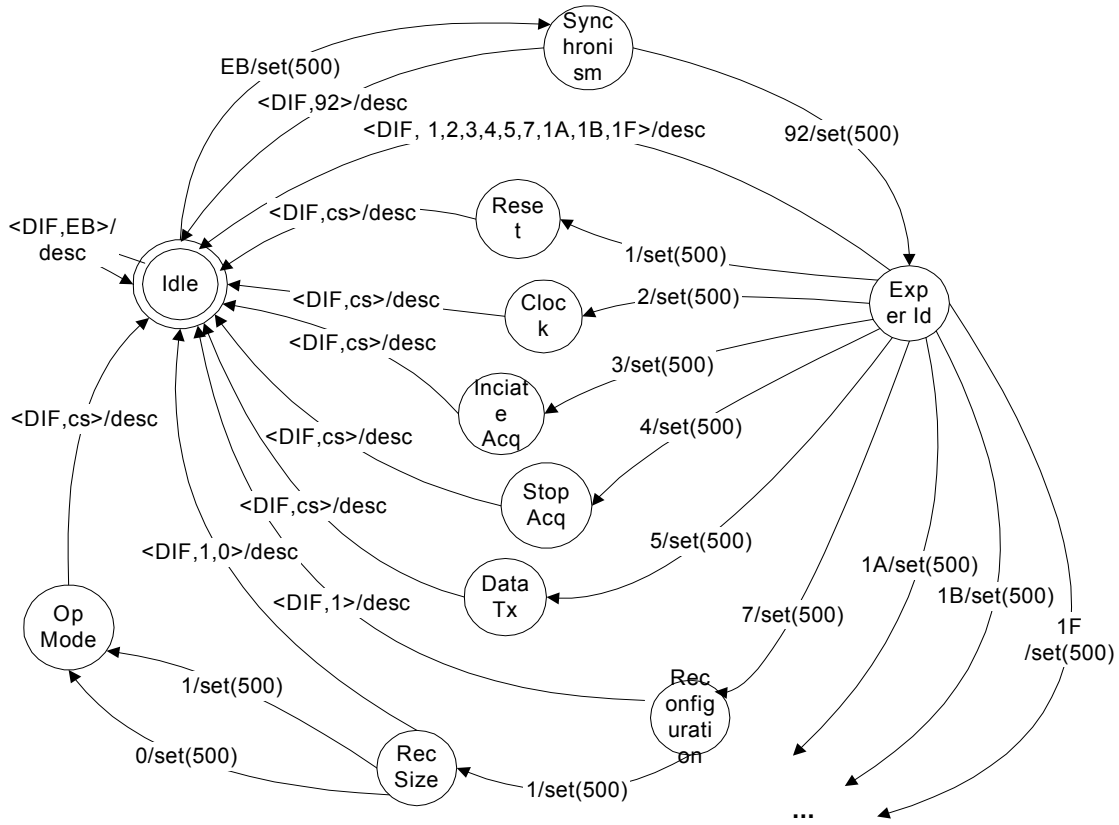


FIGURA 4.14 - Diagrama de Estados *Exceções por caminhos Furtivos* OBDH-EXP.

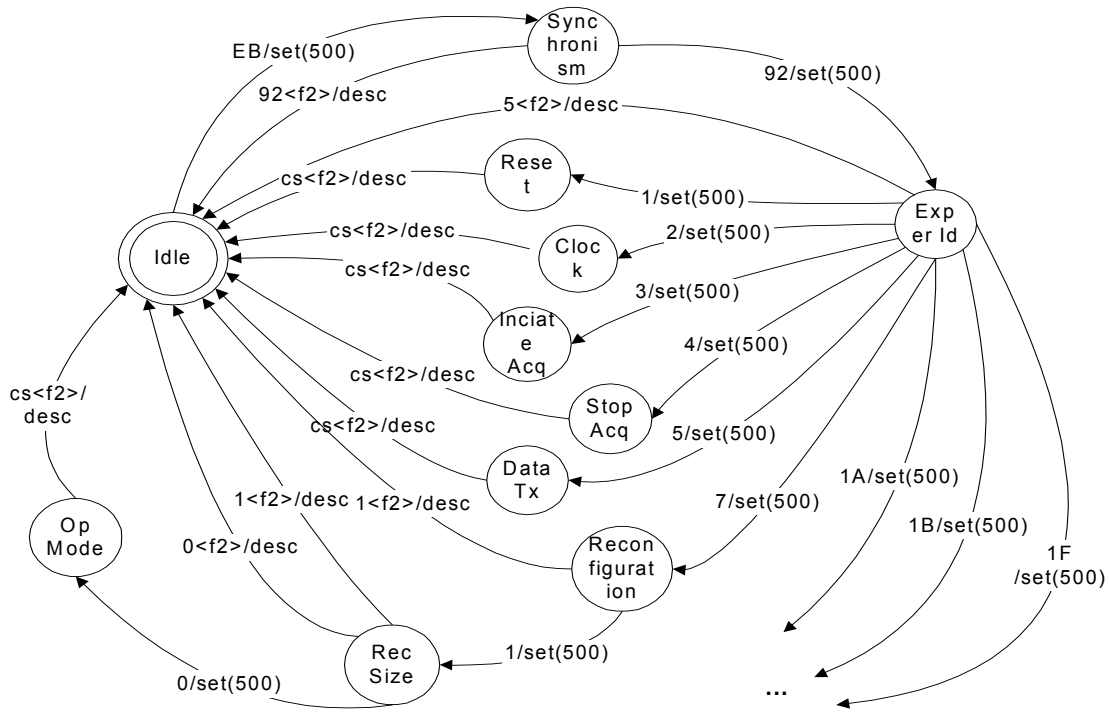


FIGURA 4.15 - Diagrama de Estados *Exceções TF atraso e perda* OBDH-EXP.

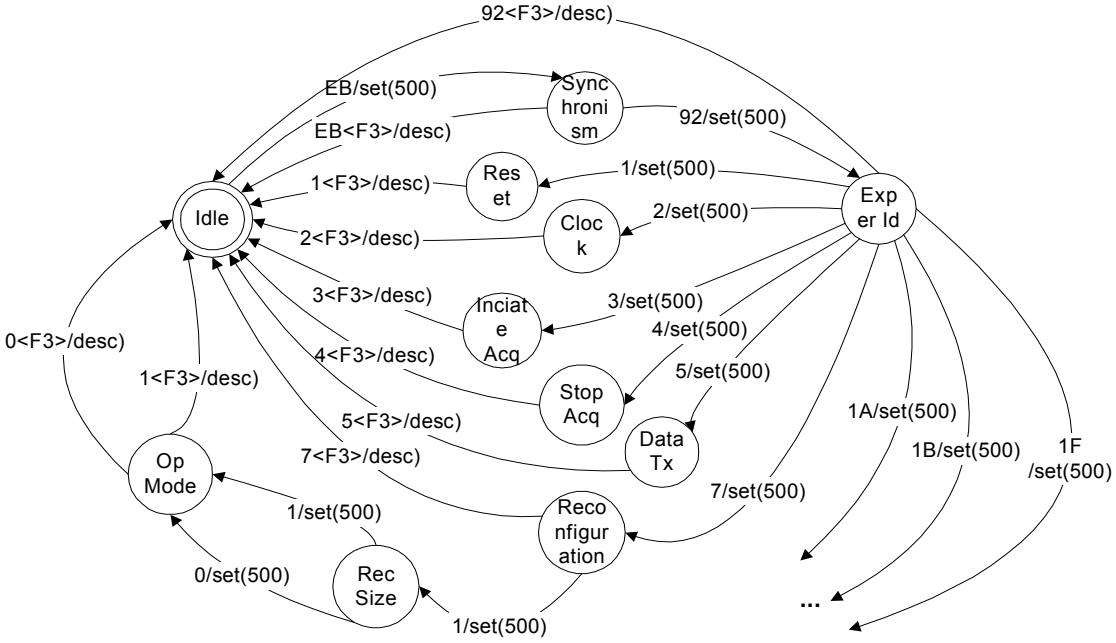
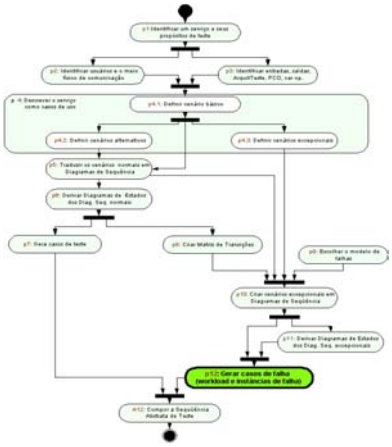


FIGURA 4.16 - Diagrama de Estados *Exceções TF de duplicação* OBDH-EXP.

4.4.12 Passo 12: Gerar casos de falha



Este passo consiste na geração dos casos de falha a partir dos cenários excepcionais. A geração pode ser de duas maneiras, a partir dos diagramas de seqüência gerados no passo 10 ou a partir dos diagramas de estado gerados no passo 11.

No primeiro caso, cada cenário dá origem a um caso de falha. Como mencionado anteriormente, os casos de falha são candidatos a serem executados com a técnica de injeção de falhas. O uso dessa técnica requer informações, tais como *quando* ativar a falha, qual *tipo* de falha, como sincronizar a injeção da falha com as entradas normais. Um caso de falha extraído dos diagramas de seqüência excepcionais contém tanto entradas normais do serviço (*workload*), como a caracterização das falhas a serem injetadas (*faultload*), respectivamente, nas interações normais e nas interações de falhas. Em um experimento de injeção de falhas, a primitiva de falha (associada à interação de falha) é tratada pelos injetores de falha e as entradas ou eventos normais são tratadas pela UUT e fluem nos PCOs respectivos.

Alternativamente, pode-se usar a ferramenta Condado para geração de casos de falha automaticamente a partir dos diagramas de estados excepcionais, como no passo 7. Aqui, a sincronização entre o injetor de falhas e os testadores LT e UT se dá em cada entrada do caso de teste. Os testadores são os componentes da arquitetura Ferry-Injection que transmitem as entradas normais ao serviço em teste, já os injetores são responsáveis por enviar os comando sobre as falhas a serem injetadas na execução do caso de falha. (Uma descrição da arquitetura Ferry-Injection encontra-se na Seção 3.4.2).

Cabe observar que os parâmetros das falhas indicadas nos diagramas de estados foram omitidos para simplificação.

Exemplo

A TABELA 4.11 ilustra 6 dos 28 casos de falha originados do diagrama de estados TF – duplicação, do exemplo OBDH-EXP (ver FIGURA 4.16) gerados pela Condado. As transições de falha são indicadas com as letras f3, indicando duplicação, após o evento. A Condado não está preparada para distinguir os casos de falha, então, as entradas marcadas com falha (f_i) devem ser transformadas em entradas para os injetores de falhas (FIC e FIM).

TABELA 4.11 - Casos de falha para o exemplo OBDH-EXP.

```
senddata(L,EB)  recdata(L,set500)
senddata(L,EBf3)  recdata(L,desc)

senddata(L,EB)  recdata(L,set500)
senddata(L,b92)  recdata(L,set500)
senddata(L,EBf3)  recdata(L,desc)

senddata(L,EB)  recdata(L,set500)
senddata(L,b92)  recdata(L,set500)
senddata(L,b1)  recdata(L,set500)
senddata(L,b1f3)  recdata(L,desc)

senddata(L,EB)  recdata(L,set500)
senddata(L,b92)  recdata(L,set500)
senddata(L,b2)  recdata(L,set500)
senddata(L,b2f3)  recdata(L,desc)

senddata(L,EB)  recdata(L,set500)
senddata(L,b92)  recdata(L,set500)
senddata(L,b3)  recdata(L,set500)
senddata(L,b3f3)  recdata(L,desc)

....
```

4.5 Considerações Gerais

Este Capítulo descreveu detalhadamente os passos da metodologia para geração de teste denominada $CoFI_m$, a qual cobre a atividade de "Definir a Seqüência Abstrata de Teste (SAT)" do processo descrito no Capítulo 3. Ao longo da descrição, os passos foram exemplificados com o protocolo OBDH-EXP. Adicionalmente à apresentação detalhada

dos passos, foram apresentados: (i) o contexto das falhas tratadas pela metodologia; (ii) o fluxo dos passos, em um diagrama de atividades, e; (iii) o relacionamento dos artefatos manipulados, em um diagrama de classes. O Capítulo é encerrado com um sumário das vantagens de cada passo e outro sumário abordando considerações para automação dos passos.

Na CoFI, como os serviços são tratados isoladamente, essa metodologia possui as seguintes vantagens: evita-se a explosão de estados no mapeamento do comportamento do sistema; todos os passos ficam simples, e; os mapeamentos podem ser realizados mesmo para uma especificação parcial (incompleta) do sistema. Além disso, a separação dos cenários normais e excepcionais e a criação de vários diagramas que mapeiam as situações normais por propósito de teste facilitam a manipulação, a aplicação e a análise dos resultados dos testes. Este fato permite que essas tarefas de teste, sejam feitas de forma incremental com o uso de técnicas formais. A TABELA 4.12 sumariza as vantagens que justificam cada um dos passos da metodologia $CoFI_m$.

A informação acrescentada de um passo para outro, na $CoFI_m$, segue um fluxo transformacional de forma a possibilitar a automação dos seus passos. A automação é evidente em alguns passos, promissora com requisitos a serem investigados em outros, enquanto que em alguns ela, essencialmente, requer a intervenção humana. Uma breve discussão de pontos que podem ser automatizados nos passos, é apresentada na TABELA 4.13.

A vantagem da metodologia proposta é poder usar os modelos do desenvolvimento do software. Entretanto, é importante lembrar que no caso de reutilização dos modelos do desenvolvimento, eles devem ser ajustados de acordo com a arquitetura de teste e os pontos de controle e observação disponíveis para realização dos testes.

TABELA 4.12 - Sumário das vantagens de cada passo da $CoFI_m$.

Passo	Descrição	Vantagem do passo
p1	selecionar serviço e associar propósito de teste	evita a proliferação de informações a serem tratadas de uma só vez
p2	Usuários, meio físico de comunicação	Estabelece o papel dos usuários, melhora o conhecimento das características, do meio de comunicação, externas à implementação
p3	Entradas, saídas, var. operacionais	delimita o escopo do serviço e estabelece suas interfaces externas
	Arquitetura de teste, PCOs	define o contexto do sistema a ser utilizado para realização da validação, antevê os recursos necessários para os testes e, aumenta a testabilidade
p4	Casos de uso e cenários	proporciona maior conhecimento dos requisitos funcionais e os cenários em que o serviço é usado
p5	Diagrama seqüência normal	estabelece a ordem das interações e ações e possibilita uma visão global da colaboração entre todas entidades
p6	Diagrama de estados normal	permite mostrar a dinâmica de manipulação de todos (muitos) eventos de um modo sintético, é uma notação formal e, permite geração automática de testes
p7	Geração automática de casos de teste de conformidade	reduz o tempo com a atividade de projeto de teste, aumenta a precisão, uniformiza a descrição das seqüências de teste, aumenta a independência da pessoa que testa e, possibilita a previsão do número de casos de teste gerados com relação à cobertura da especificação
p8	Matriz de transições	facilita a análise do comportamento do serviço em presença de eventos corretos ocorrendo em estados errôneos; força o levantamento das saídas previstas para todas essas situações
p9	Modelo de falhas	volta a atenção do testador para os erros (problemas externos) que o serviço pode sofrer do meio de comunicação (ou do ambiente)
p10	Diagrama seqüência excepcional, por classe de erro	facilita o entendimento do tratamento dos vários tipos de erros aos quais o serviço é submetido durante sua operação; simplifica os desenhos, permite definição de experimentos de injeção de falhas e, facilita visualização da sincronização dos *workload e faultload*
p11	Diagrama de estados excepcional por classe de erro	evita explosão de estados na modelagem do comportamento do serviço; a explosão de casos de falha; facilita a definição do oráculo para os eventos errôneos, graças a definição de transição de falha
p12	Geração automática de casos de falha	além das vantagens da geração automática, possibilita a previsão do número de casos de falha gerados com relação à cobertura de classes de erros do meio de externo (teste de robustez)

TABELA 4.13 - Considerações sobre automação dos passos da $CoFI_m$.

Passo	Descrição	Automação
p1	selecionar serviço e associar propósito de teste	Dificilmente poderão ser automatizados
p2	Usuários, meio físico de comunicação	
p3	Entradas, saídas, var. operacionais	
	Arquitetura de teste, PCOs	componentes que apóiam a arquitetura Ferry-injection estão sendo desenvolvidos
p4	Casos de uso e cenários	Sempre requerem a intervenção humana
p5	Diagrama seqüência normal	Requer uma linguagem que descreva o diagrama de seqüência de forma computacional como a UML
p6	Diagrama de estados normal	Implementar um algoritmo que traduza o diagrama de seq em diagrama de estado
p7	Geração automática de casos de teste de conformidade	Pode-se contar com a Condado (do projeto ATIFIS) e a MGSET (do projeto PLAVIS)
p8	Matriz de transições	Tem que ser completada manualmente.
p9	Modelo de falhas	As primitivas de falha dependem do meio físico e devem ser re-escritas para cada UUT. Uma estrutura com padrões de projeto (*design patterns)* está sendo projetada
p10	Diagrama seqüência excepcional, por classe de erro	derivação de cenários por caminhos furtivos pode fazer uso de diretiva DIF, daí, a derivação de casos de falha furtivos pode ser automatizada. É viável a definição de um algoritmo para gerar cenários excepcionais combinando cenários normais e primitivas de falha. A saída seria na mesma linguagem usada no passo 5
p11	Diagrama de estados excepcional por classe de erro	Implementar um algoritmo para gerar diagramas de estados excepcionais a partir dos diagramas de seq incluindo informações sobre as falhas. Estender a Condado para distinguir entradas para injetores de falha
p12	Geração automática de casos de falha	Definir um algoritmo para extrair as definições pertinentes à injeção de falhas

CAPÍTULO 5

AVALIAÇÕES EMPÍRICAS DA SEQÜÊNCIA DE TESTE CoFI

Neste Capítulo são discutidos os resultados obtidos em três estudos de caso com sistemas de comunicação de aplicações espaciais reais. O primeiro estudo foi o protocolo OBDH-EXP, usado nos exemplos do Capítulo 4. Este protocolo consta da comunicação entre dois sistemas a bordo de satélite. Esse caso permitiu a realização de dois tipos de experiência: execução da seqüência de teste CoFI a uma implementação desse protocolo e uma comparação da seqüência de teste $CoFI_m$ com a seqüência de teste gerada segundo a metodologia N+ (Binder, 2000). A comparação entre as seqüências foi quanto ao seu nível de confiança da adequação; medida essa obtida com a técnica de mutação em máquinas de estados que representam o comportamento do sistema em teste.

No segundo estudo de caso, a metodologia $CoFI_m$ foi aplicada à norma ECSS-E-70-41A (ECSS, 2003) desenvolvida pela ESA. Mais precisamente, a $CoFI_m$ foi aplicada em um serviço de comunicação entre aplicações em solo e a bordo de espaçonaves definido nessa norma. O estudo de caso focou a geração de casos de teste a partir de uma especificação normatizada. Os casos de teste projetados não foram executados. Apenas avaliados quanto ao nível de confiança da adequação.

No terceiro estudo de caso, foi usada a especificação de um protocolo definido pelo Centro de estudos espaciais francês, o CNES, para comunicação entre aplicações do Centro de Controle de Satélites e equipamentos de uma estação terrena de rastreio de satélite. Esse protocolo referenciado aqui, como solo-solo, está especificado na forma de um autômato. Este estudo mostra que os passos da $CoFI_m$ podem ser excluídos dependendo da documentação fornecida (especificação de teste), pois apenas os diagramas de estados foram manipulados.

Em todos os estudos, a geração automática de casos de teste contou com o apoio das ferramentas Condado (Martins et al., 1999) e do Modelador de Máquinas de Estado (MME) do projeto ATIFS (INPE, 2002).

Uma forma de comparar seqüências de teste é através da medida do seu nível de adequação a um determinado conjunto de erros. Para realizar essa medida para seqüência CoFI, foram utilizadas algumas ferramentas da plataforma PLAVIS (Plavis, 2005). Esta plataforma, entre outras capacidades, é capaz de gerar mutantes de especificações em máquinas finitas de estados (Fabri, 1996) e calcular o escore de mutação.

O escore de mutação é calculado com o número de mutantes gerados, o número de mutantes equivalentes e o número de mutantes mortos. O escore provê uma medida para o nível de confiança da adequação dos casos de teste (DeMillo, 1980). Se um mutante apresenta resultado diferente da especificação original (P), ele é dito morto, nesse caso, o conjunto de teste (T) identificou o erro no mutante, ou seja, T revelou a diferença entre a mutação e a especificação dada, caso contrário, o mutante é dito estar vivo. Dentre os mutantes vivos, diz-se que são equivalentes aqueles cuja saída é igual a saída da máquina original; portanto, nada se pode concluir. Dado um produto P (especificação em máquina de estados) e um conjunto de casos de teste T, o escore de mutação *sm*(P,T) é computado como:

$$sm(\mathrm{P},\mathrm{T}) = \frac{muttes\_mortos}{muttes\_criados - muttes\_equivalentes}$$

Os operadores de mutação utilizados para geração dos mutantes foram: (i) alterar saída; (ii) excluir saída; (iii) excluir transição; (iv) alterar evento; (v) excluir evento, e; (vi) alterar estado inicial. Estes operadores representam classes de erros de entrada e saída de transições, que são aqueles erros detectáveis pelo método de geração de teste "*Swich cover*" (Chow, 1978), implementado na ferramenta Condado. Apesar da plataforma PLAVIS possibilitar a geração de mutantes de estados também, esses mutantes não foram utilizados, pois, os casos de teste gerados pela Condado não garantem a detecção de erros de estados. A geração automática de casos de testes capazes de detectar erros de estados requer uma especificação completa do comportamento do sistema a ser testado. Na abordagem CoFI, a especificação não, necessariamente, é completa. Cabe observar que especificações completas nem sempre são possíveis no mapeamento de

sistemas reais. Além do uso parcial do conjunto com operadores de mutação oferecido pela PLAVIS, os mutantes equivalentes do conjunto de mutantes gerados não foram considerados na análise realizada neste trabalho.

A eficácia das seqüências geradas pela metodologia $CoFI_m$, para os estudos de caso ilustrados aqui, foi obtida pelas métricas fornecidas pela plataforma PLAVIS, segundo a seleção de operadores de mutação de máquinas de estados mostrados em cada estudo.

5.1 Estudo de Caso 1: o Protocolo OBDH-EXP

Este estudo de caso cobre a abordagem em que a especificação não é uma norma pública, detalhes da implementação são conhecidos e os casos de teste e de falhas foram executados e os resultados são avaliados. A especificação desse protocolo não é organizada em serviços, então a função de reconhecimento de comandos, parte do protocolo de comunicação, foi escolhida como alvo dos testes. Esse protocolo, desenvolvido no INPE, provê a comunicação entre um computador de manipulação de dados a bordo (OBDH) e um equipamento de carga útil científica (EXP). A implementação usada como alvo de teste neste estudo consta do experimento científico denominado Alpha, Proton and Electron Monitoring Experiment in the Magnetosphere (APEX) a voar no satélite *Equatorial Atmosphere Research Satellite* (EQUARS). A especificação alvo foi o documento de definição do protocolo descrito textualmente em INPE (2005).

O resultado obtido com a aplicação da metodologia, passo-a-passo, é apresentado no Capitulo 4. Os resultados da execução da seqüência de teste aplicada à implementação real são discutidos na Seção 5.1.1 e as comparações com a metodologia N+ são apresentadas na Seção 5.1.2.

5.1.1 Avaliação de uma Implementação do Protocolo

Os casos de teste normais e excepcionais foram executados contra a implementação do software desenvolvido pela equipe responsável pelo experimento APEX. Os testes

foram aplicados manualmente com o apoio de um programa de teste adaptado para apoiar a execução atraso, perda, duplicação e corrupção de bytes escolhidos pelo usuário via uma interface gráfica, no LabView. A TABELA 5.2 mostra alguns exemplos de testes aplicados: as entradas aplicadas e as saídas observadas.

TABELA 5.1 - Alguns casos de teste aplicados na implementação do APEX.

Entrada	Saída Observada	Comentários
EA9205cs	*timeout*	SYNC não reconhecido
EB(f.delay)9205	Nenhuma reação	Nenhuma reação
EB(f.delay)92	*timeout*	EXPID – não recebido
EB9905cs	*timeout*	EXPID - desconhecido
EB9299cs	*timeout*	TYPE - inválido
EB92(fdelay)01	*timeout*	TYPE - não recebido
...	...	...
EB921A04FFFF FFFF	*timeout*	Endereço incorreto
EB921F0131CC	*timeout*	CKSUM - incorreto

Os testes aplicados à implementação do protocolo APEX não revelaram nenhum erro. Por um lado, pode-se concluir que ele estava conforme a especificação, no que concerne à cobertura de todo comportamento esperado para as entradas especificadas e para os erros de comunicação selecionados. Apesar do conjunto de testes CoFI não ter encontrado erros e não conformidades na implementação do APEX, essa abordagem foi bem aceita pelo pessoal responsável pelo desenvolvimento desse sistema no INPE. A metodologia introduziu uma nova maneira de planejar o esforço dos testes com base na cobertura, tanto do modelo comportamental quanto do modelo de falhas externa; o modelo do comportamento normal permite revelar desentendimentos em interfaces associadas a eventos, enquanto que o modelo das falhas externas permite identificar as respostas do sistema frente a influências do ambiente.

5.1.2 Comparação com a Metodologia N+

Uma versão reduzida da especificação considerando-se apenas os comandos *reset* e *data transmission,* foi usada para se obter os casos de teste manualmente, com a aplicação da metodologia N+ e facilitar as comparações.

A **metodologia N+** definida por Binder (2000, Cap.7), integra elementos de teste baseados em estados, modelos de estados da UML e considerações de teste exclusivamente para implementações a objetos. Esta metodologia consiste dos seguintes passos:

- desenvolver o modelo de estados que represente o comportamento do software a ser testado. Esse modelo não expressa hierarquia nem paralelismo,
- desenvolver a Matriz Resposta, na qual a suposição de completeza é estabelecida para todas as combinações de pares (eventos, estados), inclusive para valores parâmetros das condições associados aos eventos,
- criar uma árvore de alcançabilidade a partir do modelo de estados e gerar casos de teste dos caminhos (*round-trip*),
- gerar os casos de teste de caminhos ilícitos (*sneak path)*, derivados dos elementos na Matriz Resposta, que não foram cobertos no modelo e que não representam valores mutuamente exclusivos das condições do evento,
- o último passo consiste em encontrar valores que satisfaçam às condições de guarda para os casos de teste, no entanto, este passo não foi considerado aqui.

O modelo usado na N+ corresponde ao diagrama de estados do comportamento desse protocolo ilustrado na FIGURA 5.1, também chamado modelo Total.

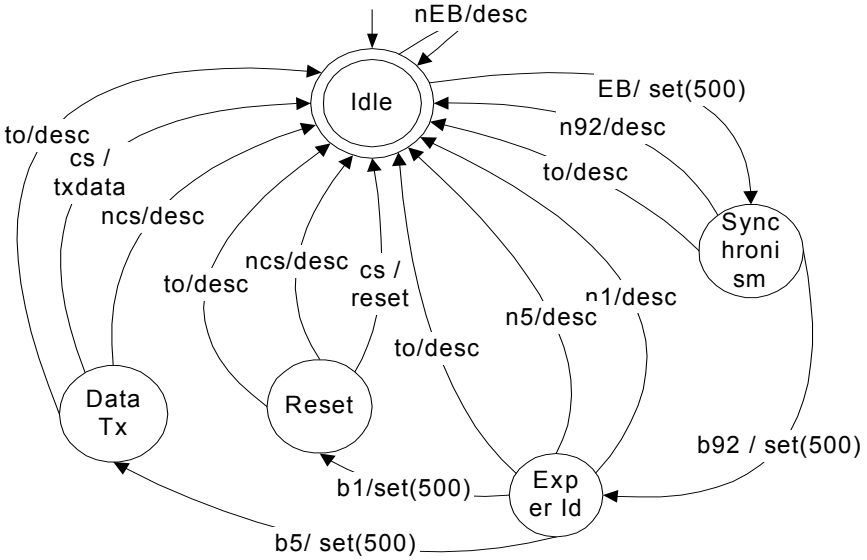


FIGURA 5.1 - Diagrama de Estados Total do OBDH-EXP 2comandos.

A árvore de alcançabilidade, a partir da qual são derivados os testes *round-trip* e, a matriz resposta, de onde são criados os teste de caminhos furtivos (*sneak path*), são ilustradas no Apêndice B. A TABELA 5.2 apresenta a seqüência de teste N+, contendo 47 casos de teste.

TABELA 5.2 - Seqüência de teste segundo a metodologia N+.

Sneak path			***Round-trip***
	EB b1	EB b92 b5 b92	nEB
b92	EB FF	EB b92 b5 n92	EB to
n92	EB cs	EB b92 b5 b5	EB nEB
b5	EB ncs	EB b92 b5 b1	EB b92 b1 cs
b1	EB b92 EB	EB b92 b5 FF	EB b92 b1 ncs
n5	EB b92 nEB	EB b92 b1 EB	EB b92 b1 to
n1	EB b92 b92	EB b92 b1 nEB	EB b92 b5 cs
cs	EB b92 n92	EB b92 b1 b92	EB b92 b5 ncs
ncs	EB b92 cs	EB b92 b1 n92	EB b92 b5 to
EB EB	EB b92 ncs	EB b92 b1 b5	EB b92 n1
EB nEB	EB b92 b5 EB	EB b92 b1 b1	EB b92 n5
EB b5	EB b92 b5 nEB	EB b92 b1 FF	EB b92 to
35			12

Na **metodologia $CoFI_m$** 41 casos de teste foram gerados a partir dos diagramas de estados parciais (ver TABELA 5.3). Os diagramas de estados parciais são apresentados na FIGURA B.2, FIGURA B.3, FIGURA B.4, FIGURA B.5 e na FIGURA B.6 do Apêndice B.

TABELA 5.3 - Seqüência de teste segundo a metodologia $CoFI_m$.

Normais	**Exceção especificada**	**Exceção caminhos furtivos**		**Exceção TF**
EB b92 b1 cs	nEB	Cs	EB b92 b92	EB b92 b5 b5
EB b92 b5 cs	EB n92	b1	EB b92 b1 EB	EB b92 b5 b1
	EB to	b5	EB b92 b1 b5	EB b92 to
	EB b92 n1	b92	EB b92 b1 b1	EB b92 b5 to
	EB b92 n5	EB b5	EB b92 b1 b92	EB EB
	EB b92 to	EB EB	EB b92 b5 b5	EB b92 b92
	EB b92 b1 ncs	EB b1	EB b92 b5 b1	EB b92 b1 cs cs
	EB b92 b1 to	EB cs	EB b92 b5 EB	EB b92 b1 b1
	EB b92 b5 ncs	EB b92 EB	EB b92 b5 b92	EB b92 b1 cs
	EB b92 b5 to	EB b92 cs		EB b92 b5 b5
2	10	19		10

Uma comparação das seqüências de teste foi realizada com métricas do nível de confiança da adequação dos casos de teste computado no escore de mutação pela PLAVIS para os operadores listados na TABELA 5.4. A geração das seqüências de teste baseou-se no modelo Total da especificação ilustrado na FIGURA 5.1.

TABELA 5.4 - Operadores de mutação do OBDH-EXP 2comandos.

Operadores	**Número de mutantes**
Altera saída (OutAlt)	48
Exclui saída (OutDel)	16
Exclui transição (TraArcDel)	9
Altera evento (TraEveAlt)	68
Exclui evento (TraEveDel)	16
Altera estado inicial (TraIniStaAlt)	4
	161

O resumo dos resultados obtidos com a avaliação das seqüências N+ e CoFI é apresentado na TABELA 5.5. Nesse exemplo, observa-se, na coluna #c.t + c.f (número de casos de teste somados ao número de casos de falha) que a seqüência CoFI é menor, isto é, contém menos casos de teste que a seqüência N+. Além disso, a seqüência CoFI é mais adequada aos erros de transições, uma vez que o escore de mutação obtido por ela é maior que o escore de mutação da seqüência N+.

TABELA 5.5 - Escore de mutação das seqüências CoFI e N+.

Seqüência de teste	# c.t + c.f	# c.t. + c.f distintos	# Mortos	# Vivos	Escore
N+	**47**	**46**	154	7	**0,956**
CoFI	**41**	**33**	161	0	**1,000**

Além da comparação da seqüência completa, a qual inclui os casos de teste e casos de falha, mostrada na TABELA 5.5, foram realizadas algumas comparações com subconjuntos da seqüência. Um subconjunto foi chamado *Core*, pois consta dos testes gerados a partir do modelo de estados do comportamento normal e do comportamento excepcional, os quais mapeiam exclusivamente as informações descritas na especificação em dois modelos distintos. Esses modelos são apresentados pelas máquinas: Normal e ExcEspecificadas ilustrados no Apêndice B. A seqüência Total foi gerada a partir do diagrama da FIGURA 5.1, que mapeia o comportamento normal e o excepcional em um único modelo.

Comparações entre as seqüências Total e *Core* mostram a equivalência entre elas (ver TABELA 5.6). Essa equivalência entre os conjuntos Total e Core apontam empiricamente que a separação do comportamento normal e do comportamento excepcional especificado, não reduz o potencial do conjunto de testes parciais quando comparado com os testes derivados do modelo do comportamento Total. Esse resultado deve-se ao fato de que as máquinas: Normal e ExcEspecificadas são complementares, sendo a união das mesmas, a máquina Total. Demonstrações matemáticas serão exploradas em trabalhos futuros. Entretanto, a união das seqüências parciais não produz

o mesmo escore de mutação resultante da soma dos escores obtidos pelas seqüências parciais, isso, porque cada seqüência de teste parcial pode ter matado mutantes em comum. A relação dos escores segue as regras:

- ms(seq1 U seq2) >= ms(seq1)
- ms(seq1 U seq2) >= ms(seq2)
- ms(seq1 U seq2) <= ms(seq1) + ms(seq1)

A TABELA 5.6 ilustra também os resultados considerando-se cada máquina isoladamente. Observa-se que as seqüências parciais individualmente não são plenamente adequadas às falhas de transições como mostram os escores das seqüências parciais.

TABELA 5.6 - Comparação entre as seqüências Total, *Core* e Parciais.

Seqüência de teste	**# c.t + c.f**	**# c.t. + c.f distintos**	**# Mortos**	**# Vivos**	**Escore**
Modelo **Total**	12	12	161	0	**1,000**
Core da especificação	12	12	161	0	**1,000**
Normal	2	2	77	84	0,478
ExcEspecificadas	10	10	143	18	0,888
ExcFurtivos	19	19	76	85	0,472
ExcTF	10	10	98	63	0,608

5.2 Estudo de Caso 2: Serviço de Verificação de Telecomando

Este estudo de caso cobre a abordagem em que uma especificação padronizada (norma) é considerada. A norma ECSS-E-70-41A (ECSS, 2003) especifica serviços de comunicação entre um computador de bordo central a bordo e um sistema de controle em solo. A FIGURA 5.2 ilustra os artefatos usados na geração da SAT.

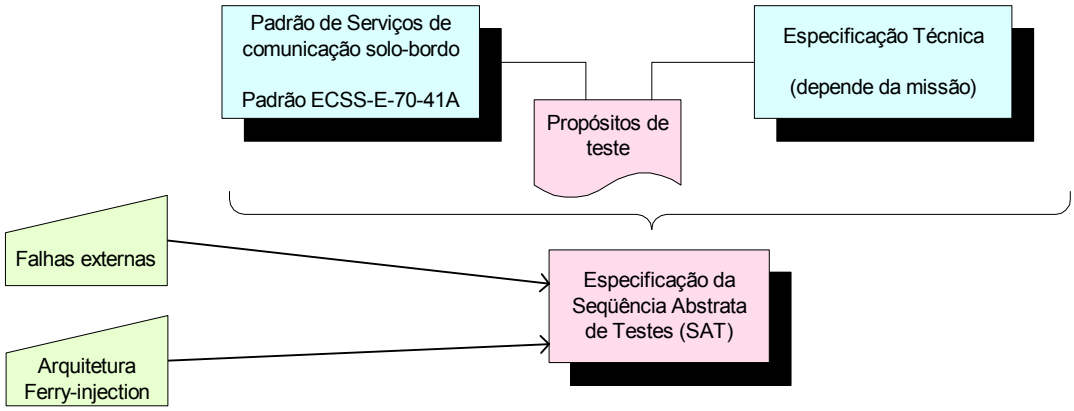


FIGURA 5.2 - Artefatos para geração da SAT do serviço Verificação de TC.

5.2.1 Escopo da Norma ECSS-E-70-41A - Packet Utilization Service (PUS)

A norma ECSS-E-70-41A – *Space Engineering – Ground Systems And Operations – Telemetry And Telecommand Packet Utilization Service* (PUS) descreve um conjunto de serviços que orienta a definição da interface solo-bordo no que concerne à comunicação entre processos de aplicação (*Application Process* - AP) implementados a bordo de satélites e aplicações pertencentes ao sistema de solo.

A norma supõe que estes serviços façam uso das camadas do protocolo CCSDS (do nível físico ao nível de pacotes) (CCSDS, 2001a), (CCSDS, 2001b), (CCSDS, 2001c), as quais são responsáveis pela transmissão de pacotes de TM e de TC entre as partes comunicantes, conforme ilustrado no sistema de telecomando da FIGURA 5.3.

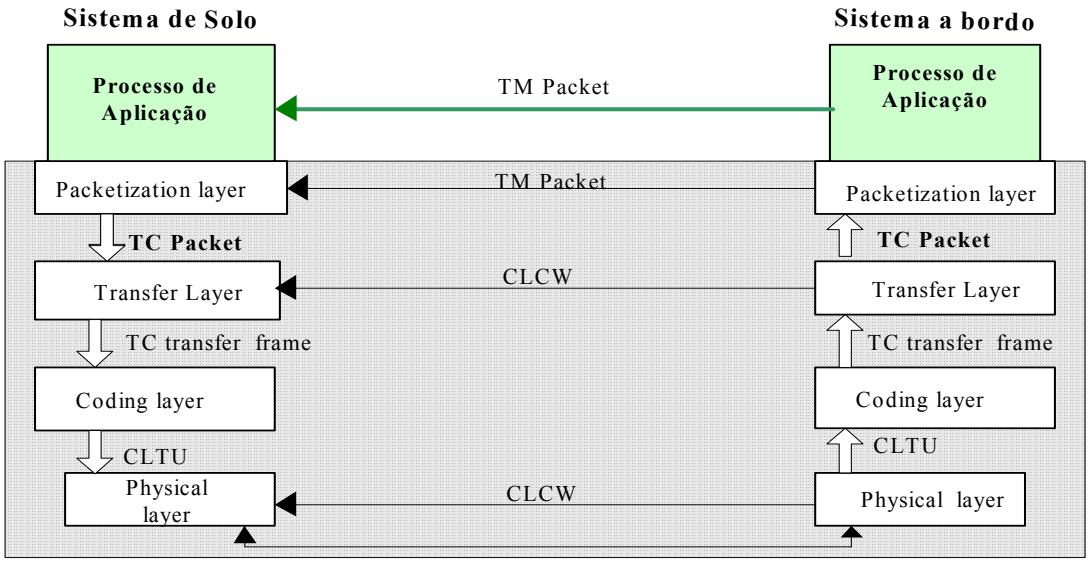


FIGURA 5.3 - Camadas do sistema de telecomando.

A camada de pacotes (Packetization layer) empacota os dados da aplicação, transporta-os nas estruturas de dados denominadas *TM packet* e *TC packet*, cujos campos de dados incluem os comandos (*request*) e as respostas (*reports*) dos serviços, respectivamente. Tais estruturas têm seus formatos definidos nas recomendações CCSDS, exceto para o campo de dados do pacote que é definido na norma ECSS-E-70-41A, conforme mostrado nas figuras: FIGURA 5.4, pacote de telecomando e FIGURA 5.5, pacote de telemetria.

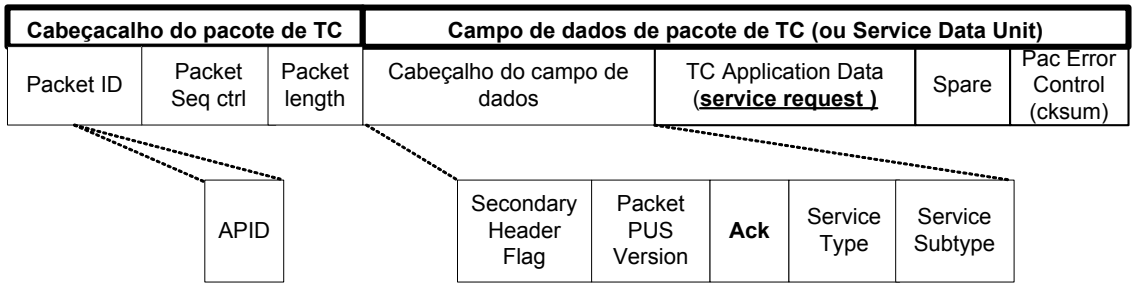


FIGURA 5.4 - Estrutura do pacote CCSDS de telecomando.

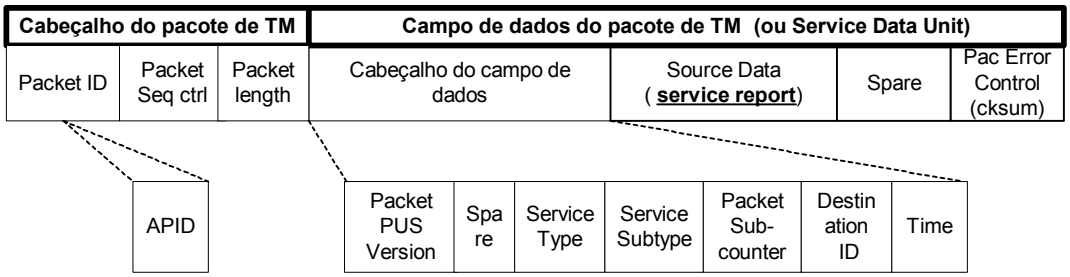


FIGURA 5.5 - Estrutura do pacote CCSDS de telemetria.

Esta norma especifica 16 serviços. Cada serviço é unicamente identificado e pode ser implementado por um ou mais processos de aplicação (*Application Process - AP)*, contém funcionalidade mínima e máxima representadas pelos subtipos do serviço e possui atividades iniciadas pelo usuário do serviço. Essas atividades são caracterizadas pelos *Requests* e *Reports*, parâmetros, descrição da unidade de dado, ações, indicações, não exaustiva, de erros.

5.2.2 Especificação do Serviço de Verificação de Telecomando

O serviço de verificação de telecomando checa a veracidade dos campos do pacote de TC recebido, isto é, faz a aceitação do pacote para um processo de aplicação a bordo e acompanha os estágios de execução do comando: início, progresso e finalização. Esse serviço é útil para execuções de longa duração, como aquelas em missões ao espaço profundo (*deep space*), durante as quais os estágios de execução do TC podem levar horas. Assim, quando esse serviço é solicitado, o cliente pode requerer informações (*reports*) tanto da aceitação quanto sobre os estágios de execução. Essa solicitação é feita com uma indicação nos quatro *bits* do campo Ack do campo de dados do pacote de TC. O bit-3 com valor 1 indica solicitação de *report* de aceitação, o bit-2 indica *report* de inicialização, bit-1 de progresso e o bit-0 de finalização. *Reports* de falha não requerem solicitação do sistema de solo, serão sempre enviados quando da ocorrência de uma falha. A TABELA 5.7 relaciona as capacidades desse serviço, as quais constam de informações (*reports*) de sucesso e de falha, os possíveis subtipos, seus valores, o *report* que ele representa, informações adicionais e o mnemônico utilizado nas descrições a seguir deste Capítulo. O código de falha para um *report* de aceitação do

pode conter: (0) identificação do processo de aplicação destino (APID) desconhecido; (1) comprimento incorreto; (2) *checksum* incorreto; (3) tipo ilegal; (4) subtipo ilegal; (5) data de aplicação inconsistente (campo do pacote TC chamado *service request data element*). Para os demais pacotes, o código de falha deve ser especificado na missão.

TABELA 5.7 - Informações (*reports*) geradas pelo ECSS-Verificação de TC.

Sub-tipo	**Descrição:** ***Report*** **(mnemônico)**	**Informações adicionais (*service report elements*)**
1	Aceitação do TC – Sucesso (AccSucc_rep)	TC packet ID
		Packet Seq Ctrl
2	Aceitação do TC – Falha (AccFail_rep)	TC packet ID
		Packet Seq Ctrl
		Código de erro (ce)
		Parâmetros de falha
3	Início da execução do TC – Sucesso (StartSucc_rep)	Idem 1
4	Início da execução do TC – Falha (StartFail_rep)	Idem 2
5	Prosseguimento da execução do TC – Sucesso (PrSucc_rep)	TC packet ID
		Packet Seq Ctrl
		Num. Passo (p)
6	Prosseguimento da execução do TC – Falha (PrFail_rep)	TC packet ID
		Packet Seq Ctrl
		Num. Passo (p)
		Código de erro
		Parâmetros de falha
7	Fim da execução do TC – Sucesso (FinSucc_rep)	Idem 1
8	Fim da execução do TC – Falha (FinAccFail_rep)	Idem 2

O Apêndice C apresenta, passo a passo, a geração da seqüência abstrata de teste de acordo com a metodologia $CoFI_m$, ilustrando todos os diagramas criados, intermediários e finais. Uma descrição desse serviço com exemplos de casos de teste e de falha escritos em TTCN pode ser encontrada no relatório (Ambrosio et al., 2004b). A seguir são discutidos os resultados da avaliação das várias seqüências de teste geradas.

5.2.3 Avaliação da Seqüência de Teste CoFI com Relação ao Modelo Total

O Modelo Total deste exemplo foi elaborado para fins de comparação com os Modelos Parciais, criados segundo a metodologia $CoFI_m$. O Modelo Total contém todas as entradas especificadas, normais e excepcionais. Esse modelo é total no sentido de que contempla todo o comportamento especificado na norma ECSS-E-7041A e na Especificação Técnica. O Apêndice C ilustra esse diagrama.

Primeiramente foram feitas comparações com relação ao tamanho dos modelos (número de estados, de transições e de entradas distintas), ao tamanho das seqüências de teste (número de casos de teste) e ao comprimento dos casos (número de entradas).

A TABELA 5.8 resume a cardinalidade dos modelos parciais (normais e excepcionais) e do Modelo Total. O campo *união dos parciais* traz o número de estados, transições e entradas distintas resultante da união dos conjuntos de estados, transições e entradas distintas dos modelos parciais. No Modelo Normal do Propósito 2, o estado *TC recebido* foi criado para representar os estados do Modelo Normal do Propósito 1. Uma relação de hierarquia foi criada nessa modelagem, por isso, a soma dos estados dos modelos parciais possui um estado a mais.

A diferença entre o número de entradas distintas do Modelo Total e a soma do número de entradas distintas dos Modelos Parciais, deve-se ao uso dos símbolos *to* e CmdDesc para representar os eventos errôneos causados por perturbações externas. Caso contrário, esses números deveriam ser iguais. A diferença de transições deve-se ao acréscimo das transições de caminhos furtivos e das transições provocadas pelas entradas *to* e *cmdDesc*.

TABELA 5.8 - Cardinalidade das Máquinas Parciais – ECSS-Verificação de TC.

Modelos		**Tamanho do Modelo**			**Tamanho da Seqüência**		**Comprimento dos casos de teste**	
		#est	**# trans**	**#entr dist**	**# c.t.**	**# c.f.**	**maior**	**menor**
Normais	Prop 1	9	10	10	3		8	7
	Prop 2	5	11	9	10		6	4
Excepcionais	Especif. Prop1	7	12	12	-	6	7	2
	Especif. Prop 2	5	10	9	-	6	6	3
	Furtivos Prop 1	9	17	10	-	10	9	8
	Furtivos Prop 2	5	22	8	-	18	18	1
	TF-atraso Prop 2	11	30	27	-	6	6	1
	TF-corrup Prop 2	5	11	9	-	7	6	1
União dos parciais		12	57	29	**13**	**53**	**18**	**1**
					66			
Modelo Total		11	30	27	**41**		**13**	**2**

O conjunto de casos de teste e de falhas gerados com os modelos obtidos pela metodologia $CoFI_m$ é chamado aqui de S_{CoFI} e designa a união das seqüências de teste dos modelos parciais. A seqüência de teste gerada a partir do Modelo Total é referenciada como $S_{MT.}$

A TABELA 5.9 apresenta a comparação traçada entre as seqüências parciais com relação à repetição de casos de teste. Neste exemplo, observa-se que:

- Seq5 contém a Seq1, isto é, três casos de teste são comuns entre eles.
- Seq2 e Seq6 contém dois casos em comum.
- Seq6 e Seq7 contém dois casos em comum.
- Seq6 e Seq8 contém dois casos em comum.

- Seq7 e Seq8 contém três casos em comum.

No total são treze (13) casos de teste comuns. Considerando-se que o conjunto total de casos de teste e casos de falha da S_{CoFI} é de 66 casos de teste, pode-se dizer então que, um conjunto de 53 casos seria suficiente. Em outras palavras, a seqüência S_{CoFI} possui 19,7% de casos de testes que poderiam ser eliminados de sua seqüência, sendo esse o aspecto negativo da automatização da atividade de geração de casos de teste.

TABELA 5.9 - Relações entre as seqüências da S_{CoFI}.

Modelos		**Nome Seqüência**	**# c.t + c.f.**	**Relação entre as seqüências parciais**
Normais	Prop 1	Seq1	3	Seq1 dif Seq3 e Seq5 contem Seq1
	Prop 2	Seq2	10	Seq2 dif Seq4 e Seq2 2c.t.= Seq6 Seq2 dif Seq7 e Seq2 dif Seq8
Excepcionais	ExcEspecif - Prop1	Seq3	6	Seg3 dif Seq5
	ExcEspecif - Prop2	Seq4	6	Seg4 dif Seq6 e Seg4 dif Seq7 Seg4 dif Seq8
	Furtivos - Prop1	Seq5	10	-
	Furtivos – Prop2	Seq6	18	Seg6 2c.t.= Seq7 e Seg6 3c.t.= Seq8
	TF-atraso - Prop 2	Seq7	6	Seg7 3c.t.= Seq8
	TF-corrup - Prop2	Seg8	7	-

Na comparação entre as duas seqüências de teste, S_{CoFI} e S_{MT}, com relação ao grau de repetição de casos de testes, a seqüência S_{MT} foi comparada com cada uma das seqüências parciais de S_{CoFI}. A TABELA 5.10 mostra esta comparação. Observa-se que apenas 9% são comuns.

TABELA 5.10 - Relações entre as seqüências parciais da S_{CoFI} e a S_{MT}.

Modelos		Seqüência parcial da S_{CoFI}	Relação entre as seqüências
Parciais	Propósito 1	Seq1	Seq1 dif S_{MT}
	Propósito 2	Seq2	Seq2 dif S_{MT}
	ExcEspecific-Prop1	Seq3	Seg3 5c.t.= S_{MT}
	ExcEspecific-Prop2	Seq4	Seg4 dif S_{MT}
	Furtivos -Prop1	Seq5	Seg5 1c.t.= S_{MT}
	Furtivos –Prop2	Seq6	Seg6 dic S_{MT}
	TF-atraso -Prop 2	Seq7	Seg7 dif S_{MT}
	TF-corrup - Prop 2	Seq8	Seg8 dif S_{MT}

A comparação da eficácia da S_{CoFI} contou com o conjunto de mutantes ilustrados na TABELA 5.11.

TABELA 5.11 - Operadores de Mutação para o ECSS-Verificação de TC.

Operadores	Número de mutantes
Altera saída (OutAlt)	546
Exclui saída (OutDel)	26
Exclui transição (TraArcDel)	21
Altera evento (TraEveAlt)	508
Exclui evento (TraEveDel)	30
Altera estado inicial (TraIniStaAlt)	10
	1141

O escore de mutação foi calculado para as seqüências S_{CoFI} e a $S_{MT.}$ Nesta comparação, dois experimentos foram realizados. No primeiro, as máquinas parciais consideradas foram aquelas ilustradas na Seção C.1 do Apêndice C, onde cada propósito de teste é modelado separadamente. No segundo, os propósitos de teste (Prop 1 e Prop 2) foram

modelados juntos em um único modelo. Esses modelos são apresentados na Seção C.2 do Apêndice C. A TABELA 5.12 apresenta o número de casos de teste e o escore de mutação para as seqüências $S_{MT,}$ S_{CoFI} –propósitos separados e S_{CoFI} –propósitos juntos.

TABELA 5.12 - Escore de mutação das seqüências S_{CoFI} e a S_{MT}.

Seqüência de teste	# c.t. + c.f.	# c.t. + c.f. distintos	# Mortos	# Vivos	Escore
S_{MT}	**41**	41	1117	24	**0,978**
S_{CoFI} propósitos separados	**66**	56	719	422	**0,630**
S_{CoFI} propósitos juntos	**107**	103	1117	24	**0,978**

Observa-se que a separação dos propósitos de teste em modelos distintos reduz a eficácia do conjunto de teste com relação à adequação às falhas de transições em FSMs. Por um lado, a estratégia de separar os modelos por propósito de teste é interessante para realização de testes incrementais, ela facilita a visualização e o foco na modelagem, mas por outro lado, ela perde o potencial de adequação com relação às falhas de transição do modelo comportamental.

As TABELA 5.13 e TABELA 5.14 apresentam as métricas das seqüências de teste geradas a partir dos modelos parciais.

TABELA 5.13 - Escore de mutação das parciais S_{CoFI} – propósitos separados.

Seqüência de teste	# c.t. + c.f.	# c.t. + c.f. distintos	# Mortos	# Vivos	Escore
Normal_prop1 Normal_prop2 ExcEspecif_prop1 ExcEpecif prop2	**25**	25	718	423	**0,629**
Normal_prop1 Normal_prop2 Furtivos_prop1 Furtivos_prop2 (~*N*+)	**41**	36	487	654	**0,426**
Normal_prop1 Normal_prop2	**13**	13	481	660	**0,421**
ExcEspecif_prop1 ExcEspecif _prop2	**12**	12	550	591	**0,482**
Furtivo_prop1 Furtivo_prop2	**28**	28	383	758	**0,335**
TF_AtrPerd_prop2 TF_Corrup_prop2	**13**	10	66	1075	**0,057**

A primeira tabela mostra a cardinalidade dos modelos por propósitos separados, enquanto que a segunda mostra os resultados obtidos com os modelos que mapeiam os propósitos um só diagrama. Os diagramas de estados que mapeiam os propósitos 1 e 2 juntos são apresentados no Apêndice C.

As métricas mostram que o escore de mutação para a seqüência de teste TF (**TF_AtrPerd_prop2 TF_Corrup_prop2**) é baixo. A explicação para este resultado é que essa seqüência de teste não é voltada para a cobertura das falhas estruturais (medida com o escore de mutação), uma vez que os eventos *cmdDesk* e *to* não foram, sequer, considerados no Modelo Total. Outra observação é que a soma dos escores dos modelos parciais não corresponde ao escore do modelo total porque há mutantes mortos em comum entre os conjuntos obtidos com os modelos parciais.

TABELA 5.14 -Escore de mutação das parciais S_{CoFI} – propósitos juntos.

Seqüência de teste	# c.t. + c.f.	# c.t. + c.f. distintos	# Mortos	# Vivos	Escore
Normal_prop1_prop2, ExcEspecif_prop1_prop2	**40**	39	1117	24	**0,978**
Normal_prop1_prop2	**21**	21	682	459	**0,597**
ExcEspecif_prop1_prop2	**19**	19	1028	113	**0,900**
Furtivo_prop1_prop2	**54**	54	599	542	**0,524**

5.3 Estudo de Caso 3: Protocolo de Comunicação entre Centro de Controle e Estações Terrenas

Nesta Seção é apresentado um estudo de caso de geração da seqüência de teste CoFI, para o protocolo de comunicação entre um Centro de Controle e uma estação terrena, definido pelo Centre Nationalle D'Etude Spatialle (CNES). Esse protocolo estabelece a comunicação entre dois sistemas em solo, por isso, é chamado aqui de protocolo solo-solo.

No protocolo solo-solo, vários passos da $CoFI_m$, como aqueles que criam os artefatos: casos de uso e diagramas de seqüência, não foram aplicados, uma vez que a especificação deste protocolo já se encontrava descrita via um autômato. A especificação do autômato representado em diagrama de estados encontra-se na Tabela de Transições (estados/eventos) do Anexo A.

O trabalho de mapeamento constou da transcrição do conteúdo da tabela para uma máquina de estados, i.e., criação do Modelo Total (ver FIGURA 5.6).

Para a geração da seqüência CoFI, foram criados:

- um diagrama de estados Normal, incluindo apenas as transições com entradas corretas (ver FIGURA 5.7);
- um diagrama de estados contendo as transições excepcionais especificadas, FIGURA 5.8, e;
- um diagrama com os caminhos furtivos (ver FIGURA 5.9).

Modelos TF não foram criados, por um lado, porque as implementações em solo não estão sujeitas à radiação e, por outro, porque falhas de atraso de perda já estão contempladas na especificação. Cada uma das máquinas foi submetida a Condado e os conjuntos de teste gerados foram analisados quanto ao escore de mutação, como nos estudos anteriores.

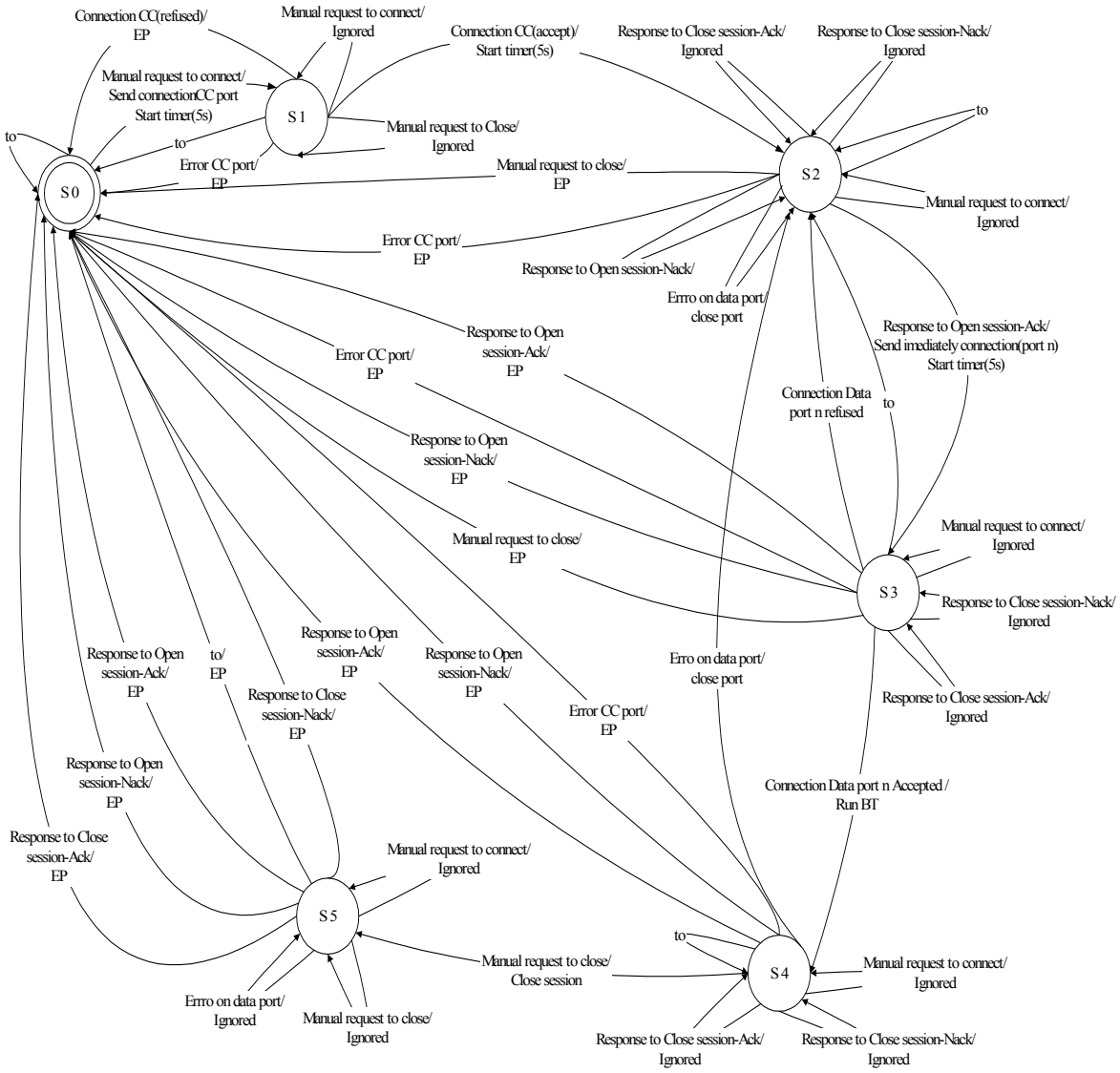


FIGURA 5.6 - Diagrama de Estados Total protocolo solo-solo.

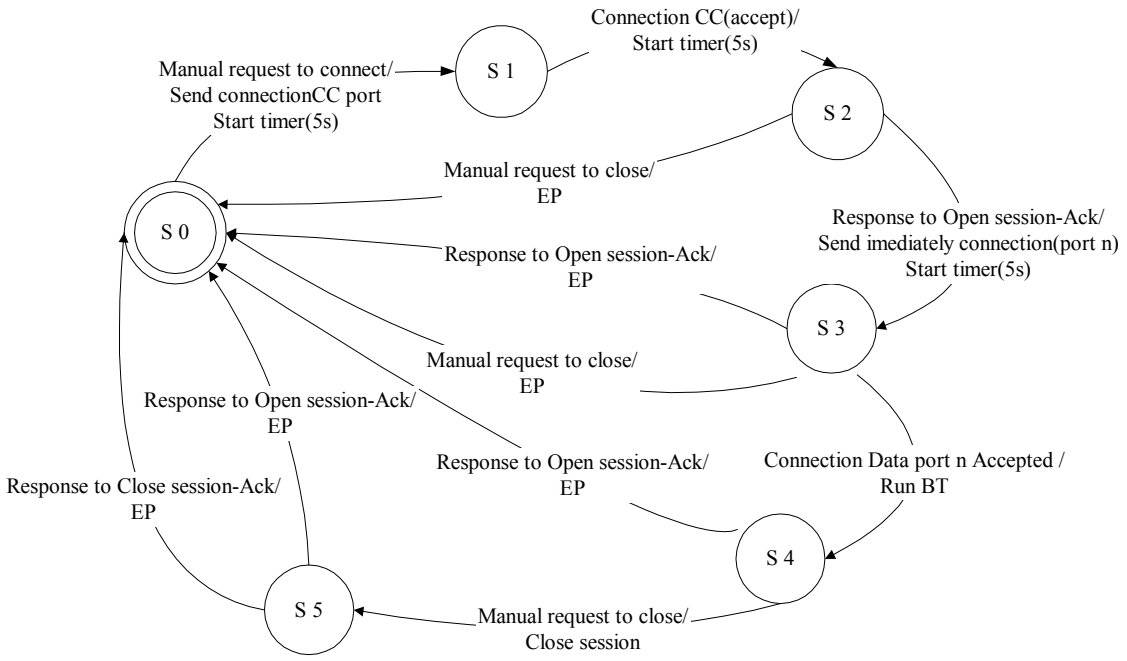


FIGURA 5.7 - Diagrama de Estados *Normal* protocolo solo-solo.

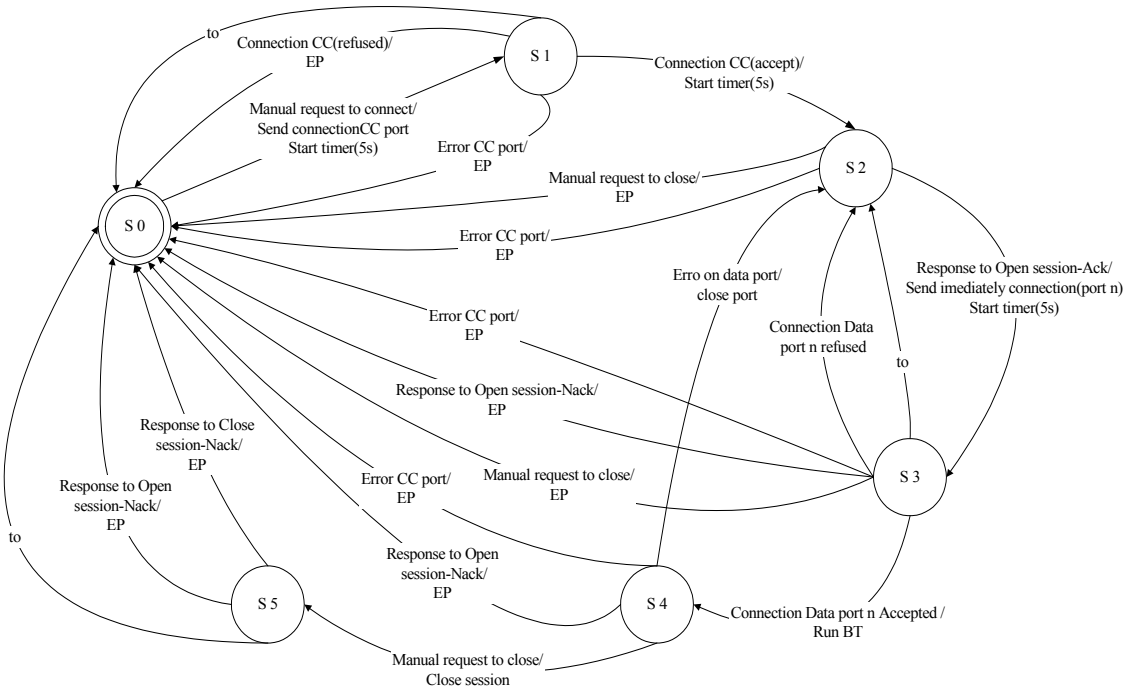


FIGURA 5.8 - Diagrama de estados *Exceções Especificadas* protocolo solo-solo.

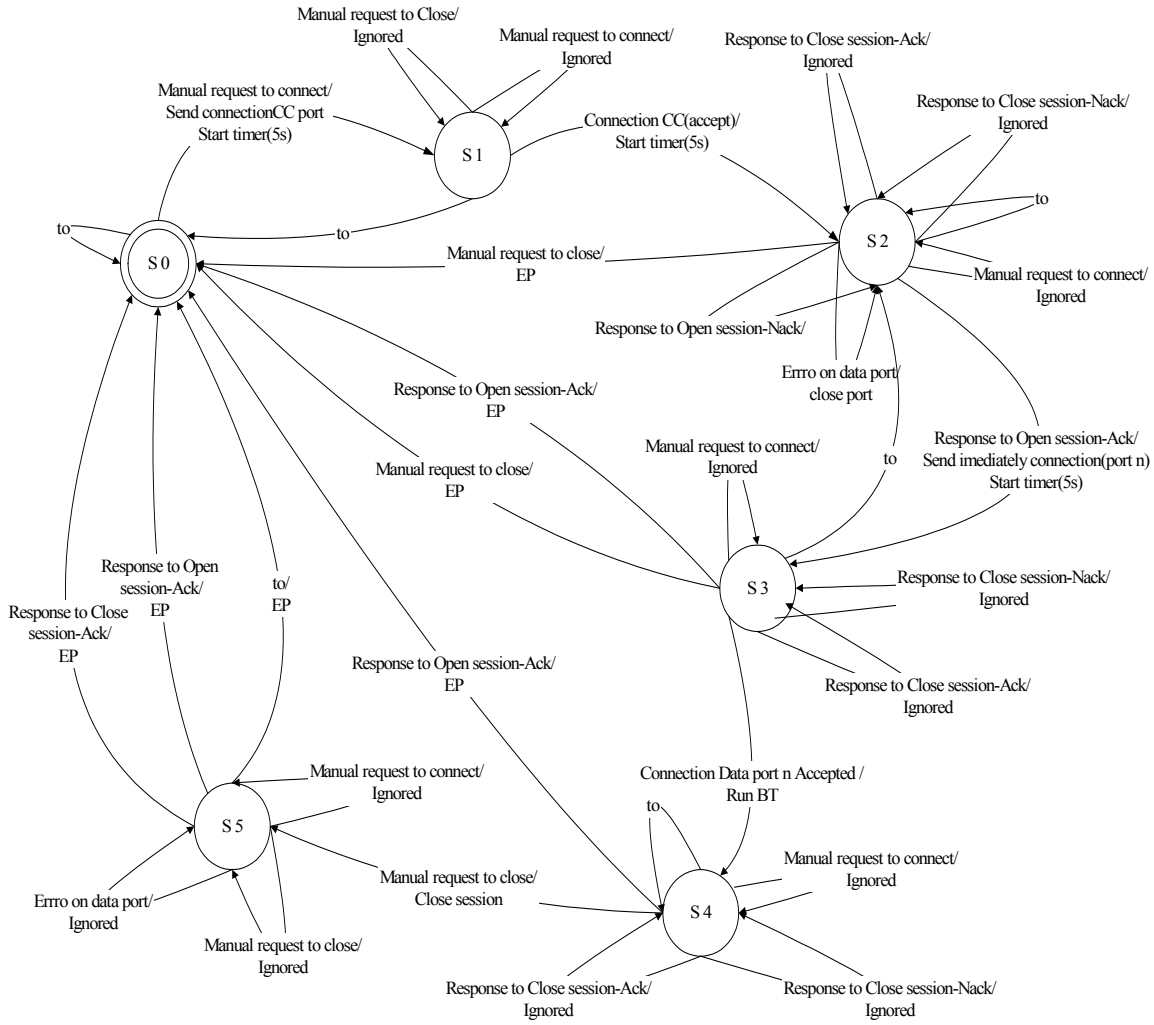


FIGURA 5.9 - Diagrama de estados *Exceções Futivas* protocolo solo-solo.

A união dos casos de teste dos modelos: normal, exceções-especificadas e caminhos furtivos constituem a seqüência S_{CoFI}. A TABELA 5.15 apresenta a cardinalidade dos modelos Total e Parciais. Nesse exemplo, o tamanho da seqüência S_{CoFI}, corresponde a 46,6 % do tamanho da seqüência S_{MT},.

Para análise da eficácia da S_{CoFI} , 441 mutantes gerados a partir do Modelo Total foram utilizados. Os operadores de mutação usados são apresentados na TABELA 5.16.

TABELA 5.15 - Cardinalidade dos modelos Total e Parciais protocolo solo-solo.

Modelos		**Tamanho do Modelo**			**Tamanho da Seqüência**	**Comprimento dos casos de teste**	
		#estados	**# transições**	**#entr dist**	**# c.t. + c.f.**	**maior**	**menor**
Normal		6	11	6	6	6	2
Excep cionais	Especificadas	6	21	12	163	13	2
	Furtivos	6	33	10	50	26	1
Soma dos parciais		6	-	13	**219**	**26**	**1**
Modelo Total		6	44	13	**470**	**17**	**1**

TABELA 5.16 - Mutantes gerados a partir do Modelo Total protocolo solo-solo.

Operadores	**Número de mutantes**
Altera saída (OutAlt)	234
Exclui saída (OutDel)	39
Exclui transição (TraArcDel)	18
Altera evento (TraEveAlt)	99
Exclui evento (TraEveDel)	46
Altera estado inicial (TraIniStaAlt)	5
	441

A TABELA 5.17 traz os resultados da avaliação quanto à adequação aos mutantes gerados. Observa-se que o escore de mutação é baixo para ambas as seqüências de teste: S_{CoFI} e S_{MT}. Esse fato deve-se a uma característica do algoritmo implementado na ferramenta Condado, que visa apenas cobrir os caminhos da especificação e não necessariamente todo tipo de erro estrutural do modelo comportamental. De qualquer forma, a relação das seqüências S_{CoFI} e S_{MT} aponta vantagens para a S_{CoFI}, pois, com 46,6% do tamanho, atinge 99% da capacidade de adequação.

TABELA 5.17 - Escore de mutação das Seqüências S_{MT} e S_{CoFI} protocolo solo-solo.

Seqüência de teste	# c.t. + c.f.	# c.t. + c.f. distintos	# Mortos	# Vivos	Escore
S_{MT}	**470**	470	232	209	**0,526**
S_{CoFI}	**213**	213	230	211	**0,521**

5.4 Considerações Finais

Este Capítulo desenvolve uma comparação empírica entre a seqüência de teste gerada segundo a metodologia $CoFI_m$ e a seqüência de teste criada segundo a metodologia N+ (Binder, 2000). O exemplo usado para essa comparação, a partir do qual ambas a seqüências foram desenvolvidas, foi uma versão simplificada do protocolo OBDH-EXP (o mesmo que ilustrou a descrição detalhada dos passos da metodologia no Capítulo 4).

Outro tipo de avaliação desenvolvida para a seqüência CoFI baseou-se no critério de adequação a falhas estruturais em máquina de estados finita. A plataforma PLAVIS apoiou essa avaliação.

Um diagrama de estados, chamado Modelo Total, contendo todas as entradas e saídas especificadas, foi criado para a especificação do sistema em teste. A partir do modelo Total, foram criados os mutantes que serviram para obter as métricas das seqüências CoFI.

Os casos de teste foram gerados com a Condado, que aceita modelos parcialmente especificados. Conseqüentemente, tem-se, por um lado, que a seqüência de teste CoFI é considerada de *conformidade fraca* (ela não cobre todos os eventos possíveis uma vez que a especificação não fora completa), por outro lado, este fato viabiliza tanto a criação de testes com antecedência no ciclo de desenvolvimento quanto o tratamento de especificações parciais, reduzindo a complexidade na atividade do projeto de casos de teste.

Os modelos de estado criados de acordo com os propósitos de teste, poderiam corresponder a modelos criados pela seleção de estados e de transições do modelo Total.

Nesta seleção, cada modelo possui um sub-conjunto do conjunto de todos os estados especificados. A seqüência de teste gerada pela união das seqüências parciais, neste caso, possui uma eficácia reduzida quando comparada com a seqüência de teste gerada a partir do modelo Total.

Na modelagem em que se separa apenas o comportamento normal do comportamento excepcional, cada modelo possui todos os estados. Neste caso, há uma seleção apenas das transições, pois, todos os estados estão incluídos em cada modelo. Neste tipo de modelagem, a eficácia da seqüência de teste composta pela união das seqüências parciais não é reduzida com relação a seqüência gerada a partir do Modelo Total.

CAPÍTULO 6

TRABALHOS CORRELATOS

Este Capítulo apresenta uma discussão sobre os trabalhos relacionados com esta tese. Um paralelo entre as características dos trabalhos relacionados e as soluções apresentadas na abordagem CoFI, são traçadas.

A abordagem CoFI tem como fundamentos as pesquisas na área de teste de conformidade de protocolos, todavia, faz uso de conceitos de validação experimental por injeção de falhas, implementadas por software e, tem pontos em comum com metodologias de teste de software que vêm sendo recentemente propostas na literatura dentro da linha de engenharia de software.

As metodologias específicas para teste de conformidade de protocolos têm sido pesquisadas desde a década de noventa. Duas abordagens de teste de protocolos assemelham-se à $CoFI_m$, à SAMSTAG (Grabowski, 1994), para teste de conformidade e à metodologia TOP de Koné (2003), voltada para teste de interoperabilidade, pois são ambas baseadas na IS-9646.

A metodologia denominada SDL *and* MSC *based test case generation* (SAMSTAG) (Grabowski, 1994), tem como meta a geração automática de casos de teste abstratos para protocolos de telecomunicação. Dois artefatos, formalmente definidos, são considerados como entrada: a especificação do protocolo, escrita em *Specification Description Language* (SDL) (IUT, 1999) e a descrição dos propósitos de teste, dados em *Message Sequence Chart* (MSC) (ITU, 1996). O método é fortemente baseado nas regras da norma IS-9646 e o estende com a formalização dos propósitos de teste e a definição de relações entre os propósitos de teste, a especificação e os casos de teste. Tanto a SAMSTAG quanto a $CoFI_m$ são fundamentadas nas atividades do processo de teste de conformidade da IS-9646. Na $CoFI_m$, a arquitetura *Ferry-injection* é proposta em substituição aos métodos de teste, as linguagens SDL e MSC não foram usadas; ao contrário, foram adotados artefatos da UML. Os propósitos de teste orientam a criação dos vários modelos (passos) e são aceitos em notação textual.

Em (Koné, 2003) é proposto um método para geração de casos de teste de interoperabilidade, denominado TOP. Neste método os testes são derivados da especificação, em máquinas de estados, das entidades que se comunicam. Caminhos combinados de ambas as máquinas representam casos de teste. Assim que cada caso é criado ele é executado de forma a evitar a criação do grafo do comportamento global e, conseqüentemente, a explosão de estados. Esse tipo de solução é chamado, em inglês, *on-the-fly*. A $CoFI_m$ não cobre interoperabilidade e os casos de teste são gerados todos inicialmente e executados posteriormente.

Metodologias de teste de software, em geral, têm sido alvo de pesquisas mais recentes do que as abordagens para teste de protocolos de comunicação. Dentre as metodologias de teste de software estudadas que possuem semelhanças com a $CoFI_m$, estão a N+ (Binder, 2000), a Scent (Ryser e Glinz, 2000), a TOTEM (Briand e Labiche, 2002), a Pluto (Bertolino e Gnesi, 2004), todas focam teste de sistema e usam artefatos UML para orientar a criação de teste.

A metodologia N+ (Binder, 2000) trata problemas de teste de sistemas orientados a objetos e descreve uma abordagem de teste semelhante à $CoFI_m$. Na N+ o comportamento de uma implementação orientada a objetos é representado em um modelo baseado em estados, sem hierarquia e sem paralelismo, denominado modelo *Flattened Regular Expression* (FREE). A metodologia que orienta a definição de casos de teste é semelhante à $CoFI_m$, mas não aborda falhas externas, nem injeção de falhas, nem propõe uma arquitetura de teste. Essa metodologia foi usada para comparação com a $CoFI_m$, ver Capítulo 5.

A metodologia *SCENario-based validation and Test of software* (SCENT) (Ryser e Glinz, 2000), orienta a criação de casos de teste a partir de cenários que capturam seqüências de interações entre o sistema e o usuário. Ela consiste de três passos principais: criar cenários em linguagem natural, formalizar cada cenário em um modelo *statechart* e derivar casos de teste dos *statecharts* usando o conceito de caminhos. Um procedimento para criação e descrição dos cenários é bem estabelecido. A $CoFI_m$ usa o mesmo procedimento de descrição dos cenários usado na SCENT. Entretanto, separa

cenários normais e excepcionais, usa outros artefatos UML que reduzem o passo de formalização de uma notação para outra.

A *Testing Object-oriented systems with the unified Modeling language* (TOTEM) (Briand e Labiche, 2002) é uma metodologia para testes de sistema e, como o próprio nome indica, é voltada para sistemas orientados a objetos. Ela se baseia nos artefatos UML: diagrama de casos de uso, descrição de casos de uso, diagrama de seqüência ou de colaboração e diagrama de classes. A descrição dos casos de uso é parametrizada e pré e pós condições de operações de objetos são expressas na linguagem *Object Constraint Language* (OCL) (OMG, 2004). Assim como a TOTEM, a $CoFI_m$ desenvolve um diagrama de seqüência a partir da descrição de cenários. Na TOTEM, os diagramas de seqüências são traduzidos para expressões regulares, cujo alfabeto são métodos públicos dos objetos. Na $CoFI_m$, não importa a tecnologia usada no desenvolvimento; ela não é exclusiva para sistemas orientados a objetos como a TOTEM.

A metodologia *Product Lines Use case Test Optimization* (PLUTO) (Bertolino e Gnesi, 2004) também é baseada na derivação de teste a partir da descrição de cenários de uso; aborda questões além do projeto de casos de teste, atacando os problemas de planejamento e gerenciamento de teste para famílias de produtos. Com o conceito de família ou linha de produtos, são formalizados requisitos gerais e específicos para os produtos. Da mesma forma, os testes são preparados para a família ou para um produto específico. Essa idéia é muito próxima do conceito de seqüência abstrata e seqüência executável de teste usada na abordagem CoFI.

As metodologias de teste de protocolo e de teste de software, citadas acima, não tratam problemas causados por falha de hardware (externos) que podem ocorrer em tempo de execução. Nesta tese, a metodologia está voltada a solucionar problemas de validação de software em aplicações espaciais e, portanto, a atenção aos problemas externos é de extrema importância. A técnica que melhor atende a esse requisito é a de injeção de falhas, por isso foi adotada como um complemento à validação de conformidade.

A técnica de injeção de falhas tem sido aplicada para avaliação experimental em que o sistema é submetido a entradas selecionadas aleatoriamente a partir de modelos de falha que as simulam. Entretanto, os trabalhos de Avresky et al. (1996), Echtle e Chen (1991) e Tsai et al. (1997) apontam melhores resultados na avaliação de sistemas quando as falhas são escolhidas deterministicamente.

Echtle e Chen (1991) mostraram analiticamente que a seleção de falhas determinísticas oferece benefício com relação à injeção aleatória em teste de protocolo de comunicação tolerantes a falhas. Eles injetam as falhas nas mensagens baseados no conhecimento do grafo do fluxo de controle do programa. A injeção é guiada pelo critério de cobertura do grafo. Na $CoFI_m$, a geração dos casos de falha também é determinística, uma vez que, ela é orientada pela cobertura dos cenários descritos em Diagramas de Seqüência ou alternativamente pela cobertura dos caminhos em uma máquina de estados finita. O fluxo do programa não é conhecido, ao contrário, as injeções são baseadas no escopo da especificação.

Injeção de falhas determinística e uso de um grafo também são abordados em Avresky. Em (Avresky et al., 1996), uma árvore de execução é montada para mapear todas as possíveis execuções dos mecanismos de tolerância a falhas. Seqüências de injeções de falha são automaticamente derivadas da árvore por análise estática. Graças à representação em árvore, os caminhos da raiz até às folhas são definidos formalmente por assertivas (ou fórmulas) e as assertivas representam as entradas para a injeção de falhas. Na $CoFI_m$, a seqüência de casos de falha, incluindo os valores dos parâmetros de injeção de falhas, é derivada de diagramas de seqüência da UML. Nesses diagramas, a informação destinada aos injetores (*faultload*) e a informação destinada ao sistema alvo (*workload*) são explicitamente independentes, conforme definido nos passos 5 e 10 da $CoFI_m$. Alternativamente, os diagramas de estados correspondentes aos diagramas de seqüência podem ser usados para esse fim. Em ambos os casos algoritmos para essa derivação ainda devem ser detalhados.

Em (Chandra et al., 2004) máquinas finitas de estados são usadas para apoiar a injeção de falhas em sistemas concorrentes. O comportamento de cada parte independente é

representado por uma máquina e o comportamento global é representado por expressões manualmente definidas. A execução é automatizada pela ferramenta chamada Loki. As falhas são injetadas somente se a expressão for satisfeita. A $CoFI_m$ não trata problemas de paralelismo. Os experimentos de injeção de falhas consistem em executar cada caso de falha uma vez: o caso é projetado de forma a colocar o sistema no estado no qual a falha pode ser injetada e, sempre que possível inclui entradas que levam o sistema ao estado inicial estável.

No trabalho descrito em Patariczа et al. (2003), *Statecharts* (UML) foram usados para modelar o comportamento das classes de sistema de controle de trem. As falhas potenciais e seus efeitos são também mapeados para análise da propagação dos erros. Objetos do ambiente são modelados. Tanto o comportamento errôneo quanto o livre de erro são modelados em um único modelo *statechart*. Na $CoFI_m$, o comportamento livre de erro e o comportamento errôneo são modelados em diagramas separados, visando à geração de casos de falha com base na descrição do comportamento do sistema e do modelo de falhas físicas do seu ambiente operacional.

A principal diferença da metodologia $CoFI_m$ para as demais metodologias discutidas aqui, está no fato dela guiar a geração de casos que contemplam a ocorrência de falhas, externamente, ao sistema em teste. Ela orienta a pessoa responsável pelos testes a identificar os problemas que podem ser causados pelo ambiente (falhas de hardware). Tais falhas, em geral, são tratadas pelos mecanismos de tolerância a falhas no software espacial. Os testes voltados para as falhas externas complementam os testes projetados para identificar não-conformidades entre a implementação e a especificação.

CAPÍTULO 7

CONCLUSÃO

Neste Capítulo, são apresentadas, primeiramente, as principais contribuições da abordagem de teste CoFI e suas limitações e em seguida, as extensões futuras vislumbradas.

7.1 Contribuições

Nesta tese foi desenvolvido um processo de teste, denominado $CoFI_p$, que se baseia no *Processo de Teste de Conformidade para protocolos ISO* especificado na norma IS 9646 (ISO, 1991). No novo processo, a técnica de injeção de falhas por software foi combinada como um complemento às atividades de teste de conformidade objetivando a validação de software em aplicações espaciais. Embora ambos os tipos de validação já sejam bastante conhecidos, eles não tinham sido explorados em uma abordagem integrada para orientar a geração de casos de teste e casos de falha.

A atividade de projetar a seqüência de teste, do processo $CoFI_p$, foi estendida com a metodologia $CoFI_m$, que detalha passo-a-passo a derivação de casos de teste de conformidade e casos de falha, para validação da robustez. Os casos de falha contêm indicações que lhes permitem serem executados com a técnica de injeção de falhas por software.

Na abordagem CoFI, os testes são derivados a partir de uma especificação normatizada, assim como na IS-9646. Essa abordagem aplicada à área espacial acompanha a tendência à padronização de serviço de software em aplicações espaciais, que vem sendo realizada pela Agência Espacial Européia (ESA) e pelo *Consultative Committe for Space Data Standardization* (CCSDS). Uma vez que serviços espaciais estejam

padronizados, uma abordagem de teste de conformidade voltada para as necessidades do software em aplicações espaciais, segue a tendência na área.

Essa metodologia foi aplicada ao serviço de *verificação de telecomando* definido na norma ECSS-E-70-41A, preparado pela Agência Espacial Européia (ESA), o que possibilitou a evidência de que a definição, mesmo que abstrata, dos pontos de controle e observação, aumentariam a testabilidade dos serviços de Packet Utilization Services (PUS), sem comprometimento com as soluções de implementação.

O projeto de uma seqüência de teste a partir de uma norma de serviços espaciais, como recomendado na abordagem CoFI, pode contribuir para a melhoria da testabilidade dos documentos gerados para uma missão particular, como a Especificação Técnica. Além disso, sendo a seqüência de teste independente da implementação, ela pode ser re-usável e ganhar mais confiabilidade e eficácia, quanto mais for aplicada, contribuindo para aumentar a qualidade e reduzir os custos em uma missão espacial.

Com relação ao método de geração de casos de teste, as contribuições deste trabalho são principalmente: (i) incorporar na seqüência de testes, casos de teste que visam detectar erros no software originados por falhas provocadas pelo ambiente espacial; (ii) modelar o comportamento normal e o comportamento excepcional em modelos de máquinas de estados finita separados, e; (iii) prover fórmulas que permitem quantificar os testes permitindo não só planejamento prévio para as atividades de teste, como também, medidas para a qualidade da seqüência de teste.

A separação do comportamento em vários modelos de estados, possibilita o uso de ferramentas de geração automática de teste que não levam à explosão do número de casos de teste. As avaliações empíricas apontaram ganhos com a seqüência CoFI, com relação ao número de casos de teste e ao potencial de adequação aos critérios de falhas de transição e saída, para as aplicações estudadas.

É importante destacar, também, que a $CoFI_m$ baseia-se em ferramentas para geração de teste fundamentadas em métodos formais, entretanto, orienta a criação dos modelos formais de forma suave, através da utilização de diagramas UML. O uso de artefatos UML evita o aprendizado com notações específicas para o projeto dos testes e permite, eventualmente, a reutilização de diagramas já projetados na fase de análise e projeto do software. Outra vantagem de se ter os serviços modelados em artefatos de UML é a possibilidade de usar ferramentas computacionais para edição, verificação e validação desses modelos, em visões parciais do sistema.

Na área de desenvolvimento de software em aplicações espaciais, espera-se contribuir na validação de softwares de comunicação, sejam desenvolvidos no INPE, em empresas ou agências que tenham as mesmas necessidades.

7.2 Limitações

Embora a abordagem CoFI seja simples e a comparação de sua seqüência de teste tenha mostrado bons resultados nos estudos de caso empíricos, alguns passos da metodologia podem ser muito extensivos se realizados manualmente. Além disso, para algumas aplicações, a geração de casos de teste a partir de uma especificação incompleta pode levar a um escore de mutação insatisfatório. Experimentos novos são necessários para uma avaliação mais precisa.

Outra limitação é com relação ao uso da máquina de estado clássica usada para geração automática de casos de teste. As máquinas clássicas requerem modelos de estados maiores para representar variações de valores de dados (e não apenas os eventos de controle). A abordagem CoFI poderá ser estendida para aceitar variações do modelo de estado clássico.

7.3 Trabalhos Futuros

Os futuros desdobramentos deste trabalho podem ser classificados em quatro níveis de atuação: (i) implementação de ferramentas de apoio a outras atividades de teste do processo $CoFI_p$ além da atividade de Definir a Seqüência Abstrata de Teste; (ii) implementar tradutores das linguagens usadas nos vários passos da $CoFI_m$; (iii) avaliação da seqüência CoFI contra implementações com apoio da Arquitetura *Ferry-injection*, e; (iv) re-definição dos passos da metodologia usando-se modelos matemáticos.

Com relação ao modelo de falhas, a proposta atual abordou apenas falhas de comunicação, entretanto, primitivas de falhas de memória ou de processador poderiam ser contempladas.

No processo $CoFI_p$, nenhuma linguagem específica é recomendada para escrever a seqüência executável, desde que ela possa ser executada por computador. Todavia, cabe observar que o comitê europeu para padronização espacial (ECSS), recomenda a linguagem PLUTO, descrina na norma ECSS-E-70-32A (ECSS, 2001), para elaboração de procedimentos de teste e operações de satélites, na tentativa de fazer convergir a indústria espacial européia em torno de uma linguagem única. Roteiros gerados a partir da seqüência de teste nessa linguagem podem ser explorados futuramente.

A geração de experimentos de injeção de falhas, a partir dos modelos que cobrem o comportamento excepcional, merece investigações práticas quanto ao potencial de encontrar falhas em implementações reais. Uma pesquisa paralela e complementar a esta está em andamento para o projeto de injetores da arquitetura *Ferry-injection* para aplicações de comunicação a bordo de satélites. As falhas de comunicação podem anular completamente o sucesso de uma missão espacial. Nos estudos realizados, poucos trabalhos com injeção de falhas exploraram uso de SWIFI com modelo de falhas de comunicação em aplicações espaciais. Resultados práticos de experimentos com injeção de falhas em sistemas espaciais não foram cobertos neste trabalho, entretanto,

ele proporciona diretrizes para realização do mesmo; planejam-se tais experimentos em futuro próximo.

Em curto prazo, a abordagem CoFI será aplicada no projeto Qualidade de software embarcado em aplicações espaciais (QSEE), financiado pela FINEP em abril de 2004 e, sendo desenvolvido pelo INPE, UNICAMP e por uma empresa de desenvolvimento de software. Neste projeto outros parâmetros de avaliação da seqüência de teste bem como do processo, poderão ser contemplados.

Em longo prazo a abordagem CoFI pode-se tornar um padrão para guiar a geração de casos de teste para quaisquer softwares reativos, sejam aqueles desenvolvidos para atender as missões espaciais nacionais, aqueles implementados nas divisões da área de Engenharia e Tecnologia Espaciais ou implementados pela indústria de software nacional e validados pelo INPE, sejam aqueles não dedicados a sistemas espaciais mas que incorporem funções de comunicação e de controle.

REFERÊNCIAS BIBLIOGRÁFICAS

Alagar, V.S.; Periyasamy, K. **Specification of software systems**. New York: Springer-Verlag, 1998. 422p.

Ambrosio, A.M.; Martins E.; Mattiello-Francisco M.F.; Silva C. S.; Vijaykumar N. L. On the use of test standardization in communication space applications. In: International Conference on Space Operations, 8. (SpaceOPs), 17-21 May 2004, Montreal, Canadá. **Proceedings**... Montreal:AIAA, 2004a. Disponível em: <www.spaceops2004.org>. Acesso em: 2 sep. 2004.

Ambrosio, A.M.; Martins E.; Vijaykumar N. L.; Carvalho, S.V. **CoFI:** conformance and fault injection – a testing process including test and fault cases derivation for space application software validation. São José dos Campos: INPE, 2004b. 77p. (INPE-11485-RPQ-779).

Ambrosio, A.M.; Martins, E.; Vijaykumar, N. L.; Carvalho, S.V. Systematic Generation of Test and Fault Cases for Space Application Validation. In.: Data System In Aerospace, 9. (DASIA), 30 mai – 2 jun 2005, Edinburgh, Scotland. **Proceedings**… Noordwijk: ESA Publications, 2005a. 1 CD-ROM.

Ambrosio, A.M.; Martins E.; Vijaykumar N. L.; Carvalho, S.V. A Methodology for Designing Fault Injection Experiments as an Addition to Communication Systems Conformance Testing. In: Workshop on Dependable Software - Tools and Methods, 1. (DSN), 28 jun – 1 jul 2005, Yokohama, Japan. **Proceedings**... Tóquio: IEEE, 2005b.

Anido, R.; Cavali, A R.; Lima-Jr., L.P.; Yevtushenko, N. Test suite minimization for testing in context. **Software Testing, Verification and Reliability**, v. 13, n. 3, p. 141-155. 2003.

Araújo, M.R.R. **Fsofist -** uma ferramenta para teste de protocolos tolerantes a falhas. 2000. 52p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas. 2000.

Arlat, J.; Aguera, M.; Amat, L.; Crouzrt, Y.; Fabre, J.-C.; Lapri, J.-C.; Martins, E.; Powell, D. Fault Injection for Dependability Validation: A Methodology and Some Applications. **IEEE Transactions on Software Engineering**, v. 16, n.2, p. 166-182, 1990. (ISS 0098-5589).

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.

Avresky, D; Arlat, J; Laprie, J-C; Crouzet, Y. Fault Injection for the Formal testing of Fault Tolerance. **IEEE Transations on Reliability**, v.45, n.3, p.443-455, 1996.

Baumgarten, B.; Giessler, A. **OSI conformance testing methodology and TTCN**. Amsterdan: Elsevier, 1994. 328p.

Bertolino, A.; Gnesi, S. PLUTO: a test methodology for Product Families. In: International Workshop on Software Product-Family Engineering, 5. (PFE), 4-6 November 2003, Siena, Italy. **Proceedings**… Berlin: Springer-Verlag 2004, p.181-197.

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

Blackburn, M.; Busser, R.; Nauman, A. Why model-based test automation is different and what you should know to get started. In: International Conference of Practical Software Quality and Testing, 2004 ESAT. (PSQT), 22-26 mar 2004, Washington DC. **Proceedings**... Software Productivity Consortioum, 2004. Disponível em: <http://www.psqtconference.com/2004east/tracks/Tuesday/PSTT_2004_blackburn.pdf>

Bochmann, G.; Das, A.; Dssouli, R.; Dubuc, M.; Ghedamsi, A.; Luo G. Fault Models in Testing. In: Protocol Testing Systems, 4. 15-17 Oct 1991. Leidschendam, The Netherlands, **Proceedings**... North Holland: IFIP TRANSACTIONS C-3. p.17-30. (ISBN 0-444-89517-5).

Bochmann, G.; Petrenko, A. Protocol Testing: review of Methods and Relevance for Software Testing. In: International Symposium on Software Testing and Analysis, 17-19 August 1994, Seatle, Washington, USA. **Proceedings**... Seatle: ACM Press. p.109-124.

Briand, L; Labiche, Y. A UML-based approach to system testing. **Software and Systems Modeling**, v.1, n.1, p. 10-24, 2002.

Carnegie Mellon - Software Engineering Institute – **CMU/SEI-2002-TR-012 ESC-TR-2002-012 -** Capabilitu Maturity Model Integration (CMMI), Version 1.1, March 2002.

Cavali, A.R.; Favreau, J.P.; Phalippou, M. Standardization of formal methods in conformance testing of communication protocols. **Computer Networks and ISDN Systems,** n. 29, p.3-14, 1996.

Chandra, R.; Lefever, R.M.; Cukier, M; Sanders, W.H. A global-state triggered fault injector for distributed system evaluation. **IEEE Transactions on Parallel and Distributed Systems,** v.15, n.7, p.593-605, July 2004. (ISBN: 1045-9219)

Chanson, S.T.; Lee, B.P.; Parakh, N.J.; Zeng H.X. Design and implementation of a ferry clip test system. In: Symposium on Protocol Specification Testing & Verification, IFIP 9., 06-09 June 1989, Enschede, The Netherlands. **Proceedings**... North-Holland: Elsevier. p.101-118, 1990.

Chestnutwood, M.C. Implementation Conformance Testing. In: Conformance and Combatibility of Naval Surface Warfare Center. 30 Sep 1996, Crane. **Talk…** Disponível em: < WWW.ACQ.OSD.MIL/OSJTF/PDF/ETINTRO.PDF >. Acesso em: 02 set. 2004.

Chow, T.S. Testing Software Design Modeled by Finite-State Machines. **IEEE Transactions on Software Engineering**, v.4, n. 3, p. 178-187, 1978.

Consultative Committee for Space Data Systems (CCSDS) **Standards**. Disponível em: < http://www.ccsds.org > Acesso em: 2 jun. 2004.

Consultative Committee for Space Data Systems (CCSDS). **CCSDS-201.0-B-3 -** Telecommand part 1—channel service. Blue Book. Issue 3. June 2000. Disponível em: <http://www.ccsds.org/ccsds/recommandreports.html>. Acesso em: 2 jun. 2004.

_____. **CCSDS-202.0-B-3 -**Telecommand part 2 - data routing service. Washington DC: NASA. . Issue 3. June 2001a. Blue Book.

_____. **CCSDS-203.0-B-3 -**Telecommand part 3 - data management service. Blue Book. Washington DC: NASA. Issue 2. June 2001b.

_____. **CCSDS-202.1-B-2 -**Telecommand part 2.1 – command operation procedures. Blue Book. Washington DC: NASA. Issue 2. June 2001c.

Cristian, F. Understanding fault-tolerant distributed systems. **Communications of the ACM**, v.34, n.2, fev. 1991.

Damm,W.; Harel, D. LSCS: breathing life into message sequence charts. **Formal Methods in System Design**, n.19, p. 45-80, 2001.

DeMillo, R.A. Mutation analysis as a tool for software quality assurance. In: Computer Software and Applications Conference, 4. (COMPSAC) Oct 1980, Chicago. **Proceedings**... Los Alamitos: IEEE Computer Society Press, 1980. p. 390-393.

Drabick, R. D. **Best practices for the formal software testing process**: a menu of testing tasks. New York: Dorset House Publishing, 2004. 286p. (ISBN 0-932633-58-7).

Dssouli, H.; Salek, K.; Aboulhamid, E.; En-Nouaary, A.; Bourhfir, C. Test Development for Communication Protocols: Towards Automation. **Computer Networks**, n. 31, p. 1835-1872, 1999.

Echtle, K.; Chen, Y. Evaluation of Deterministic Fault Injection for Fault-Tolerant Protocol Testing. In: Annual International Symposium on Fault-Tolerant Computing**,** 21. 25-27 June 1991, Montreal. **Proceedings**... Montreal: IEEE, 1991. p.418-425.

European Cooperation for Space Standardization (ECSS). **Standards**. Noordwijk: ESA publication Division. Disponível em: <http://www.ecss.nl/>. Acesso em: 2 jun. 2003a.

_____. **ECSS-E-70-32A** - Space engineering procedure language for user in test and operations: PLUTO. Issue Draft 5, 2001. Noordwijk: ESA publication Division. Disponível em: <http://www.ecss.nl/>. Acesso em: 6 dec. 2003.

_____. **ECSS-E-70-41A** – Ground systems and operations: telemetry and telecommand Packet utilization. January, 2003b. Noordwijk: ESA publication Division. Disponível em: <http://www.ecss.nl/>. Acesso em: 2 jun. 2003.

_____. **ECSS-E-40** Space engineering – software – principles and requirements. Part 1B, 2003c. Noordwijk: ESA publication Division. Disponível em: <http://www.ecss.nl/>. Acesso em: 6 dec. 2003.

European Telecommunications Standards Institute. **ETSI ES 2001-873-1.** Methods for Testing and Specification (MTS) - The Testing and Test Control Notation, v2.2.1, 2003.

Fabri, S.C.P.F. **A análise de mutantes no contexto de sistemas reativos: uma contribuição para o estabelecimento de estratégias de teste e validação**. 1996. Tese (Doutorado em Ciência da Computação) – Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, São Carlos. 1996.

Gonnenc, G. A method for the design of fault detection experiments. **IEEE Transactions on Computers**, C-19, p. 551-558, 1970.

Grabowski, J. **SDL and MSC based test case generation:** an overall view of the SAMSTAG method. Berne: University of Berne, 1994. 23p. (IAM-94-0005).

Hagar, J. Lessons learned from incorporation of commercial computer aided software engineering tools in a flight critical software test environment. In: Digital Avionics Systems Conference, 15. (DSAC) 27-31 Oct 1996, Atlanta. **Proceedings**... : AIAA/IEEE , 1996. p.125-130.

Holzmann, G. J. **Design and validation of computer protocols.** Englewood Cliffs: Prentice Hall, 1990. 512p. (ISBN 0-13-539834-7). Disponível em: <http://spinroot.com/spin/doc/book91.html >. Acesso em:02 sep. 2002.

Hopkroft, J.E.; Ullman, J.D. **Introduction to automata theory, languages and computation**. Reading, MA: Addison-Wesley, 1979. p.

Ihara, H. What do fatal failures in japonese programs suggest us? In: IEEE Pacific Rim International Symposium on Dependable Computing, 10. (PRDC'04), 03 - 05 mar, 2004, Papeete, Tahiti. **Proceedings...** Los Alamitos, California: IEEE computer Society, 2004. p.335. (ISBN: 0-76952-076-6)

Instituto Nacional de Pesquisas Espaciais (INPE)/ Universidade Estadual de Campinas (UNICAMP). **ATIFS project**. Disponível em: <http://www.inpe.br/atifs>. Acesso em: 5 sep. 2002.

Instituto Nacional de Pesquisas Espaciais (INPE). **EXP-OBDH communication protocol definition- a case study for Plavis**. São José dos Campos, 2005. Technical report. M13-IF-009. (to appear)

Institute of Electrical and Electronics Engineers (IEEE). **ANSI/IEEE Std.1008**. NewYork, USA. IEEE Standard for Software Unit Testing. 1986.

_____. **IEEE Std 829-1998.** Standard for software test documentation. NewYork, USA,1998.

_____. **IEEE/EIA Std-12207.0-1996.** Standard for information technology – software life cycle processes. New York, USA, 1998.

International Organization for Standardization (ISO). **IS 7498-1** - information technology - open systems interconnection - basic reference model: the basic model. Geneve, 1994.

_____. **IS 9646** - International standard conformance testing methodology and framework. Geneve, 1991.

International Telecommunication Union - Telecommunication Standardization Sector of ITU (ITU-T). **Recomendation X.290** – OSI conformance testing methodology and framework for protocol recommendations for ITU-T applications – general concepts. Geneve, Apr., 1995.

_____. **Recommendation Z.120** - Message Sequence Chart (MSC), Geneve, 1996.

_____. **Recommendation Z.100** - Specification and Description Language (SDL), Geneve, 1999.

Jagadeesan, L.J.; Porter, A.; Puchol, C.; Ramming, J. C.; Votta, L. G. Specification-based Testing of Reactive Software: A Case Study in Technology Transfer. In: International Conference on Software Engineering, 19. (ICSE) 20 May 1997, Boston. **Proceedings**... New York: ACM Press, 1997.

Karlsson, J.; Lidén, P.; Dahlgren, P.; Johansson, R.; Gunneflo, U. Using heavy-ion radiation to validate fault-handling mechanisms. **IEEE Micro**, v.14, n. 1, p.8-32, 1994.

Koné, O. An Interoperability Testing Approach to Wireless Application Protocols. **Journal of Universal Computer Science**, v. 9, n.10, p. 1220-1243, 2003.

Krüger, I.; Grosu, R.; Scholz, P.; Broy, M. From MSC to Statecharts. In: IFIP WG10.3/WG10.5 International workshop on Distributed and parallel embedded systems. (DIPES), 5-6 Oct 1998, ScholoB Eringerfeld, Germany, 1998. **Proceedings**... Norwell: Kluwer Academic Publishers, 1999. p.61-71.

Lai, R. A survey of communication protocol testing. **The Journal of Systems and Software**, n. 62, p. 21-46, 2002.

Larson, W.; Werts, J. **Space mission analysis and design.** Dordrecht: Kluwer Academic Publisher, 1992. 865p.

Lee, D.; Yannakakis, M. Principles and methods of testing finite state machines: a survey. **Procceedings of IEEE**, v.84, n. 8, p. 1090-1123, 1996.

Levenson, N. The role of software in recent aerospace accidents. In: International system safety conference, 19. (ISSC), 10 – 14 Sept 2001, Huntville, USA. **Proceedings**… Huntsville, USA: System Safety Society, 2001.

Madeira, H.; Some, R. R.; Moreira, F.; Costa, D.; Rennels, D. Experimental evaluation of a COTS system for space applications. In: International Conference on Dependable Systems and Networks (DSN'02), June 23 - 26, 2002, Washington, D.C., USA. **Proceedings**... Washington, D.C.: IEEE Computer Society, 2003. p.325-330.

Martins, E. **ATIFS**: um ambiente de teste baseado em injeção de falhas de softwarwe. Campinas: UNICAMP, 1995. 32p. (DCC-95-24).

Martins, E.; Sabião, S.B.; Ambrosio, A. M. ConData: a tool for automating specification-based test case generation for communication systems. **Software Quality Journal**, v.8, n. 4, p. 303-319, 1999.

Martins, E.; Mattiello-Francisco M.F.; Morais, A. N.P. Uso da ferramenta de testes FSOFIST na validação de uma aplicação espacial. In: Jornada Ibero-Americana de Engenharia de Software e Engenharia de Conhecimento, 2. (JIISIC), 30 out – 01 nov 2002, Salvador, Brasil. **Anais**... Salvador: ICMC-USP, UNIFACS, 2002. 1 CD-ROM.

Martins, E.; Ambrosio, A.M.; Mattiello-Francisco M.F. ATIFS: a testing toolset with software fault injection. In: Workshop UK Testing Research, 2. (SoftTest), 4-5 Sept 2003, York, England. **Proceedings**... York: Department of Computer Science/ University of York, 2003a.

Martins, E.; Mattiello-Francisco, M.F. A tool for fault injection and conformance testing of distributed systems. In: Latin-American Dependable Computing Symposim, 1., São Paulo, Brazil, Oct 2003. **Lecture Notes in Computer Science**. Berlin: Springer Verlag, p. 282-302, 2003b.

Mazza, C. Standards: the foundations for space IT. In: Workshop on Space Information Technology in the 21st Century, 27 Sept 2000, Darmstadt, Germany. **Talk**… Disponível em:

<www.esoc.esa.de/pr/documents/workshops/it_2000/it_in_future/esa_c_mazza.ppt >. Acesso em: 02 Sept. 2002.

Mattielo-Francisco, M.F. **Sistemas computacionais em aplicações espaciais**. São José dos Campos: INPE, 2003. 17 p. (INPE-9604-PUD/125).

Melton, B. Electrical AIT and EGSE. In: ESA Space System Design, Verification and AIT Workshop, 2., 15-16 Apr 2003, Noordwijk, The Netherlands, **Talk**… Disponível em: http://www.estec.esa.nl/conferences/03C28/ Acesso em: 15 Mar 2004.

Menezes, P.B. **Linguagens formais e autômatos**. Porto Alegre: Editora Sagra Luzzato, 2001. 165 p.

Merri, M.; Rüting, J.; Schurman, P. Validation of the ESA Packet Utilization Standard by object-Oriented Analysis. In: International Conference on Space Operations, 4. (SpaceOPs 96), 16-20 Sept. 1996, Munich, Germany. **Proceedings**.... Gilching: CAM GmbH. 1 CD-ROM.

Merri, M.; Melton, B.; Valera, S.; Parkes, A. The ECSS Packet Utilization Standard And Its Support Tool. In: International Conference on Space Operations, 7. (SpaceOPS02), 9-12 October 2002, Houston, USA. **Proceedings**... Houston: AIAA, 2002. (ISBN 1-56347-585-5). Disponível em: <www.spaceops2002.org/papers/spaceops02-p-t5-06.pdf > Acesso em: 02 fev. 2004.

Martins, M. Advanced Protocol Testing Methods and Tools. In: International Conference on Space Operations, 7. (SpaceOPs02), 9-12 October 2002, Houston, USA. **Proceedings**... Houston: AIAA, 2002. (ISBN 1-56347-585-5). Disponível em: <www.spaceops2002.org/papers/spaceops02-a-t1-31.pdf>. Acesso em: 02 fev 2004.

Myers, G. J. **The art of software testing**. NewYork: Wiley, 1979.

Naito, S.; Tsunoyama, M. Fault detection for seqüencial machines by transition tours. In: International Symposium on Fault Tolerant Computer Systems, 11. 1981, Portland. **Proceedings**... Portland: IEEE. p. 238-243. 1981.

Ngo, D.; Harris, M. A Reliable Infrastructure Based On COTS Technology For Affordable Space Application. In: IEEE Aerospace Conference, 10-17 Mar 2001, Big Sky, MT. **Proceedings**... Big Sky: IEEE, 2001. p. 2435 – 2441. (ISBN: 0-7803-6599-2)

Object Management Group (OMG). **UML 2.0 testing profile specification**. Disponível em:<HTTP://WWW.OMG.ORG/DOCS >. Acesso em: 10 jan. 2005.

_____. **UML 2.0 object constraint language specification**. Disponível em:<HTTP://WWW.OMG.ORG/DOCS >. Acesso em: 10 Oct. 2004.

Pataricza, A.; Majzik, I.; Huszer, G.; Varnai, G. UML-based design and formal analysis of the safety-critical railway control software module. In: Tarnai, G. and Schnieder, E. **Formal methods for railway operation and control systems**, proceedings of symposium FORMS 2003. Budapest: L'Harmattan Hongrie, 2003, p.83-90.

Paula-Filho, W. P. **Engenharia de software: fundamentos, métodos e padrões**. Rio de Janeiro: LTC Editora, 2001. 584p.

Petrenko, A.; Bochmann, G.v.; Yao, M. On fault coverage of tests for finite state specifications. **Computer Networks and ISDN Systems**, v.29, p.81-106, 1996a.

Petrenko, A.; Yevtushenko, N.; Bochmann, G.; Dssouli, R. Testing in context: framework and test derivation. **Computer Communications**, v.19, p.125-140, 1996b.

Universidade de São Paulo (ICMC-USP)/ Instituto Nacional de Pesquisas Espaciais (INPE)/ Universidade Estadual de Campinas (IC-UNCAMP). **Projeto PLAVIS** – Plataforma de Validação e Testes de Integração de Software. CNPq- Edital UNIVERSAL 01/2002 - PROCESSO 473396/2003-3. Disponível em: < http://www.labes.icmc.usp.br/plavis/index.html >. Acesso em: 29 abril 2005.

Pignol, M. Overview of the methodology and tools developed for the validation of CNES fault-tolerant architectures. In: International Conference on Dependable Systems and Networks (DSN'04), June 2004, Florence, Italy. **Proceedings**... Los Alamitos: IEEE Computer Society – Supplemental Volume, 2004. p.144-145.

Powell, D. **Delta-4:** a generic architecture for dependable distributing computing. Berlin: Springer-Verlag. 1991. 484p. ESPRIT Project 818/2252 Delta-4 Volume1. (ISBN-3-540-54985-4).

Pradhan, D.K. **Fault-tolerant computer system design**. London: Prentice Hall International Limited, 1995. 550p. (ISBN-0-13-057887-8).

Reed, P.R.Jr. **Desenvolvendo aplicativos com visual basic e UML**. Tradução Mario Moro Fecchio. São Paulo: Makron Books, 2000. 462p. Tradução de: Developing Applications with Visual Basic and UML. (ISBN-88.346.1198-X) .

Rocha, A.R.C.; Maldonado, J.C.; Weber, K.C. **Qualidade de Software**: teoria e prática. São Paulo: Prentice Hall, 2001. 303p. (ISBN 85-87918-54-0)

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. **Object-oriented modeling and design**. New York: Prentice Hall International. 1991. 500p. (ISBN-0-13-630054-5).

Ryser, J.; Glinz, M. **SCENT**: a method employing scenarios to systematically derive test cases for system test. Zürich: Institut für Informatik, Universität Zürich, 2000. 106p. (Technical report 2000/03).

Sabião, S.B. **Um modelo para Geração de teste baseado em máquina finita de estado estendida combinando técnicas de teste caixa-preta**. 1998. 106p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas. 1998.

Sabnani, K.; Dahbura, A.T. A protocol testing procedure. **Computer Networks and ISDN Systems**, v.15, p.185-297, 1988.

Sommerville, I. **Software engineering**. 6 ed. Harlow, England: Addison-Wesley, 2001. 693 p. (ISBN-0-201-39815-X).

Stefani, M.R. **Análise de traço com geração de diagnóstico para testes de comportamento de uma implementação de protocolo de comunicação em presença**

de falhas. 1997. 97p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas. 1997.

Tretmans, G.J. **A Formal approach to conformance testing**. 1992. 267 p. PhD thesis. University of Twente, Den Haag, The Netherlands. 1992. (ISBN 90-9005643-2).

Tretmans, J.; Belinfante, A. Automatic Testing with Formal Methods. In: Conference on Software Testing Analysis and Review, 7. (EuroSTAR´99), 8-12 Nov.,1999, Barcelona, Spain. **Proceedings**... Galway: EuroStar Conferences, 1999.

Tretmans, J. **An overiew of OSI conformance testing**. University of Twente, Jan, 2001. 14 p. Disponível em: < www.cs.ru.nl/~tretmans/testtechnieken/iso9646.ps > Acesso em: 02 fev. 2002.

Tsai, T.K.; Upadhyaya, S.J.; Zhao, H.; Hsueh, M.C. Path-based fault injection. In: Conference on Reliability and & Quality in Design, 3. (ISSAT), 1997. Anaheim, Ca. **Proceedings**... Anaheim, Ca: ISSAT, p. 121-125, 1997.

Ural, H. Formal methods for test sequence generation. **Computer Communication**, v.15, n.5, p.311-325, june 1992.

Vardanega, T. ; David, P. ; Chane, J-F. ; Mader, W. ; Messaros, R. ; Arlat, J. On the development of fault-tolerant on-board control software and its evaluation by fault injection. In: International Symposium on Fault Tolerant Computing, 25. (FTCS), 27-30 June 1995, Pasadema, USA. **Proceedings**... Los Alamitos: IEEE Computer Society, 1995. p. 510-515. (ISBN 0-8186-7079-7).

Voas, J. M.; McGraw, G. **Software fault injection**: inoculating programs against errors. New York: John Wiley & Sons, INC. 1998. 353p. (ISBN 0-471-183810-4).

Zeng, H. X.; Rayner, D. The impact of the Ferry concept on protocol testing. In: International Conference on Protocol Specification, Testing and Verification, 5., June 10-13, 1985, Toulouse-Moissac, France. **Proceedings**... North-Holland: IFIP WG6.1. 1985. p. 519–531. (ISBN:0-444-87881-5).

APÊNDICE A

IS-9646: METODOLOGIA E ARCABOUÇO PARA TESTE DE CONFORMIDADE DE PROTOCOLOS ISO

IS-9646: *Conformance Testing Methodology and Framework* é uma norma da *International Organization for Standardization* (ISO), que orienta o teste de conformidade de protocolos especificados por esse mesmo orgão. O resumo dessa norma, apresentado neste apêndice, foi preparado com base na própria descrição da norma publicada pela ITU-T (ITU, 1995) e nas referências Baumgarten (1994) e Tretmans (2002).

Baseado no conceito de *teste de conformidade* (i.é, "checar, através de testes se uma implementação de um protocolo padronizado está de acordo com sua especificação"), a norma IS-9646 define uma metodologia para teste, uma terminologia correspondente, uma linguagem de especificação de teste, provê um arcabouço para orientar a especificação de uma seqüência abstrata de testes de conformidade, orienta a realização dos meios (recursos) de teste, a execução dos testes e a análise dos resultados, assim, provendo um guia para todas as atividades de um processo de avaliação de conformidade de uma implementação de um protocolo do padrão ISO com relação à sua especificação. A norma IS-9646 é composta de sete partes, cada uma definindo um aspecto do teste de conformidade:

- parte 1: introdução e conceitos gerais;
- parte 2: descrição do processo de especificação de uma seqüência abstrata de teste para protocolos ISO;
- parte 3: definição da notação *Tree and Tabular Combined Notation (TTCN)* para descrever os casos de teste abstratos;
- parte 4: regras para execução dos testes;

- parte 5: descrição dos requisitos para os laboratórios de teste e seus clientes no processo de avaliação da conformidade*;*

- parte 6: definição de questões sobre especificação de testes para perfis de protocolos e;

- parte 7: orientações para realização de declarações de conformidade da implementação, como questionários, *templates*, etc.

A-1 Certificação de Protocolos

A norma IS-9646 define esquemas de teste e certificação bem aceitos e efetivos, e seu propósito é evitar repetições caras ou variações de teste de conformidade da mesma implementação. Para isso ela incorpora, em grande parte, as necessidades práticas e experiências de especialistas em questões concretas de teste de protocolo, proporcionando aceitação dos resultados de teste produzidos por diferentes laboratórios (testadores).

Três entidades estão envolvidas no processo de certificação: a Organização de Padronização, o Laboratório de Teste e o Cliente, como mostra a FIGURA A.1. A organização de padronização define as especificações do produto e as proformas (questionários) a serem preenchidas pelos clientes. Esses questionários visam deixar os requisitos de conformidade mais claros e não ambíguos. O cliente, quem implementou o produto a ser testado, preenche as proformas: *Protocol Implementation Conformance Statement* (PICS) e *Protocol Implementation extra Information for Testing* (PIXIT). Finalmente, o Laboratório de teste conduz a avaliação de conformidade da *Implementation Under Test* (IUT) e gera o Relatório do Teste de Conformidade do Protocolo (*Protocol Conformance Test Report* - PCTR). O papel do cliente e do laboratório de teste e os documentos a serem gerados em todas as fases de teste, são especificados na norma, mas não são detalhados aqui.

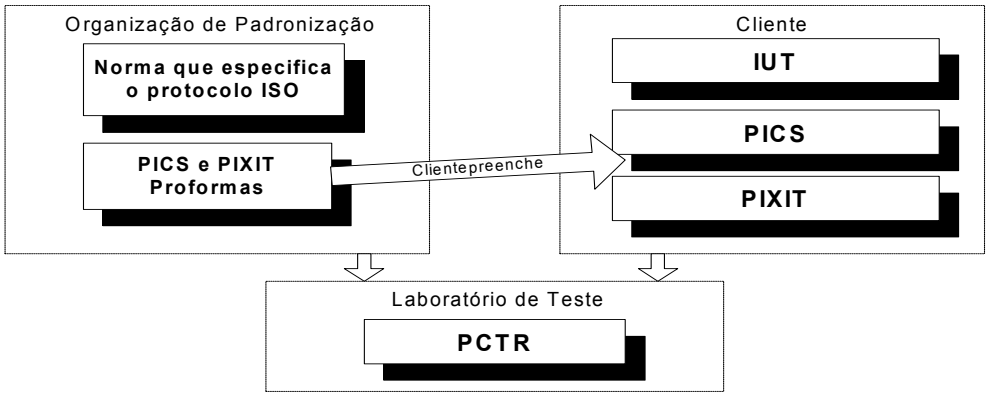


FIGURA A.1 - Entidades no processo de certificação de protocolos ISO.

A-2 Processo de Teste de Conformidade ISO

As atividades para realização dos testes de conformidade estão organizadas no processo de teste de conformidade, o qual possui três fases: Preparação para o teste, Operações dos testes e, Produção do relatório de Teste.

A FIGURA A.2 ilustra as fases e atividades do processo, cujas principais definições e siglas são apresentadas na TABELA A.1.

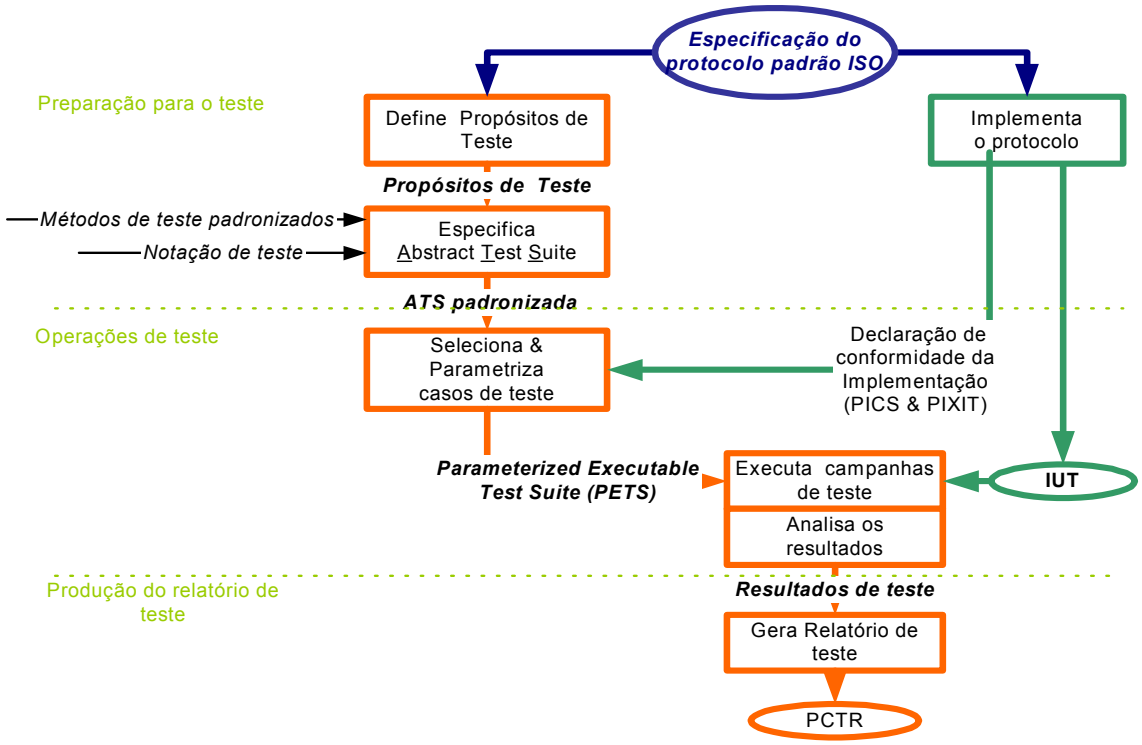


FIGURA A.2 - Processo de teste de conformidade da ISO.

TABELA A.1 - Conceito e siglas: teste de conformidade ISO.

Abstract Test Suite **(ATS)**	Conjunto de casos de teste abstratos. Um caso de teste abstrato é uma especificação completa de ações necessárias para se atingir um propósito de teste.
Prot. Impl. Conformance Statement **(PICS)**	Declaração feita pelo fornecedor da uma implementação listando todas as capacidades que foram implementadas.
Protocol Implementation extra Information for Testing **(PIXIT)**	Informações sobre o SUT (como endereços, etc.); parâmetros ou temporizadores, capacidades testáveis e não testáveis da IUT; informações administrativas.
Parameterized Executable Tests Suite **(PETS)**	Conjunto selecionado de casos de teste, no qual todos os casos foram parametrizados de acordo com o PICS.
Protocol Conformance Test Report **(PCTR)**	Documento que fornece detalhes dos testes executados para uma ATS particular. Ela lista todos os casos de teste abstratos e identifica aqueles para os quais a versão executável foi rodada e declara o veredito do teste.

A especificação (padrão) do protocolo, a qual inclui requisitos de conformidade dinâmicos e estáticos, guia ambas as tarefas, de teste (à direita da Figura A.2) e de implementação (à esquerda). A declaração de conformidade da implementação do protocolo (PICS) e as proformas de informações extras sobre a implementação (PIXIT) também são entradas para o processo de teste. As proformas e declarações devem ser

precisamente preenchidas com detalhes da implementação (IUT) antes de serem entregues ao Laboratório de Teste.

A-3 Descrição das Fases do Processo de Teste

A primeira fase, ***Preparação para os testes,*** inicia com a definição dos *propósitos de teste* baseados nos requisitos de conformidade declarados na especificação do protocolo padrão ISO. Não há regras para definição dos propósitos de teste, eles são definidos com base na experiência das pessoas que preparam os testes. No próximo passo, derivam-se casos de teste genéricos, os quais devem descrever as ações necessárias para alcançar os propósitos de teste sem considerar o ambiente no qual o teste será realmente executado. (A norma não orienta como derivar os propósitos de teste nem como derivar os casos de teste). Em seguida, cada caso de teste genérico é convertido em um caso de teste abstrato. Nessa atividade, um *método de teste* particular e as restrições implicadas no ambiente devem ser consideradas. O conjunto completo de casos de teste compreende a *Abstract Test Suite* (ATS). Os casos de teste são descritos na *notação* TTCN, a qual independente da implementação. Detalhes da implementação como valores de temporizadores, endereços de portas de comunicação são acrescentados aos casos de teste mais tarde. A especificação de um protocolo ISO pode ter vários perfis diferentes. Uma ATS é definida para cada perfil. A ATS é independente da implementação, mas, ela é associada a um dos 4 *métodos de teste* padronizados na IS-9646, (ver Seção A.4).

Uma vez que a ATS está pronta, a fase de ***Operação dos testes*** pode começar. Nessa fase, os casos de teste são selecionados a partir da ATS de acordo com as informações do PICS e depois parametrizados com as informações providas nos documentos PICS e PIXIT. Assim, a ATS é transformada na PETS levando em consideração o ambiente onde a IUT será executada. Uma vez que os PETS estão completos, as campanhas de teste são iniciadas e os resultados obtidos são analisados.

A última fase, ***Produção do relatório de teste***, consiste da preparação do relatório de conformidade PCTR. Durante as campanhas de teste o comportamento da IUT é observado e registrado. Para cada caso de teste um dos seguintes resultados é possível:

passou, falhou, inconclusivo, erro no caso de teste (abstrato ou executável), término anormal do caso de teste. Todos os resultados são analisados e o veredito sobre a conformidade da IUT com respeito a especificação do protocolo (os requisitos de conformidade) são atribuídos

A-4 Os Métodos de Teste

Um *método de teste* define como a IUT deve ser testada. Ele descreve uma arquitetura abstrata consistindo de uma configuração das funções (LT, UT, LTCF and TCP) aplicáveis a um contexto, indicando quais funções são desempenhadas pelo Sistema de Teste e pelo Sistema em Teste. O contexto corresponde ao número de canais de comunicação requeridos pela IUT. O contexto pode ser: *single-party* (um canal); *multi-party* (vários canais). A TABELA A.2 descreve as siglas relacionadas aos métodos de teste. O método de teste representa também um modelo de acessibilidade da implementação ao sistema de teste, em termos dos PCOs e dos eventos de teste (ASPs e PDUs).

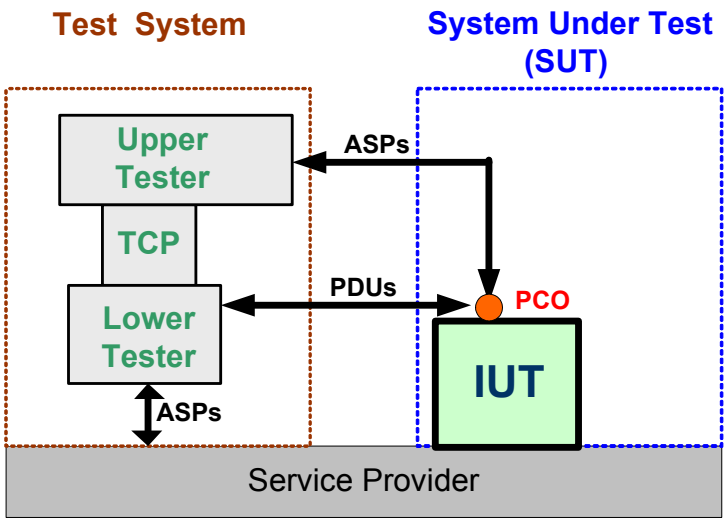


FIGURA A.3 - Método de teste Local no contexto *single-party*.

A IS-9646 define 4 tipos de métodos de teste: local, distribuído, coordenado e remoto. Por exemplo, se a IUT é executada no mesmo sistema de teste ou remotamente a ele, o

método de teste é local ou distribuído, respectivamente. A título de ilustração, a FIGURA A.3 mostra o método de teste local no contexto *single-party*.

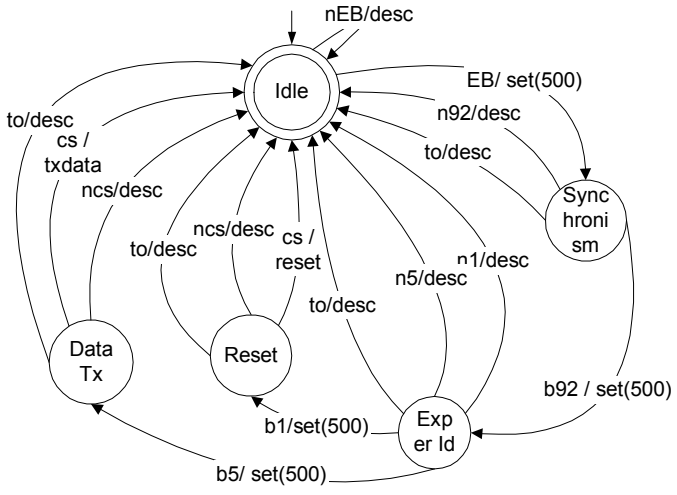
TABELA A.2 - Siglas relacionadas aos métodos de teste.

LT	*Lower Tester* – testador superior
UT	*Upper Tester* – testador superior
LTCF	*Lower Tester Control Function* - função de controle do testador inferior
TCP	*Test Coordination Procedure* – procedimento de coordenação dos testes
PCO	Ponto de Controle e Observação. Um ponto dentro do ambiente de teste onde a ocorrência de um evento de teste deve ser controlada e observada. Caracterizado por um conjunto de ASPs e/ou PDUs.
ASP	*Abstract Service Primitive* – primitiva abstrata de serviço. Uma descrição independente da implementação de uma interação entre um provedor e um usuário do serviço.
PDU	*Protocol Data Unit* – unidade de dado do protocolo

APÊNDICE B

DIAGRAMAS DO OBDH-EXP 2 COMANDOS

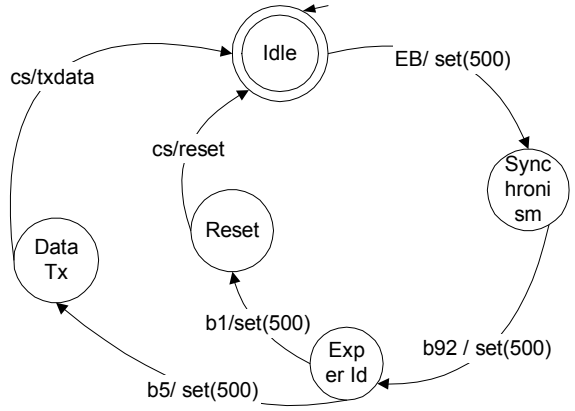
A especificação Total contendo todos os comandos especificados para o exemplo do protocolo OBDH-EXP com dois comandos é ilustrada na FIGURA B.1. Note que a máquina total não é a máquina completa (aquela que contempla a suposição completeza). Os diagramas de estados produzidos de acordo com a metodologia $CoFI_m$ para o referido exemplo são apresentados nas figuras FIGURA B.2, FIGURA B.3, FIGURA B.4, FIGURA B.5, FIGURA B.6.



Casos de teste gerados pela Condado (#c.t. = 12)		
nEB	EB b92 n5	EB b92 b1 to
EB n92	EB b92 to	EB b92 b5 ncs
EB to	EB b92 b1 cs	EB b92 b5 cs
EB b92 n1	EB b92 b1 ncs	EB b92 b5 to

Estados: 5
Transições: 16
Entradas distintas: 11

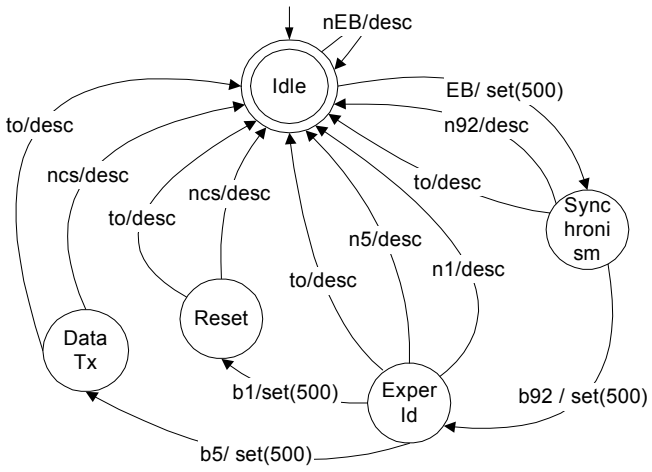
FIGURA B.1 - Diagrama de Estados Total – OBDH-EXP 2comandos.



Casos de teste gerados pela Condado (#c.t. = 2)
EB b92 b1 cs
EB b92 b5 cs

Estador: 5
Transições: 6
Entradas distintas: 5

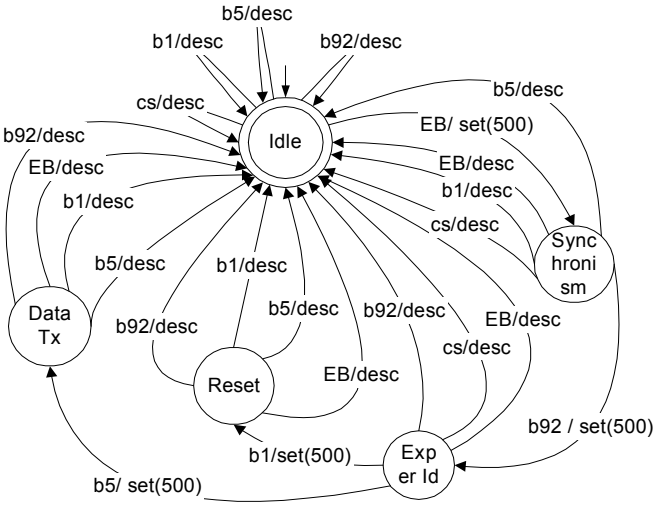
FIGURA B.2 - Diagrama de Estados *Normal* OBDH-EXP 2comandos.



Casos de teste gerados pela Condado (#c.t. = 10)		
nEB	EB b92 n5	EB b92 b5 ncs
EB n92	EB b92 to	EB b92 b5 to
EB to	EB b92 b1 ncs	
EB b92 n1	EB b92 b1 to	

Estados: 5
Transições: 14
Entradas distintas: 10

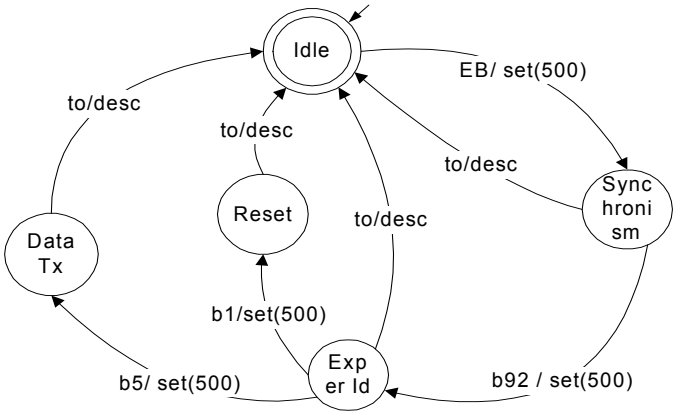
FIGURA B.3 - Diagrama de Estados *Exceções Especificadas* OBDH-EXP 2comandos.



Casos de teste gerados pela Condado (#c.t. = 19)		
cs	EB cs	EB b92 b1 b92
b1	EB b92 EB	EB b92 b5 b5
b5	EB b92 cs	EB b92 b5 b1
b92	EB b92 b92	EB b92 b5 EB
EB b5	EB b92 b1 EB	EB b92 b5 b92
EB EB	EB b92 b1 b5	
EB b1	EB b92 b1 b1	

Estados: 5
Transições: 23
Entradas distintas: 5

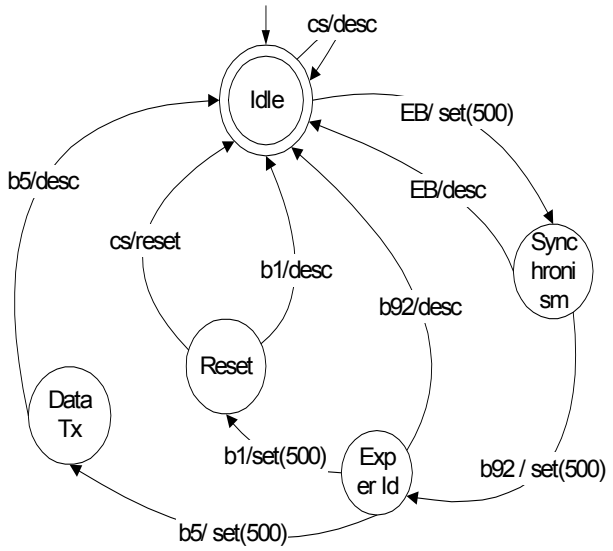
FIGURA B.4 - Diagrama de Estados *Exceções Furtivas* OBDH-EXP 2comandos.



Casos de teste gerados pela Condado (#c.t. = 4)	
EB to	EB b92 to
EB b92 b1 to	EB b92 b5 to

Estados: 5
Transições: 8
Entradas distintas: 5

FIGURA B.5 - Diagrama de Estados *Exceções TF atraso* OBDH-EXP 2comandos.



Casos de teste gerados pela Condado (#c.t. = 6)		
EB EB	EB b92 b1 cs cs	EB b92 b1 cs
EB b92 b92	EB b92 b1 b1	EB b92 b5 b5

Estados: 5
Transições: 10
Entradas distintas: 5

FIGURA B.6 - Diagrama de Estados *Exceções TF duplicação* OBDH-EXP 2comandos.

A árvore de alcançabilidade e a Matriz Resposta para o exemplo OBDH-EXP com dois comandos, geradas de acordo com a metodologia N+ são ilustradas na FIGURA B.7 e na TABELA B.1 respectivamente.

TABELA B.1 - Matriz Resposta da N+ para OBDH-Exp 2comandos.

Eventos	Condições	Estados				
		Idle	Synchronism	Exper ld	Reset	Data Tx
EB		✓	d	d	d	d
nEB		✓	d	d	d	d
b92		i	✓	d	d	d
n92		i	✓	d	d	d
b1		i	d	✓	d	d
n1		i	d	d	d	d
b5		i	d	✓	d	d
n5		i	d	d	d	d
n1n5		i	d	✓	d	d
cs		i	d	d	✓	✓
ncs		i	d	d	✓	✓
to		na	✓	✓	✓	✓

i = ignora, fica no mesmo estado;

d = descarta, volta ao estado inicial.

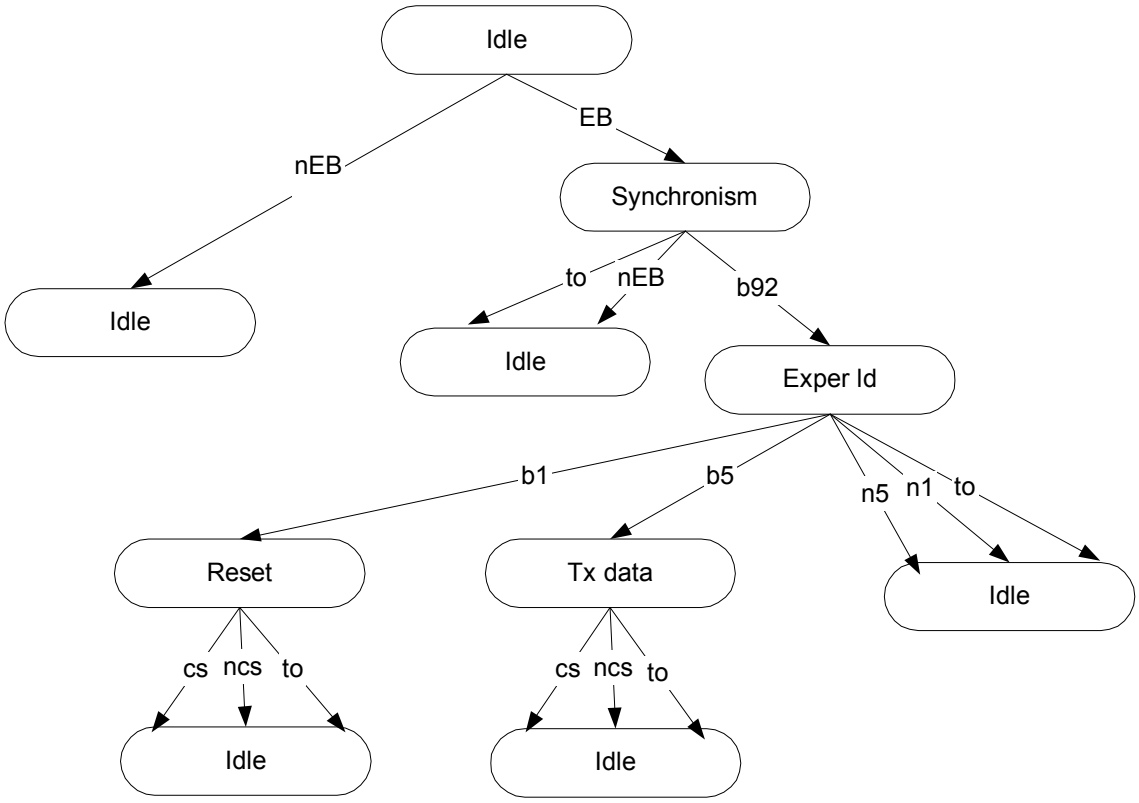


FIGURA B.7 - Árvore de alcançabilidade da N+ para OBDH-Exp 2comandos.

APÊNDICE C

MODELOS DO SERVIÇO ECSS-TCVERIFICATION

Este Apêndice apresenta, passo a passo, a geração da seqüência abstrata de teste de acordo com a metodologia $CoFI_m$ do serviço Verificação de TC especificado na norma ECSS-E-7041ª. Na modelagem desse serviço foram consideradas duas situações, uma em que propósitos de teste são modelados separadamente (em diagramas de estados distindos) e outra em que os propósitos são mapeados juntos no modelo de estados. Na primeira parte deste Apêndice, são mostrados os passos da $CoFI_m$ em que os propósitos são disjuntos, na segunda parte, são ilustrados os diagramas de estados com os propósitos de teste juntos e ainda na terceira parte, o diagrama do comportamento Total é ilustrado.

C.1 Modelagem Passo-a-Passo do Serviço

Passo 1: identificar um serviço e associá-lo a um ou mais propósitos de teste.

O serviço Verificação de Telecomando foi selecionado por ser um serviço básico usado por outros serviços da norma. Dois propósitos de teste foram definidos para ele.

Serviço: Verificação de Telecomando

Propósitos de teste:

1) testar a corretude do telecomando recebido e a viabilidade de sua execução

2) testar a geração de *reports* sobre a execução do TC, conforme solicitado pelo sistema de solo.

Passo 2: Identificar usuários e o meio físico de comunicação.

Usuários: Processos aplicativos do Sistema de solo, Sistema externo que executa o telecomando (pode ser um processo no mesmo sistema computacional ou um equipamento à parte, como por exemplo, um robô).

Meio de comunicação: Sistema de comunicação solo-bordo incluindo *links* de rádio freqüência entre antenas no sistema de solo e antenas no sistema de bordo, apoiados pelo protocolo CCSDS como ilustra a FIGURA 5.3.

Passo 3: listar entradas, saídas, arquitetura de teste e pontos de controle e observação e variáveis operacionais.

A comunicação entre o serviço e o sistema de solo é bem estabelecida na norma, através dos *reports* e *requests* transportados em pacotes de TM e TC respectivamente. Por outro lado, a comunicação com o sistema Executor do TC não é especificada, pois o executor pode ser um processo aplicativo (software), um equipamento no mesmo satélite ou um equipamento fora do satélite. Entretanto, o serviço deverá obter informações sobre a execução do TC para cumprir seu objetivo de reportar o estado dessa execução ao sistema de solo. Essa interface é suposta, então, estar definida na Especificação Técnica. Para fins de demonstração do estudo de caso, algumas suposições foram feitas e deverão ser confirmadas (ou compatibilizadas) com a Especificação Técnica da missão quando ela estiver disponível. As suposições com interfaces abstratas estão na TABELA C. 1.

TABELA C. 1 - Entradas e saídas do ECSS-Verificação de TC.

Entradas			
	Nome	*PCO*	*Descrição*
	TC	PCO_1	pacote de telecomando. Ver FIGURA 5.4.
	StartOK	PCO_3	indicação de sucesso no início da execução do TC
	PrOK(p)	PCO_3	indicação de sucesso no passo p da execução do TC
	FinOK	PCO_3	indicação de sucesso na finalização da execução do TC
	StartNOK	PCO_3	indicação de falha no início da execução do TC
	PrNOK(p)	PCO_3	indicação de falha no passo p da execução do TC
	FinNOK	PCO_3	indicação de falha na finalização da execução do TC
Saídas			
	Nome	*PCO*	*Descrição*
	TM	PCO_2	pacotes de telemetria (ver FIGURA 5.5); contendo informações (*reports)* listados na tabela TABELA 5.7.

A arquitetura *Ferry-injection* (ver Seção 3.4.2) para os testes do serviço Verificação de TC é ilustrada na FIGURA C. 1.

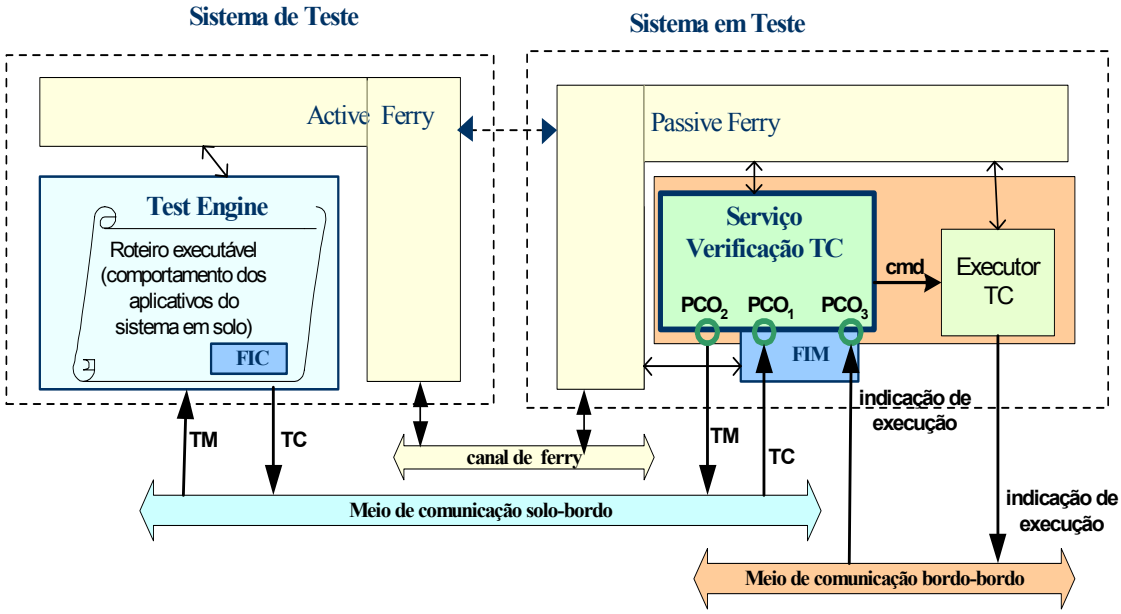


FIGURA C. 1 - Arquitetura de teste do ECSS-Verificação de TC.

Com base nessa arquitetura, os pontos de controle e observação para o teste, no nível de abstração apresentado na descrição do serviço na norma são: (i) PCO_1 - onde fluem os pacotes de TC; (ii) PCO_2 - onde fluem os pacotes de TM; (iii) PCO_3 - onde fluem as informações sobre os estágios de execução do TC. Detalhes da implementação do serviço e do Executor de TC são definidos no documento Especificações Técnicas, que trazem detalhes específicos dos equipamentos e subsistemas da missão. As variáveis operacionais que auxiliam a descrição dos campos dos pacotes e o entendimento do serviço são descritas na TABELA C. 2.

TABELA C. 2 - Variáveis operacionais do ECSS-Verificação de TC.

Variáveis Operacionais		
	Nome	*Descrição*
Campos do pacote de TC	apid	Campo do cabeçalho do pacote de TC e de TM. Identifica o processo de aplicação em bordo, que recebe ou que gerou o pacote.
	length	Campo do cabeçalho do pacote, contendo o comprimento do pacote.
	type	Campos do cabeçalho do campo de dados, indicando o tipo de serviço.
	subtype	Identifica o subtipo de serviço.
	checksum	Campo de controle de erro do pacote.
	appData	Dados para o processo de aplicação.
	Ack .bit3 (1 indica report de aceitação) .bit2 (1 indica report de inicialização) .bit1 (1 indica report de progresso) .bit0 (1 indica report de finalização)	Quatro bits do cabeçalho do campo de dados do pacote de TC, que indicam a solicitação, do sistema de solo, de reports de aceitação, inicialização acompanhamento e finalização da execução.
	p	Passo corrente da execução do TC
	f	Código da falha no estágio de execução do TC a serem especificados na missão.
	ce	Código de erro de sintaxe do pacote de TC. Especificado na norma.

Passo 4: Descrever o serviço como casos de uso.

Os casos de uso satisfazem os propósitos de teste, isto é, para cada propósito de teste foi criado um caso de uso, conforme mostrado na TABELA C. 3 e na TABELA C. 4.

TABELA C. 3 - Casos de uso do Propósito 1.

Nome: validação do TC.

Ator: sistema de solo, aquele que transmite os pacotes de TC ao sistema de bordo.

Descrição**:** O SERVIÇO recebe um pacote de TC, checa sua validade e a viabilidade de execução. Transmite um *report* de aceitação se solicitado ou um erro foi detectado.

Condição de entrada:

Chegada de um pacote de TC

Condição de saída:

Cenário Base (1):

1. o SERVIÇO checa que os campos do pacote de TC são válidos
2. o SERVIÇO checa que as condições permitem que o TC seja executado, isto é, há viabilidade,
3. o SERVIÇO identifica que AcK.bit3 = 1, isto é, há solicitação de *report* de aceitação,
4. o SERVIÇO transmite o *report* de sucesso na aceitação requisitado
5. o SERVIÇO envia o TC para execução.

Cenários alternativos:

Cenário 2:

3.a.1. o SERVIÇO identifica AcK.bit3 = 0, isto é, não há solicitação de report de aceitação,
3.a.2. o SERVIÇO dispara execução do TC,
3.a.2. encerra.

Cenário 3:

3.a.1. o SERVIÇO identifica que o TC é do tipo imediato,
3.a.2. o SERVIÇO dispara execução do TC,
3.A.3. encerra.

Cenários de exceção:

Cenário 4:
1.a.1. o SERVIÇO identifica que apid é ilegal,
1.a.2. o SERVIÇO transmite TM com report de erro na aceitação com código 0,

Cenário 5:
1.a.1. o SERVIÇO identifica que comprimento (lenght) é inválido,
1.a.2. o SERVIÇO transmite TM com report de erro na aceitação com código 1,

Cenário 6:
1.a.1. o SERVIÇO identifica checksum incorreto,
1.a.2. o SERVIÇO transmite TM com report de erro na aceitação com código 2,

Cenário 7:
1.a.1. o SERVIÇO não reconhece o tipo (type) de serviço,
1.a.2. o SERVIÇO transmite TM com report de erro na aceitação com código 3,

Cenário 8:
1.a.1. o SERVIÇO não reconhece o subtipo (subtype) de serviço,
1.a.2. o SERVIÇO transmite TM com report de erro na aceitação com código 4,

Cenário 9:
1.a.1. o SERVIÇO identifica dado da aplicação(appData) inconsistente,
1.a.2. o SERVIÇO transmite TM com report de erro na aceitação com código 5

TABELA C. 4 - Caso de uso do Propósito 2.

Nome: verificação dos estágios de execução do TC.

Ator: Sistema de solo e Executor do TC.

Descrição**:** O SERVIÇO acompanha os estágios de execução do TC e envia um report conforme requisitado no campo Ack ou conforme a informação de falha recebida do Executor.

Condição de entrada:

Chegada de um pacote correto de TC cuja execução é permitida

Condição de saída:

Cenário Base (1):

1. o SERVIÇO identifica que Ack.bit2 = 1, isto é, há solicitação de report de inicialização,
2. o SERVIÇO recebe uma indicação de sucesso na inicialização do TC (StartOK), do Executor,
3. o SERVIÇO transmite o report de sucesso na inicialização ao sistema de solo,
4. o SERVIÇO identifica Ack.bit1 = 1, isto é, há solicitação de report de progresso,
5. o SERVIÇO recebe uma indicação de sucesso no estágio p de execução do TC (PrOK(p)), do Executor,
6. o SERVIÇO transmite o report de sucesso no passo 1 ao sistema de solo,
7. o SERVIÇO identifica Ack.bit0 = 1, isto é, há solicitação de report de finalização,
8. o SERVIÇO recebe uma indicação de sucesso na finalização do TC (FinOK), do Executor,
9. o SERVIÇO transmite o report de sucesso na finalização ao Sistema de solo.

Cenários alternativos:

Cenário 2:
1.a.1. o SERVIÇO identifica que Ack.bit2 = 0, isto é não há solicitação de report de inicialização,
1.a.2. o SERVIÇO retorna ao passo 4.

Cenário 3:
5.a.1. o SERVIÇO identifica Ack.bit1 = 0 isto é não há solicitação de report de progresso,
5.a.2. o SERVIÇO retorna ao passo 7.

Cenário 4 :
8.a.1. o SERVIÇO identifica Ack.bit0 = 0 isto é não há solicitação de report de finalização,
8.a.2. o SERVIÇO encerra.

Cenários de exceção:

Cenário 5:

2.a.1. o SERVIÇO recebe do Executor uma indicação de falha na inicialização do TC (StartNOK),
2.a.2. o SERVIÇO transmite o *report* de falha na inicialização ao sistema de solo,
2.a.3. o SERVIÇO encerra.

Cenário 6:

5.a.1. o SERVIÇO recebe do Executor uma indicação de falha no estágio p de execução (PrNOK(p)),
5.a.2. o SERVIÇO transmite o *report* de falha no estágio p ao Sistema de solo,
5.a.3. o SERVIÇO encerra.

Cenário 7:

8.a.1. o SERVIÇO recebe do Executor uma indicação de falha na finalização do TC (FinNOK)
8.a.2. o SERVIÇO transmite o *report* de falha na finalização do Sistema de solo,
8.a.3. o SERVIÇO encerra.

Passo 5: definir diagrama de seqüência normal.

Entidades: Processo de aplicação solo, meio de comunicação solo-bordo, Executor de TC e meio de comunicação bordo-bordo.

As interações constam das entradas e saídas definidas no passo 3, incluindo seus parâmetros ou campos de dados. A TABELA C. 5 detalha essas interações usadas aqui.

As variações nas interações de saída, isto é, as mensagens de telemetria (TM) são identificadas pelos mnemônicos mostrados na TABELA 5.7 (ver Seção 5.2), cujos valores numéricos correspondentes são atribuídos ao campo cabDados.subtipo do pacote de TM. Aqui, a variação de valores das variáveis operacionais (passo 3) é analisada para extensão definição dos cenários definidos no passo 4, a serem escritos em diagramas de seqüência. A TABELA C. 6, traz a variação dos valores válidos da variáveis operacionais usadas aqui.

A criação dos cenários escritos em diagramas de seqüência sugere a inclusão dos parâmetros nas interações. O uso de parâmetros induz uma combinação de valores que originam outros cenários. Para controlar o número de cenários nesse passo, a combinação exaustiva de todos os valores possíveis de todas as interações é evitada da seguinte forma: (i) os cenários continuam sendo separados por propósito de teste e (ii) suposições gerais e específicas por propósito de teste, foram estabelecidas, conforme mostra a TABELA C. 7.

TABELA C. 5 - Parâmetros das interações do ECSS-Verificação de TC.

Interação	*Descrição*	*Parâmetros*
TC	chegada de pacotes de TC	(packet.011, packetId.**apid**, packetSeq, packet**Lenght**, cabDados.flag, cabDados.versão, cabDados.**Ack**, cabDados.**tipo**, cabDados. **subtipo**, **appData**, spare, **cksum**)
TM	saída de pacotes de TM gerados por algum processo de aplicação a bordo	(packet.001, packetId.apid, packetSeq, packetLenght, cabDados.versão, cabDados.tipo, cabDados.**subtipo**, cabDados.subcounter, cabDados.destination, cabDados.time, cabDados. spare, source-Data, spare, cksum).
StartOK	Chegada de informação sobre inicialização correta para execução	
PrOK	Chegada de informação sobre execução correta do estágio p	(p)
FinOK	Chegada de informação sobre finalização correta na execução	

TABELA C. 6 -Valores válidos das variáveis operacionais do ECSS-Verificação TC.

Nome	*Valores válidos que podem assumir*
Apid	faixa de valores permitidos, estabelecida na missão
Length	depende do tipo de pacote de TC.
Type	1 para o serviço Verificação de TC
Subtype	1, 2, 3, 4, 5, 6, 7, 8 permitidos para o tipo 1
Checksum	calculado segundo uma função matemática
Ack .bit3 (1 indica report de aceitação) .bit2 (1 indica report de inicialização) .bit1 (1 indica report de progresso) .bit0 (1 indica report de finalização)	0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
P	depende do tipo de TC escolhido para o teste
Ce	0,1,2,3,4,5 para erros de sintaxe.

TABELA C. 7 - Suposições para elaboração dos diagramas de seqüência normais.

Gerais: • TCs cuja execução pode ser observada são escolhidos para serem utilizados nos testes • Toda informação fluindo em PCO_3 é correta • Relações de tempo não foram consideradas • Os campos não explicitamente citados nas interações contêm valores corretos
Propósito de teste 1: • O campo Ack conterá os valores: 0001 e 0000
Propósito de teste 2: • Todos os campos do pacote de TC que fluem em PCO_1 são corretos • um TC com dois estágios de execução é escolhido

O diagrama de seqüência normal para o propósito 1 é ilustrado na FIGURA C.2 e para o propósito 2 estão nas FIGURA C.3 e FIGURA C. 4.

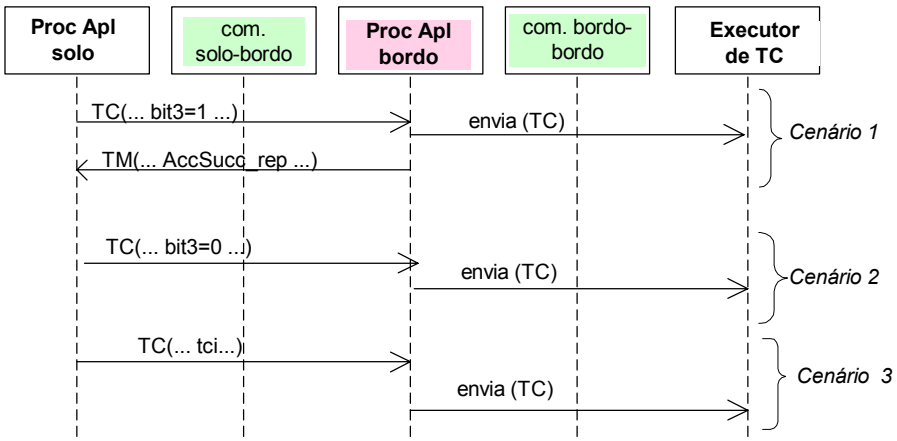


FIGURA C.2 - Diagrama de Seqüência *Normal Propósito 1* ECSS-Verificação TC.

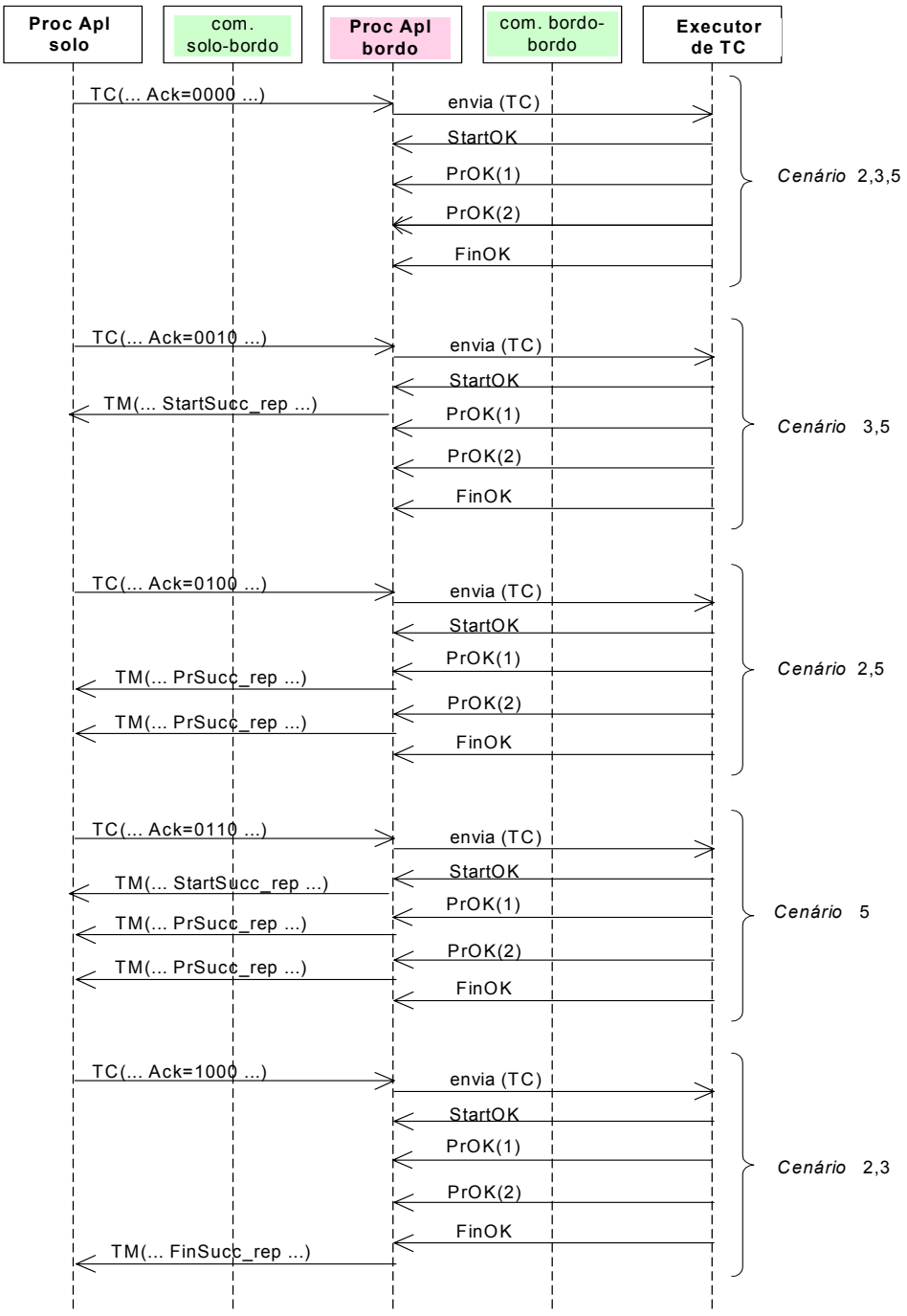


FIGURA C.3 - Diagrama de Seqüência *Normal Propósito 2* ECSS-Verificação TC.

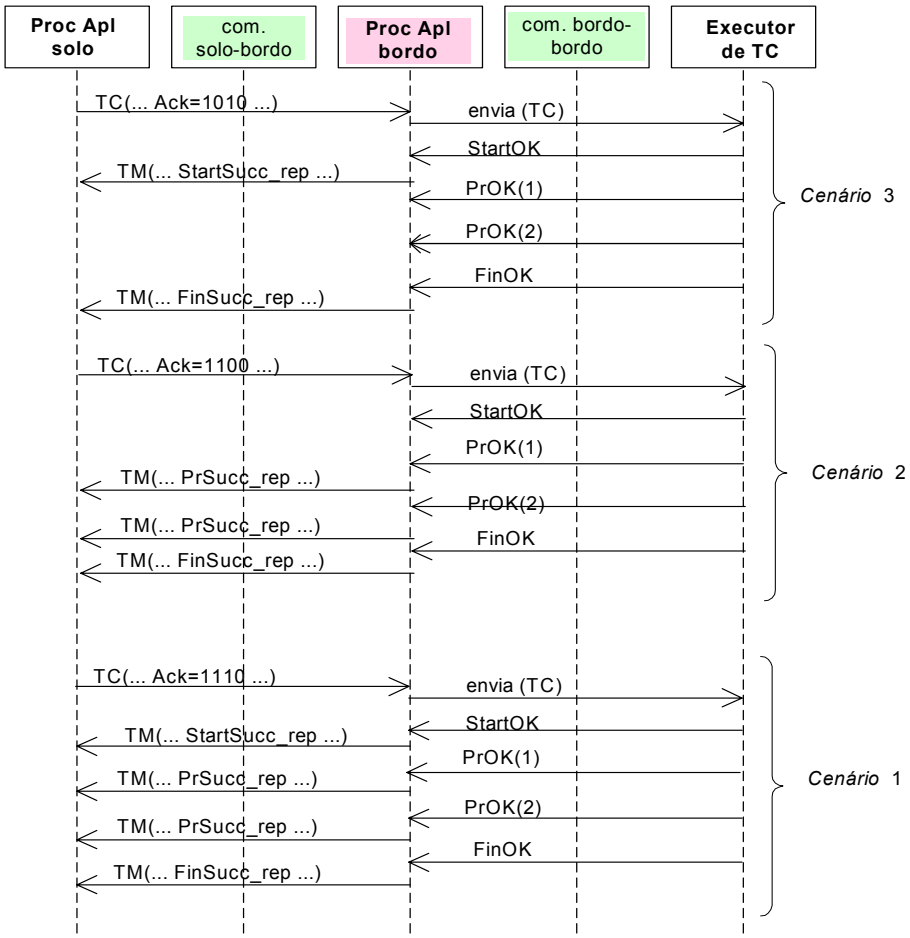


FIGURA C. 4 - Diagrama de Seqüência *Normal Prop 2* ECSS Verificação TC - cont.

Passo 6: Definir Diagramas de Estados Normais.

Os diagramas de estado criados com objetivo de gerar casos de teste exigiram a definição das variáveis conseqüentes mostradas na TABELA C. 8. Os diagramas para os propósitos 1 e 2 são apresentados na FIGURA C. 5 e na FIGURA C. 6, respectivamente.

TABELA C. 8 - Variáveis conseqüentes dos Diagramas de Estados Normais.

Apid_OK	Resultado positivo da análise do valor do Apid, uma vez que o pacote de TC tenha esse valor correto.
Lenght_OK	Resultado positivo da análise do valor do comprimento.
Cksum_OK	Resultado positivo da análise do valor do checksum.
Type_Ok	Resultado positivo da análise do valor do tipo do serviço
Subtype_Ok	Resultado positivo da análise do valor do sub-tipo do serviço
AppData_OK	Resultado positivo da análise do valor do campo de dados para aplicação.
AccOK	Resultado positivo da verificação de todos campos do pacote de TC, uma vez que e o pacote enviado tenha todos os campos corretos.
ViabiliddOK	Resultado positivo da avaliação de viabilidade de execução do TC, uma vez que a configuração da UUT tenha sido compatível com para permitir a execução do TC.

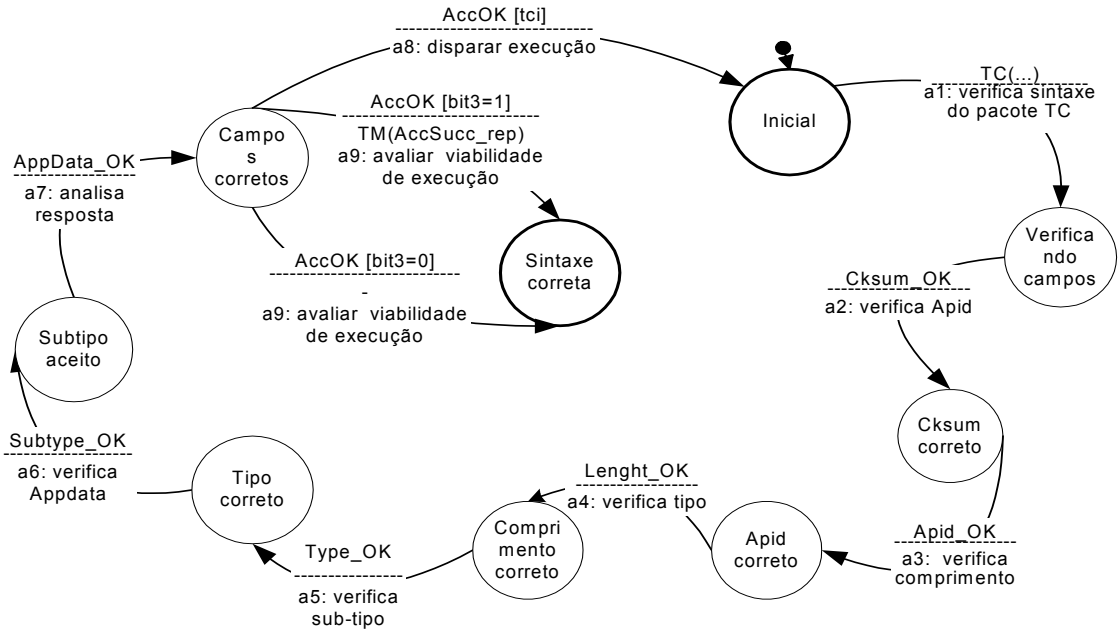


FIGURA C. 5 - Diagrama de Estados *Normal* Propósito 1.

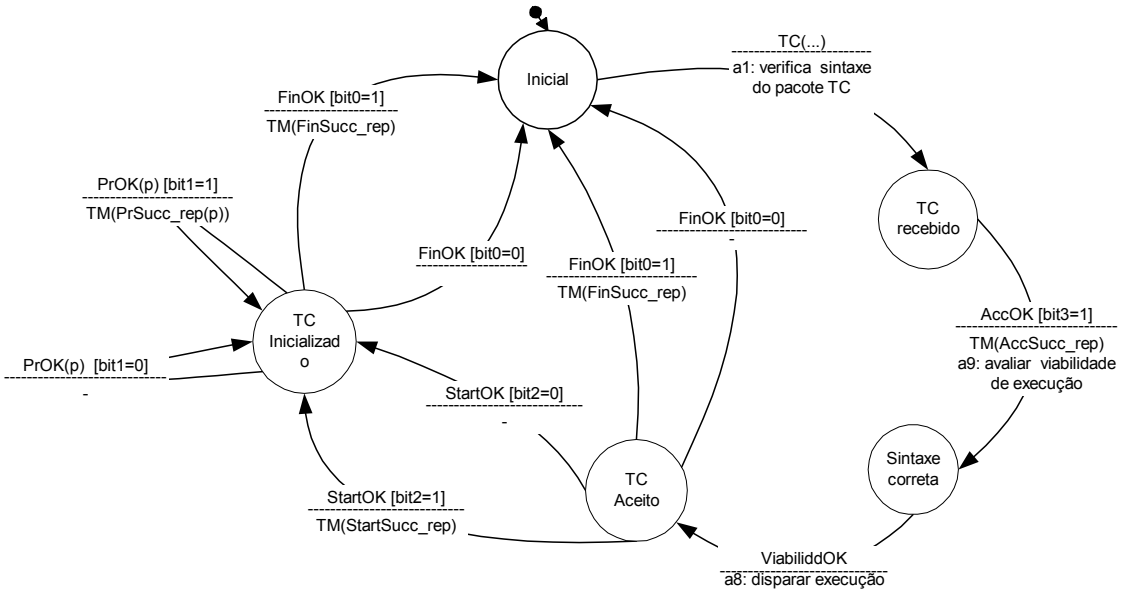


FIGURA C. 6 - Diagrama de Estados *Normal* Propósito 2.

Passo 7: gerar casos de teste.

A ferramenta Condado gerou os casos de teste mostrados na TABELA C. 9 para o Propósito 1. Alguns casos mostrados na TABELA C. 10 cobrem o Propósito 2, considerando-se a passagem em apenas uma vez em cada laço (*loop*).

TABELA C. 9 - Alguns casos de teste para o Propósito 1.

```
senddata(L,Tc)  recdata(L,a1VerificaSintaxe)
senddata(L,CkSumOk)  recdata(L,a2VerificaApid)
senddata(L,ApidOk)  recdata(L,a3VerificaComprimento)
senddata(L,LenghtOk)  recdata(L,a4VerificaTipo)
senddata(L,TypeOk)  recdata(L,a5VerificaSubTipo)
senddata(L,SubTypeOk)  recdata(L,a6VerificaAppdata)
senddata(L,AppDataOk)  recdata(L,a7AnalisaResposta)
senddata(L,AccOKBit3igual0)  recdata(L,a9AvaliarViabilidade)

senddata(L,Tc)  recdata(L,a1VerificaSintaxe)
senddata(L,CkSumOk)  recdata(L,a2VerificaApid)
senddata(L,ApidOk)  recdata(L,a3VerificaComprimento)
senddata(L,LenghtOk)  recdata(L,a4VerificaTipo)
senddata(L,TypeOk)  recdata(L,a5VerificaSubTipo)
senddata(L,SubTypeOk)  recdata(L,a6VerificaAppdata)
senddata(L,AppDataOk)  recdata(L,a7AnalisaResposta)
senddata(L,AccOKBit3igual1)  recdata(L,TMAccSuccRep)  recdata(L,a9AvaliarViabilidade)
...  3
```

TABELA C. 10 - Alguns casos de teste para o Propósito 2.

senddata(L,TC) recdata(L,a1VerificaSintaxe) senddata(L,AccOKBit3igual1) recdata(L,TMAccSuccRep) recdata(L,a9AvaliarViabilidade) senddata(L,ViabilidadOK) recdata(L,a8DisparарExecucao) senddata(L,StartOKBit2igual1) recdata(L,TMStartSuccRep) senddata(L,FinOKBit0igual0) recdata( )
senddata(L,TC) recdata(L,a1VerificaSintaxe) senddata(L,AccOKBit3igual1) recdata(L,TMAccSuccRep) recdata(L,a9AvaliarViabilidade) senddata(L,ViabilidadOK) recdata(L,a8DisparарExecucao) senddata(L,StartOKBit2igual1) recdata(L,TMStartSuccRep) senddata(L,FinOKBit0igual1) recdata(L,TMFinSuccRep)
senddata(L,TC) recdata(L,a1VerificaSintaxe) senddata(L,AccOKBit3igual1) recdata(L,TMAccSuccRep) recdata(L,a9AvaliarViabilidade) senddata(L,ViabilidadOK) recdata(L,a8DisparарExecucao) senddata(L,StartOKBit2igual1) recdata(L,TMStartSuccRep) senddata(L,PrOKpBit1igual0) recdata( ) senddata(L,PrOKpBit1igual1) recdata(L,TMPrSuccRepp) senddata(L,PrOKpBit1igual0) recdata( ) senddata(L,FinOKBit0igual1) recdata(L,TMFinSuccRep)
....
senddata(L,TC) recdata(L,a1VerificaSintaxe) senddata(L,AccOKBit3igual1) recdata(L,TMAccSuccRep) recdata(L,a9AvaliarViabilidade) senddata(L,ViabilidadOK) recdata(L,a8DisparарExecucao) senddata(L,FinOKBit0igual1) recdata(L,TMFinSuccRep)
senddata(L,TC) recdata(L,a1VerificaSintaxe) senddata(L,AccOKBit3igual1) recdata(L,TMAccSuccRep) recdata(L,a9AvaliarViabilidade) senddata(L,ViabilidadOK) recdata(L,a8DisparарExecucao) senddata(L,FinOKBit0igual0) recdata( ) 22

Passo 8: definir Matriz de Transições.

Na matriz de transições do Propósito 2, supomos que Ack.bit3 permanece com o valor = 1 e que a variação dos demais Ack.bits é coberta. A sigla "na" atribuída a alguns campos das tabelas TABELA C. 11 e TABELA C. 12 significa "*not aplicable"*isto é, a ocorrência de evento neste estado não acontece por uma condição mutuamente exclusiva no estado do sistema.

TABELA C. 11 - Matriz de Transições *Propósito 1* ECSS-Verificação TC.

Eventos	**Condições**	**Estados**								
		Inci	Verificando	Cksum cor	APIDco	Compri	Tipo corr	Subtipo	Campos corr	Sintaxe
TC(...)		✔	tbd	tbd	Tbd	tbd	tbd	tbd	tbd	tbd
Cksum_OK		na	✔	na	Na	na	na	na	na	na
Apid_Ok		na	na	✔	Na	na	na	na	na	na
Lengt_OK		na	na	na	✔	na	na	na	na	na
Type_OK		na	na	na	Na	✔	na	na	na	na
Subtype_OK		na	na	na	Na	na	✔	na	na	na
AppData_OK		na	na	na	Na	na	na	✔	na	na
AccOK	bit3=1	na	na	na	Na	na	na	na	✔	na
	bit3=0	na	na	na	Na	na	na	na	✔	na
	tci	na	na	na	Na	na	na	na	✔	na

TABELA C. 12 - Matriz de Transições do *Propósito 2* ECSS-Verificação TC.

Eventos	**Condições**	**Estados**					
		Incial	TCreceb	Sintaxe	TC aceito	TC incializ	Execução
TC(...)		✔	Tbd	tbd	tbd	tbd	tbd
AccOk	bit3=1	na	✔	Na	na	na	na
ViabiliddOK		na	Na	✔	na	na	na
StartOK	bit2=1	tbd	Tbd	tbd	✔	tbd	tbd
	bit2=0	na	Na	Na	✔	na	na
PrOK	bit1=1	tbd	Tbd	tbd	tbd	✔	✔
	bit1=0	na	Na	Na	na	✔	✔
FinOK	bit0=1	tbd	Tbd	tbd	✔	✔	✔
	bit0=0	na	Na	Na	✔	✔	✔

Passo 9: definir Modelo de Falhas.

O meio de comunicação solo bordo conta com as várias camadas do protocolo CCSDS, que garante a entrega em ordem correta dos pacotes e evita duplicação dos mesmos. Entretanto, a corrupção nos campos mensagens não é assegurada, sendo, portanto, uma tarefa do Serviço de Verificação de TC, tolerar falhas de corrupção, validando os campos do pacote de TC antes de entregá-lo ao Executor. Este serviço garante que TCs críticos sejam executados indevidamente ou que sejam ignorados. Portanto, o modelo de falhas externas de comunicação para a comunicação solo-bordo é o de corrupção.

O meio de comunicação bordo-bordo, como dito anteriormente, não é definido na norma por ser dependente da missão. Entretanto, por se tratar de um ambiente espacial, supõe-se a existência dos mecanismos de tolerância a falhas de corrupção, de perda e de atraso de informações, para ilustrar a metodologia de teste.

Os mecanismos de tolerância a falhas comportam-se da seguinte forma: (i) mensagens e parâmetros desconhecidos e são informados ao sistema de solo através do *report* de falha com um código que indica falha (f), como definido na TABELA C. 2 do passo 3, e; (ii) atrasos na chegada das informações de execução, maiores que *to* unidades de tempo, podem ser informadas ao sistema de solo com *reports* de falha.

Passos 10 e 11: definir Diagramas de Seqüência Excepcionais e Diagramas de Estados correspondentes.

Sub-passo 10.1: cenários derivados das exceções explicitamente especificadas:

No Propósito 1, as indicações de exceção são representadas por valores errôneos recebidos nos campos do TC. Os valores incorretos para os quais a norma expressa um tratamento são apresentados na Seção 5.1.2. Na TABELA C. 13 analisa-se a variação de valores das variáveis operacionais (passo 3) para definição dos cenários de exceção nos diagramas de seqüência, cujo resultado é mostrado na FIGURA C. 7. Já para a geração do diagrama de estados, os eventos conseqüentes são mostrados na TABELA C. 14 e o diagrama resultante na FIGURA C. 8.

TABELA C. 13 - Valores errôneos para os campos do pacote de TC.

Nome	*Valores errôneos especificados*
Apid	faixa de valores não permitidos, estabelecida na missão
Length	depende do tipo de pacote de TC.
Type	valor diferente de 1 para o serviço Verificação de TC.
Subtype	valor diferente de [1, 2, 3, 4, 5, 6, 7, 8] se tipo = 1
Checksum	calculado segundo uma função, tomar um valor errado
Ack .bit3 .bit2 .bit1 .bit0	Como todas as combinações de valores binários para os 4 bits são previstas, não se tem teste de exceção para esse campo.

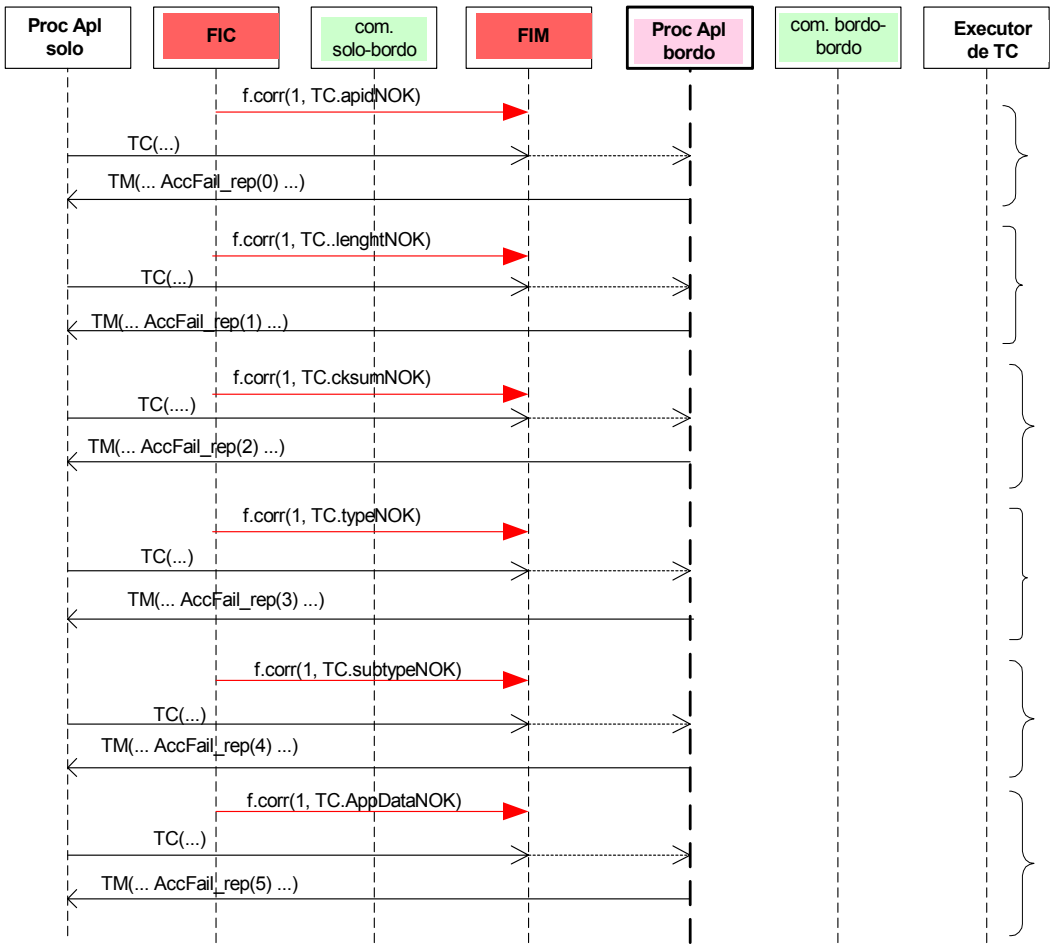


FIGURA C. 7 - Diagrama de Seqüência *Exceções Especificadas* Propósito 1.

TABELA C. 14 - Eventos conseqüentes nos Diagramas de Estado Excepcionais.

Apid_NOK	Resultado negativo da análise do valor do Apid, uma vez que o pacote de TC tenha esse valor incorreto.
Length_NOK	Resultado negativo da análise do valor do comprimento, uma vez que o pacote de TC tenha esse valor incorreto.
Cksum_NOK	Resultado negativo da comparação entre o valor do checksum recebido e o calculado a bordo.
Type_NOk	Resultado negativo da análise do valor do tipo do serviço, uma vez que o pacote de TC tenha o valor diferente de 1 para o serviço de Verificação de TC.
Subtype_NOk	Resultado negativo da análise do valor do sub-tipo do serviço.
AppData_NOK	Resultado negativo da análise do valor do campo de dados para aplicação.
ViabiliddNOK	resultado negativo da avaliação de viabilidade de execução do TC. Resultante da configuração adequada para permitir a execução do TC.

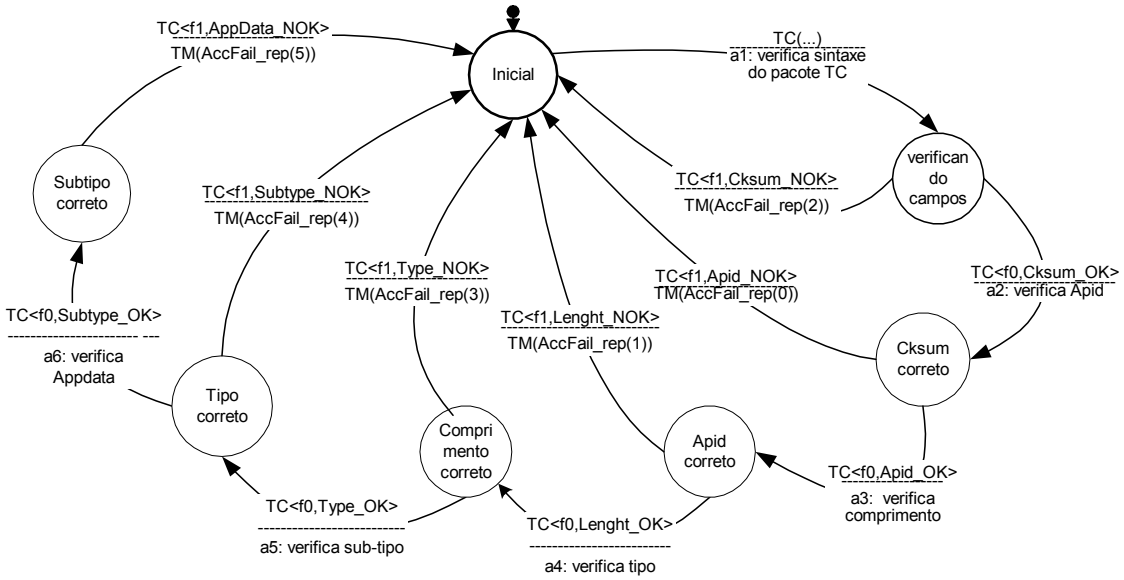


FIGURA C. 8 - Diagrama de Estados *Exceções Especificadas* Propósito 1.

No Propósito 2, as indicações de exceção na execução do TC são fornecidas pelos eventos: **StartNOK(f)**, **PrNOK**(p, f) e **FinNOK**(f) (conforme suposições feitas no passo 3). No passo 10 supomos que os parâmetros dessas mensagens são sempre corretos. O diagrama de seqüência da FIGURA C.9 ilustra os cenários de exceções

especificadas do Propósito 2. A FIGURA C.10 mostra o diagrama de estados equivalente.

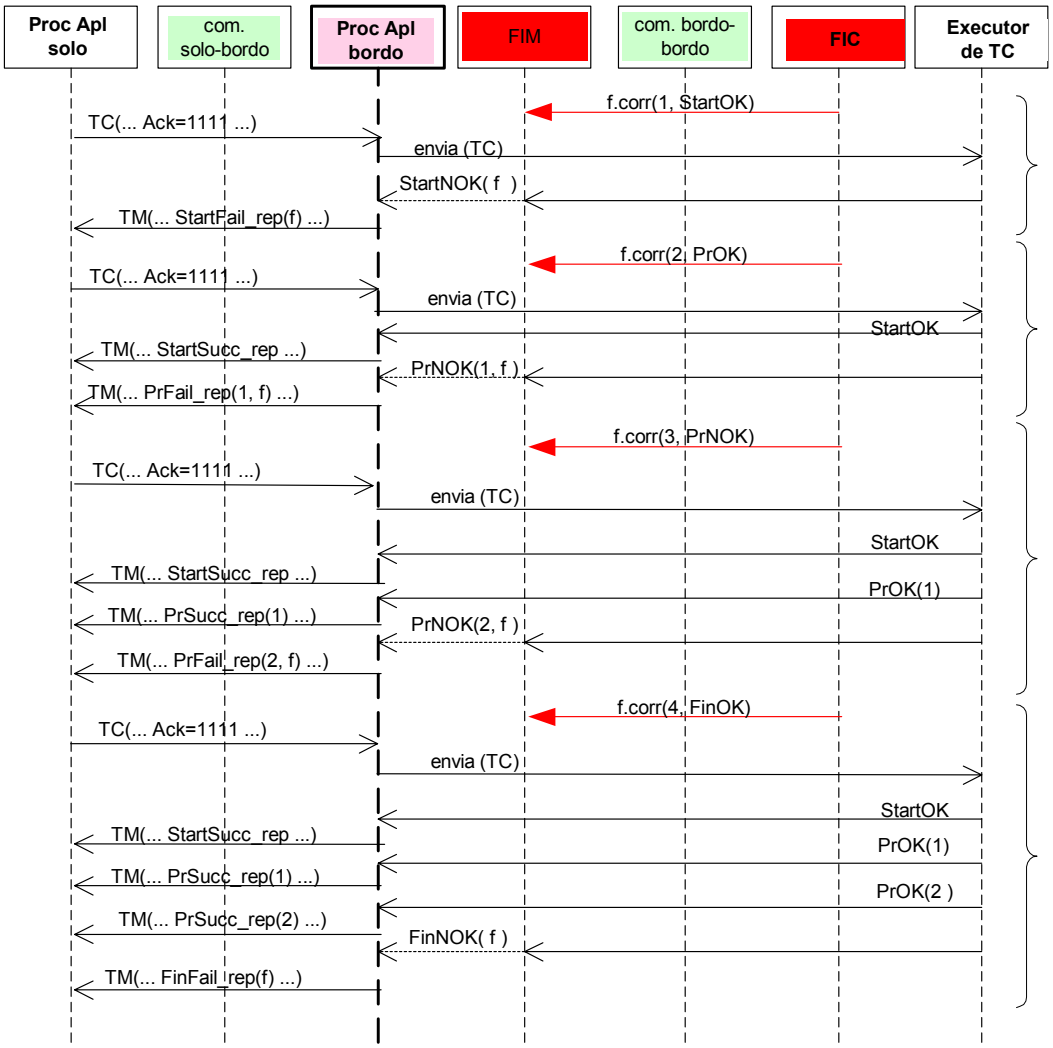


FIGURA C.9 - Diagrama de Seqüência *Exceções Especificadas* Propósito 2.

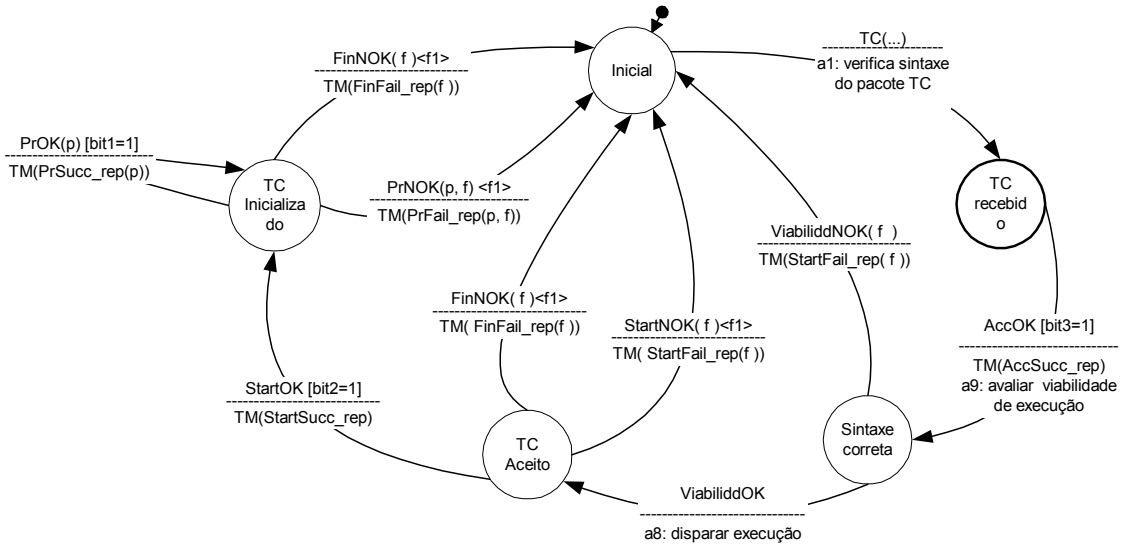


FIGURA C.10 - Diagrama de Estados *Exceções Especificadas* Propósito 2.

Sub-passo 10.2: cenários derivados da análise da completeza da matriz de transições (exceções provocadas por eventos previstos em estados errôneos) são apresentados nos diagramas de estado nas figuras FIGURA C.11 e FIGURA C.12 e, respectivamente para o Propósito 1 para o Propósito 2. Os diagramas de seqüência seguem a mesma regra de formação. Eles não são mostrados aqui para simplificação do texto.

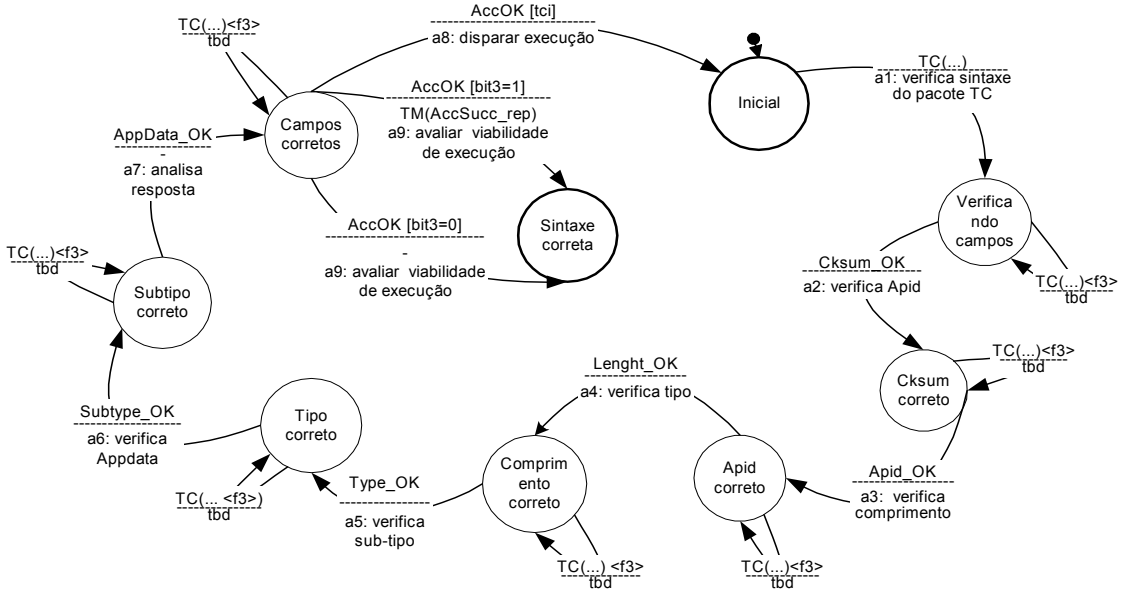


FIGURA C.11 - Diagrama de Estados *Exceções caminhos furtivos* Propósito 1.

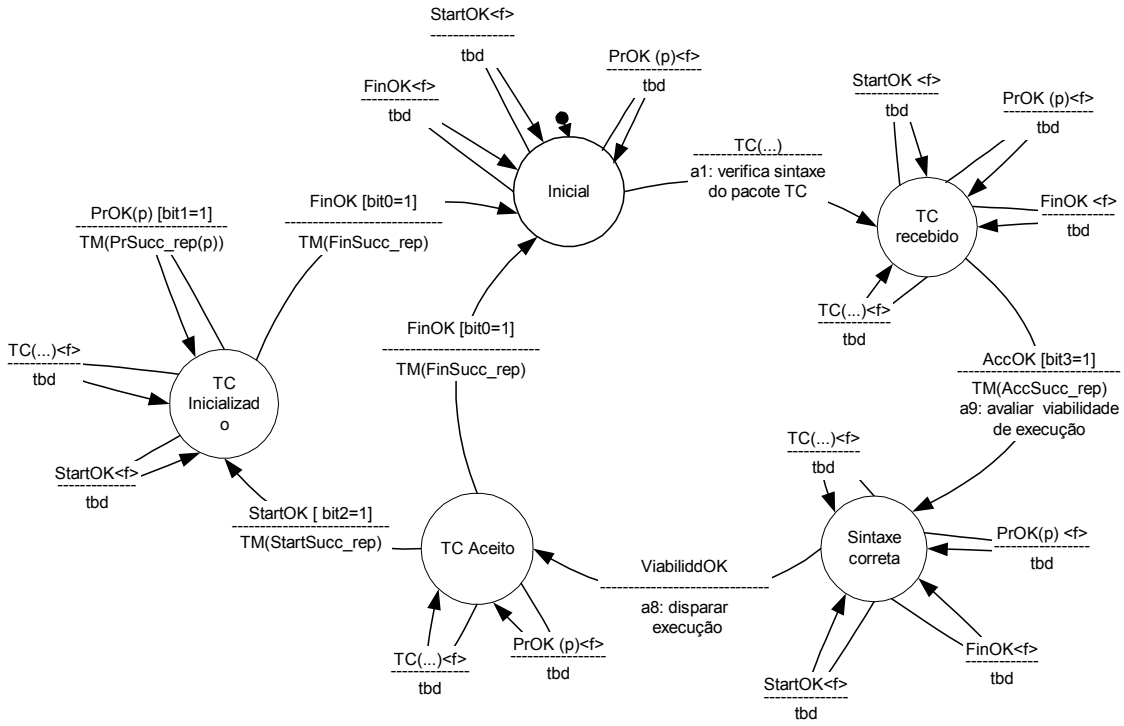


FIGURA C.12 - Diagrama de Estados *Exceções furtivas* Propósito 2.

Sub-passo 10.3: cenários derivados da combinação do modelo de falhas com os cenários normais, visando geração de testes de robustez. Neste caso, dois modelos de falhas foram considerados: o modelo do meio solo-bordo e o modelo do meio bordo-bordo.

O Modelo de falhas do meio solo-bordo influencia os cenários do Propósito 1, mas não os do Propósito 2. Aplicando-se falhas de corrupção aos cenários normais do propósito 1, observa-se a criação dos mesmos cenários definidos no passo 10.1, de exceções especificadas, (ver FIGURA C. 7).

O Modelo de falhas no meio bordo-bordo aplicado aos cenários normais do propósito 2, induz a geração dos cenários excepcionais de tolerância a falhas externas. A FIGURA C. 13 mostra os cenários resultantes do tratamento de falhas de corrupção de mensagens, a FIGURA C. 14 do tratamento de perda de mensagens e a FIGURA C. 15 do atraso de mensagens.

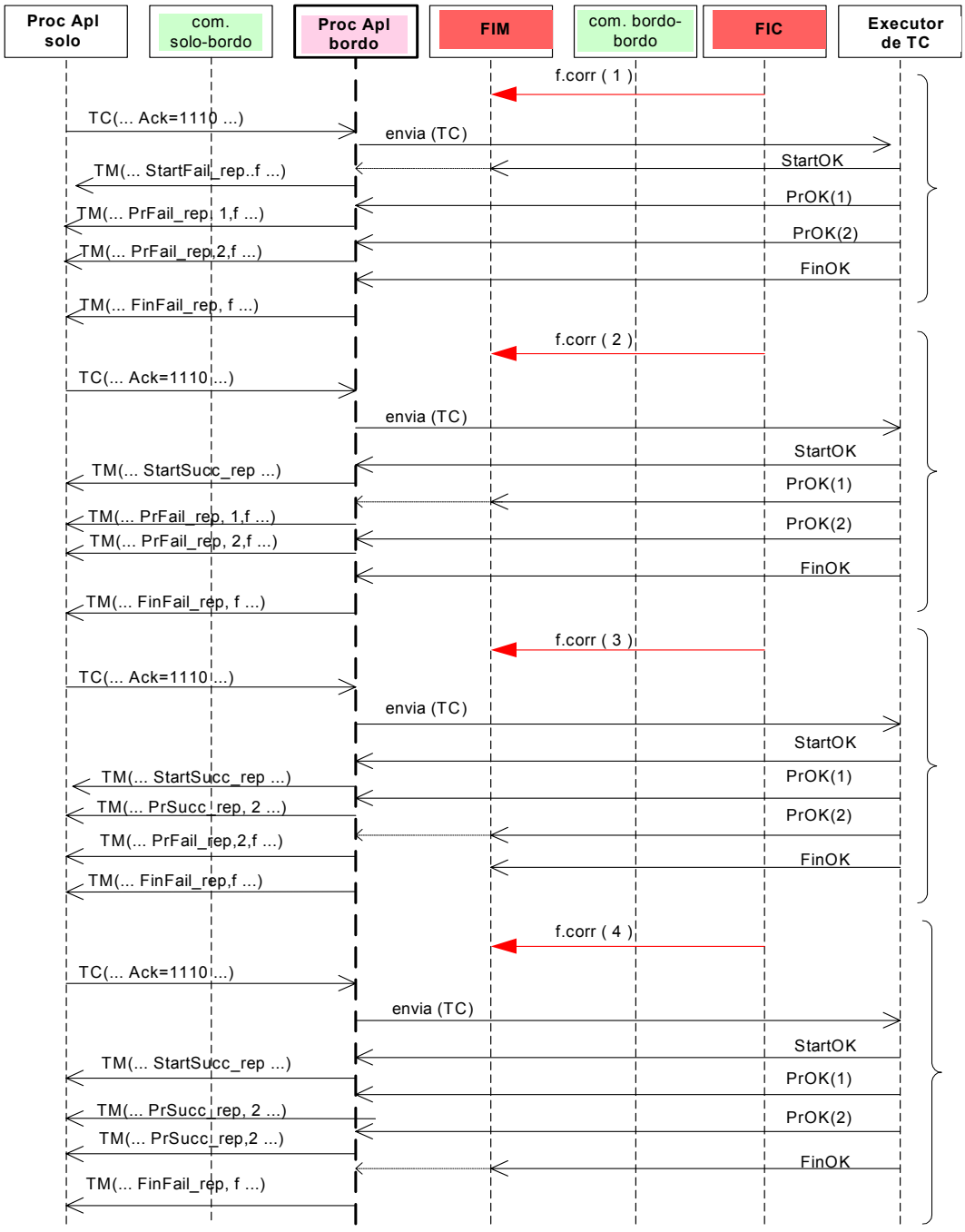


FIGURA C. 13 - Diagrama de Seqüência *Exceções TF corrupção* Propósito 2.

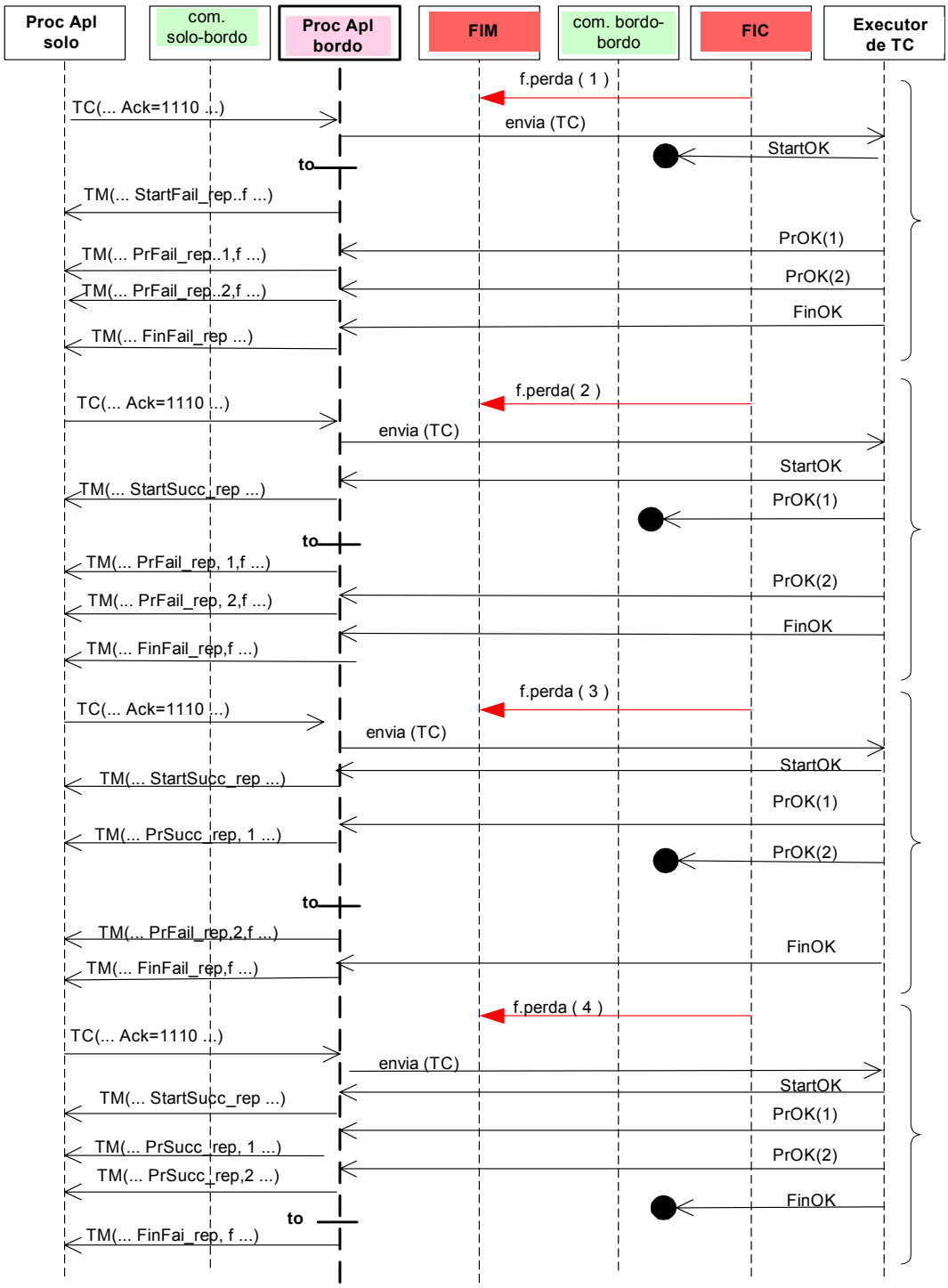


FIGURA C. 14 - Diagrama de Seqüência *Exceções TF perda* Propósito 2.

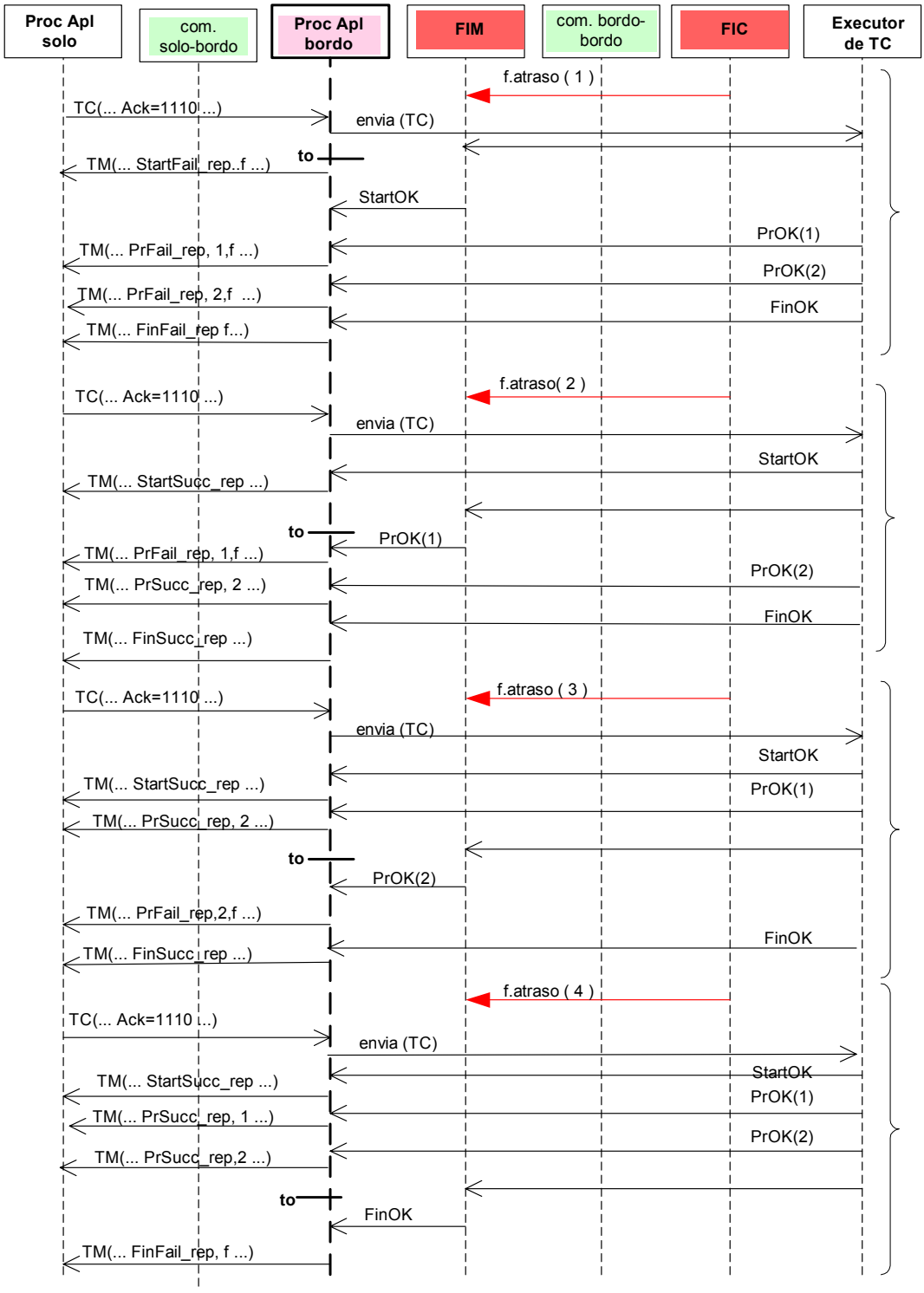


FIGURA C. 15 - Diagrama de Seqüência *Exceções TF atraso* Propósito 2.

As informações de tratamento de *tempo esgotado* (“timeout”) e de comandos desconhecidos recebidos do Executor de TC não estão definidas na norma. Entretanto, falhas no meio de comunicação podem levar a atraso, perda e corrupção a ponto do comando não ser identificado pelo serviço. Para elaboração do diagrama de estado, dois eventos foram acrescentados, com objetivo de completar os testes de robustez:

- *to* - indica que o intervalo de tempo esperado pela resposta do Executor de TC esgotou-se,

- *CmdDesc* – indica que um comando desconhecido foi recebido do Executor de TC. Um tratamento para este evento deverá ser definido posteriormente. Estas ações irão substituir a expressão *tbd*.

O diagrama de estados que retrata o comportamento do serviço frente a falhas de atraso e perda de mensagens, encontra-se na FIGURA C. 16. O comportamento quando em presença da falhas de corrupção é ilustrado na FIGURA C. 17.

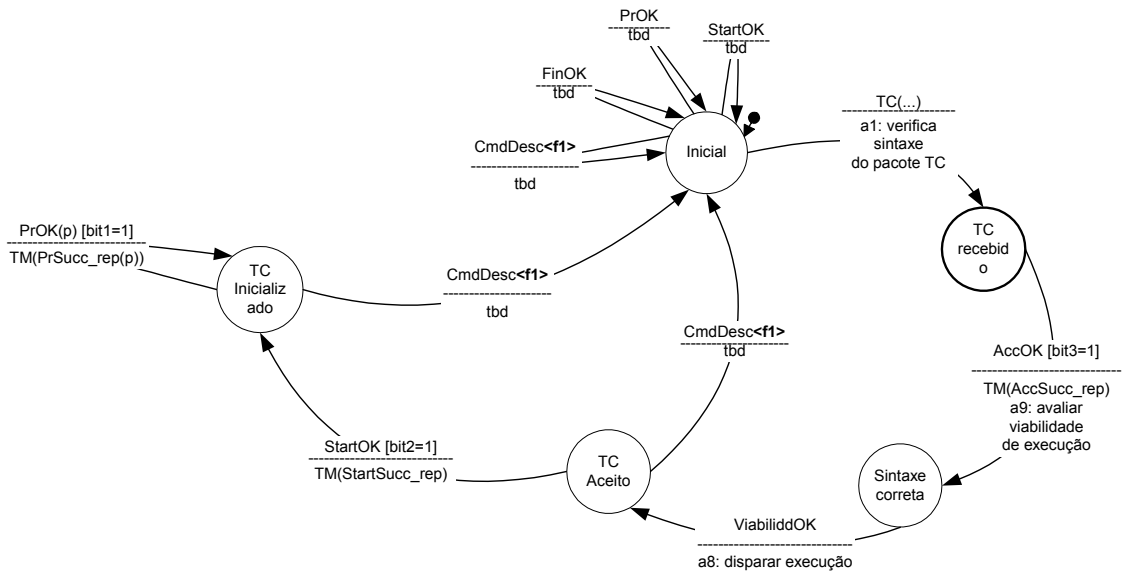


FIGURA C. 16 - Diagrama de Estado *Exceções TF Corrupção* Propósito 2.

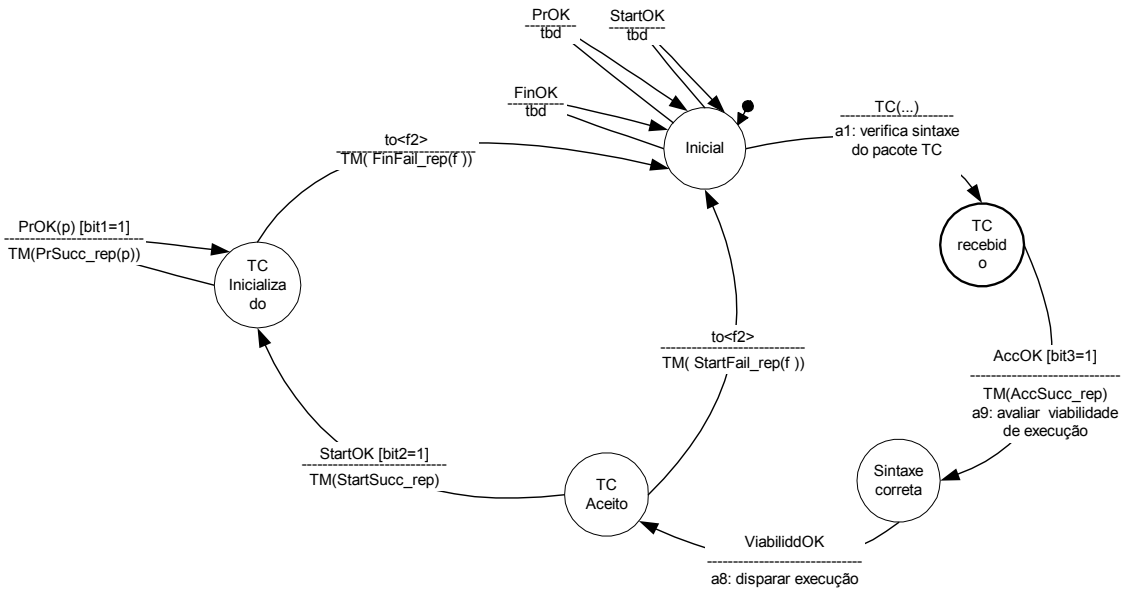


FIGURA C. 17 - Diagrama de Estados *Exceções TF Atraso e Perda* Propósito 2.

Passo 11: gerar Casos de Falha.

Todos os casos de falha foram gerados com a ferramenta Condado, a partir das máquinas de estados definidas no passo 10. A saída da Condado segue o mesmo padrão apresentado no passo 7.

C.2 – Diagramas com Propósitos de Teste em um Único Modelo

Os diagramas de estados dos modelos parciais que refletem os propósitos 1 e 2 unificados para o tratamento do comportamento normal, do comportamento de exceções especificadas e do comportamento de exceções por caminhos furtivos, do exemplo ECSS-*TC Verification,* são ilustrados nas figuras FIGURA C.18, FIGURA C.19 e FIGURA C.20, respectivamente.

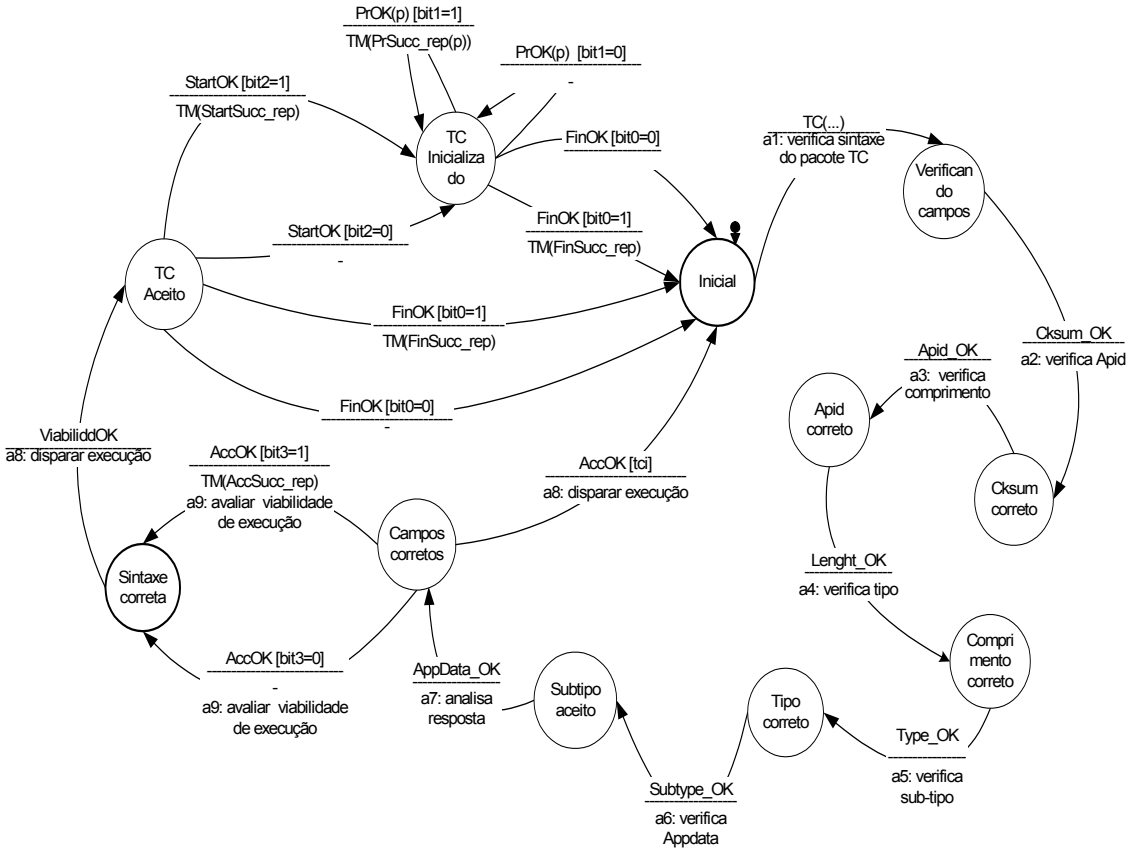


FIGURA C.18 - Diagrama de Estados *Normal* Propósitos 1 e 2 juntos.

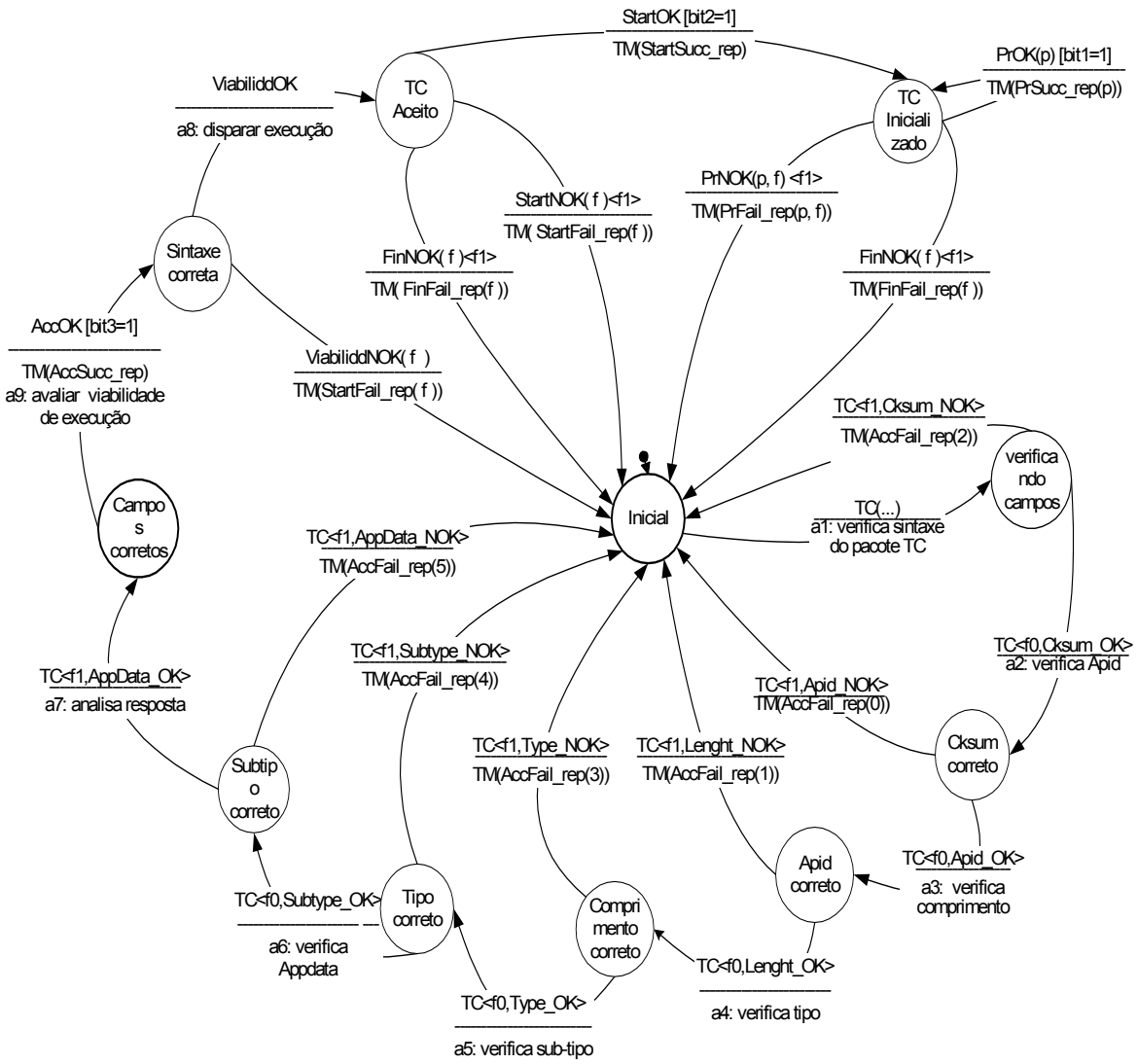


FIGURA C.19 - Diagrama de Estados *Exceções Especificadas* Propósitos 1 e 2.

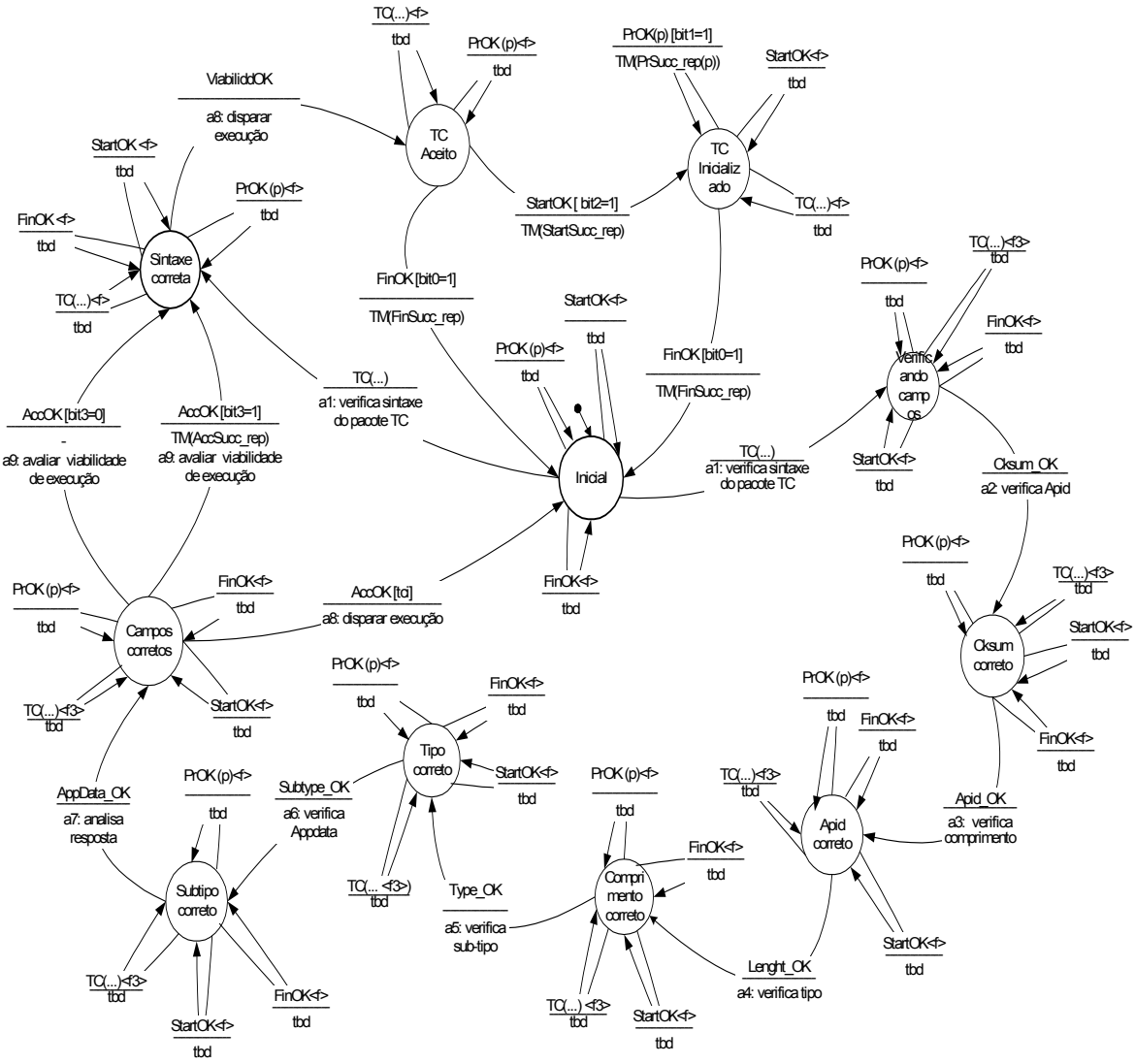


FIGURA C.20 - Diagrama de Estados *Exceções Furtivas* Propósitos 1 e 2.

C.3. – Diagrama de Estado do Modelo Total

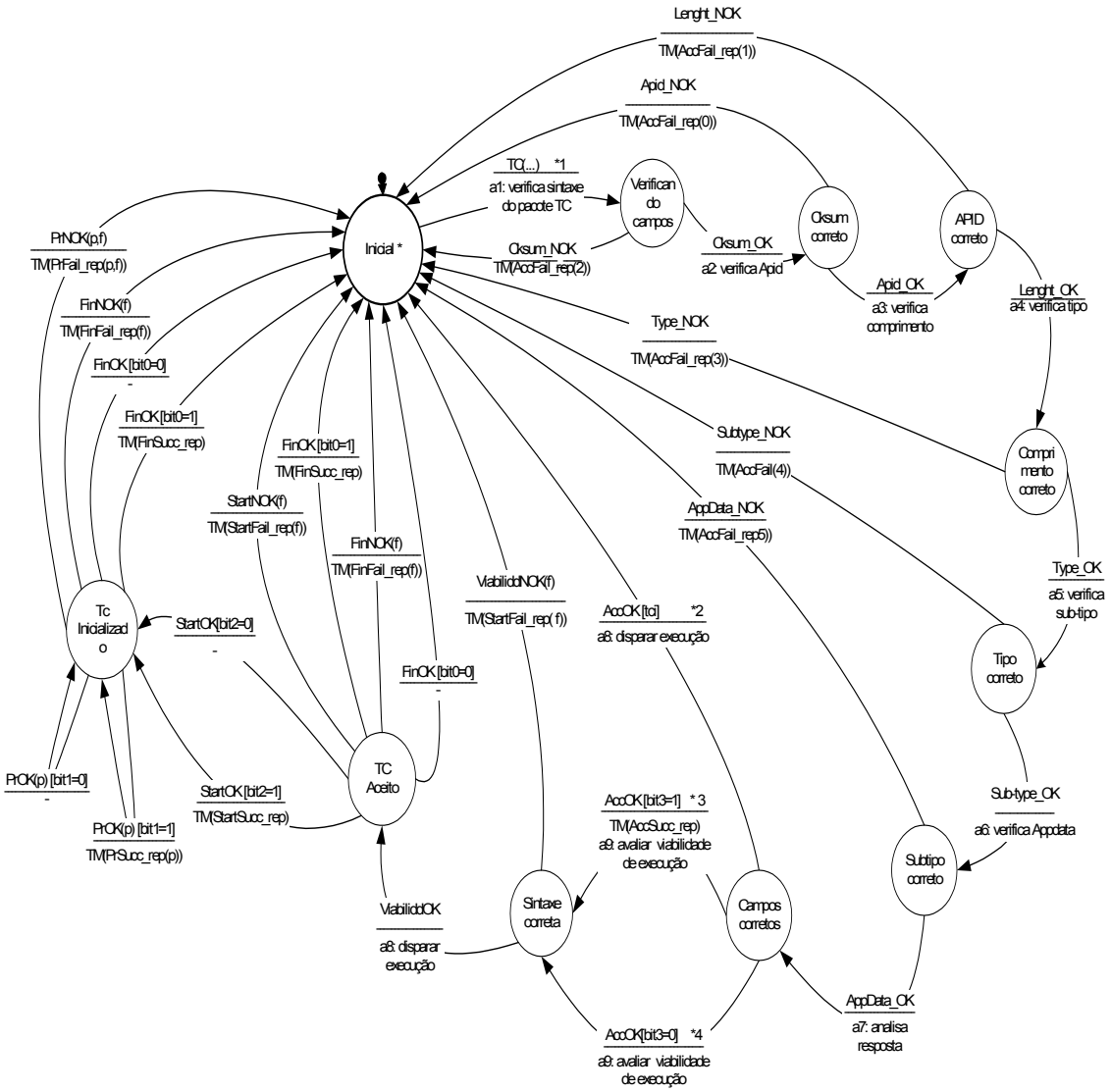


FIGURA C. 21 - Modelo Total do serviço ECSS- Verificação TC.

ANEXO A

ESPECIFICAÇÃO DO PROTOCOLO SOLO-SOLO

The gray cells show the error cases and are nominally not expected. Each case where a gray cell is reached a detailed message giving the cause of the anomaly will be provided in the CLIENT logbook.

The real time protocol philosophy is: error on CC port leads to the end of real time exchanges; error on Data port closes the Data port. Definitions used in the table:

Application message: message like OPEN_SESSION, CLOSE_SESSION, ACK_OPEN_SESSION, etc, …

CC port: Command/Control port where application message are exchanged

Data port n: correct number n data port where data are exchanged (n is the number given in an application message)

Error on CC port: "CC port TCP-IP error" or "CC port reset" or "CC port close" received

Error on Data port n: "Data port n TCP-IP error" or "Data port n reset" or "Data port n close" received

E.P. : "End of Process" = client tasks to close properly the connection with the Server i.e. : close all the client "c" data ports

close the client "c" CC portB.T.: "Background Task" for data port = send and/or receive data in tcp-ip sockets

***: see specific comments attached to the Cell at the end of the table. These comments give processing details

n/a: transition cannot happen

NB: Parameter allowing to connect or send messages several times ("NB" =1 minimum and 3 maximum)

Initial State : The client is in **State 0**. All the Data ports and the CC port are closed for the client. The client has always the initiative.

CLIENT Received events - - - - >	*Nothing*	*Manual request to connect to the server*	*Connection on CC port accepted*	*Connection on CC port refused*	*Response to a Open_session*	*Response to a Open_session*	*Connection on Data port **n** accepted*	*Connection on Data port **n** refused*	*Error on Data port*	*Error on CC port*	*Manual request to send Close_session*	*Response to a Close_session*	*Response to a Close_session*
States ↓					*ACK*	*NACK or anything*						*ACK*	*NACK or anything*
State 0 : *Wait for request to be connec-ted to server*	**0.1 go to State 0**	**0.2 send connection on CC port** then **go to State 1**	**0.3 n/a**	**0.4 n/a**	**0.5 n/a**	**0.6 n/a**	**0.7 n/a**	**0.8 n/a**	**0.9 n/a**	**0.10 n/a**	**0.11 n/a**	**0.12 n/a**	**0.13 n/a**
State 1 : *Wait **5s** *** for CC connect. accept*	**1.1 - *** -go to State 0**	**1.2 ignored** then **go to State 1**	**1.3 send Open_session (for port n)** then **go to State 2**	**1.4 E.P.** then **go to State 0**	**1.5 n/a**	**1.6 n/a**	**1.7 n/a**	**1.8 n/a**	**1.9 n/a**	**1.10 E.P.** then **go to State 0**	**1.11 ignored** then **go to State 1**	**1.12 n/a**	**1.13 n/a**
State 2 : *Wait **5s** for an Open_session response*	**2.1 go to State 2**	**2.2 ignored** then **go to State 2**	**2.3 n/a**	**2.4 n/a**	**2.5 - *** - send connection on data port n** then **go to State 3**	**2.6 go to State 2**	**2.7 n/a**	**2.8 n/a**	**2.9 close port then go to State 2**	**2.10 E.P.** then **go to State 0**	**2.11 E.P.** then **go to State 0**	**2.12 ignored** then **go to State 2**	**2.13 ignored** then **go to State 2**

CLIENT Received events *- - - - >*	*Nothing*	*Manual request to connect to the server*	*Connection on CC port accepted*	*Connection on CC port refused*	*Response to a Open_session*	*Response to a Open_session*	*Connection on Data port **n** accepted*	*Connection on Data port **n** refused*	*Error on Data port*	*Error on CC port*	*Manual request to send Close_session*	*Response to a Close_session*	*Response to a Close_session*
States *↓*					*ACK*	*NACK or anything*						*ACK*	*NACK or anything*
State 3 : *Wait **5s** for data port connect. accept*	**3.1 go to State 2**	**3.2 ignored** then **go to State 3**	**3.3 n/a**	**3.4 n/a**	**3.5 E.P.** then **go to State 0**	**3.6 E.P.** then **go to State 0**	**3.7 Run B.T.** then **go to State 2 or 4**	**3.8 go to State 2**	**3.9 n/a**	**3.10 E.P.** then **go to State 0**	**3.11 E.P.** then **go to State 0**	**3.12 ignored** then **go to State 3**	**3.13 ignored** then **go to State 3**
State 4 : *Wait for request to send "Closesession" message*	**4.1 go to State 4**	**4.2 ignored** then **go to State 4**	**4.3 n/a**	**4.4 n/a**	**4.5 E.P.** then **go to State 0**	**4.6 E.P.** then **go to State 0**	**4.7 n/a**	**4.8 n/a**	**4.9 close port then go to State 2**	**4.10 E.P.** then **go to State 0**	**4.11 send Close_session** then **go to State 5**	**4.12 ignored** then **go to State 4**	**4.13 ignored** then **go to State 4**
State 5 : *Wait **5s** for a Close_session response*	**5.1 E.P.** then **go to State 0**	**5.2 ignored** then **go to State 5**	**5.3 n/a**	**5.4 n/a**	**5.5 E.P.** then **go to State 0**	**5.6 E.P.** then **go to State 0**	**5.7 n/a**	**5.8 n/a**	**5.9 ignored** then **go to State 5**	**5.10 E.P.** then **go to State 0**	**5.11 ignored** then **go to State 5**	**5.12 E.P.** then **go to State 0**	**5.13 E.P.** then **go to State 0**