MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-9594-PRE/5222

USO DA FERRAMENTA DE TESTES FSOFIST NA VALIDAÇÃO DE UMA APLICAÇÃO ESPACIAL

Eliane Martins*
Maria de Fátima Mattiello-Francisco
Anderson Nunes Paiva Morais*

*Instituto de Computação - UNICAMP

Trabalho publicado nos Anais da 2ª Jornada Ibero-Americana de Engenharia de Software e Engenharia de Conhecimento – Salvador, 30 out. a 01 de nov., 2002.

INPE
São José dos Campos
2003

Uso da ferramenta de testes FSOFIST na validação de uma aplicação espacial

Eliane Martins[1] Maria de Fatima Mattiello Francisco[2] Anderson Nunes Paiva Morais[1]

[1] Instituto de Computação - Unicamp
CP 6176 Campinas 13083-970 SP
Tel.: 19.3788.58.72 Fax: 19. 3788.58.47
{eliane, Anderson.Morais} @ic.unicamp.br

[2] Instituto Nacional de Pesquisas Espaciais (INPE)
CP 1515 SJCampos 12227-010 SP
fatima@dss.inpe.br

RESUMO

O artigo apresenta validação de uma pequena aplicação da área espacial utilizando uma ferramenta de apoio aos testes de protocolos de comunicação, denominada FSOFIST. Esta ferramenta é baseada na arquitetura *ferry*, proposta para apoiar os testes de conformidade de protocolos de comunicação. A FSOFIST estende essa arquitetura para permitir também os testes por injeção de falhas. Neste texto apresentamos uma descrição da ferramenta e os resultados obtidos nos testes de conformidade de uma aplicação real. Os testes também tiveram como objetivo avaliar o uso na prática de uma ferramenta desenvolvida em meio acadêmico, como parte de um primeiro passo de um processo de transferência de tecnologia.

PALAVRAS-CHAVE:

Testes de conformidade de protocolos - injeção de falhas – arquitetura de testes – o conceito de ferry-clip

1. INTRODUÇÃO

Nas últimas décadas observou-se o crescimento contínuo de sistemas distribuídos nos mais diversos ambientes: indústrias, escritórios, instituições de ensino e pesquisa, bancos, organizações comerciais, hospitais. Na área espacial, sistemas distribuídos são muito utilizados para comunicação entre segmentos solo e espacial.

Em sistemas distribuídos a funcionalidade (interface com usuário, armazenamento de dados) está distribuída em diferentes computadores. Esses diferentes computadores estão interconectados através de redes. Para que esses computadores possam se comunicar com sucesso, é necessário um conjunto bem definido de regras de comunicação. Um *protocolo* descreve um conjunto de regras que um sistema computacional deve seguir para que possa se comunicar com outros sistemas. Uma *entidade de protocolo* é a parte do sistema computacional responsável por garantir localmente a comunicação através do protocolo utilizado

A dependência crescente neste tipo de sistema faz com que requisitos como confiabilidade, disponibilidade, segurança e desempenho se tornem cruciais. O uso de métodos formais para a especificação e verificação de protocolos representam uma grande contribuição para a área, pois permitem que a validação comece cedo no ciclo de vida de uma implementação de um protocolo. Por validação aqui entenda-se tanto a verificação, com o intuito de determinar se o sistema apresenta as propriedades desejadas, quanto a avaliação de medidas tais como a confiabilidade, disponibilidade e desempenho [13].

O uso de métodos formais para especificação e verificação é necessário mas não suficiente para garantir que implementações de entidades de protocolos desenvolvidas em ambientes heterogêneos sejam capazes de se comunicar com sucesso. Testes devem ser realizados para complementar essas atividades. Organismos internacionais de padronização, como a ISO e a CCITT, recomendam que sejam feitos vários tipos de testes [11]: de conformidade, de interoperabilidade, de desempenho e de robustez. Os *testes de conformidade* visam determinar se uma implementação de uma entidade é conforme à especificação do protocolo. Os *testes de interoperabilidade* visam determinar se duas implementações de entidades conformes à mesma especificação são capazes de interoperar. Nos *testes de desempenho* é avaliado se a implementação do protocolo apresenta o desempenho esperado. E finalmente os *testes de robustez* visam determinar o quanto uma implementação é capaz de se recuperar de situações de erro não previstas.

Testes de robustez, no entanto, apresentam um modelo de falhas limitado [4]. Por essa razão, propomos os testes por injeção de falhas, pois este método permite validar um sistema em presença de uma grande variedade de situações de erro, previstas ou não na especificação. Antes de prosseguirmos, cumpre notar que o termo *falha* é empregado aqui como a tradução de *fault*, conforme a terminologia definida em [14].

Este texto apresenta uma ferramenta construída com o objetivo de dar apoio a diversos tipos de testes de protocolos, em especial os testes de conformidade e os testes por injeção de falhas. O projeto da ferramenta utiliza a arquitetura *ferry* [24, 25], proposta para os testes de protocolos. A escolha dessa arquitetura se deveu a uma série de características:

- permite implementar as arquiteturas de testes padronizadas;

- apoia a execução de testes tanto de conformidade quanto de interoperabilidade;
- oferece alto grau de portabilidade, facilitando sua utilização em diversas plataformas;
- possui alto grau de modularidade;
- apresenta um grau reduzido de intrusão no Sistema sob Teste.

A ferramenta estende esta arquitetura de modo a incorporar os testes por injeção de falhas, daí o seu nome: FSOFIST (ferry-clip with Sofware Fault Injection Support Tool) [3].

Este documento está estruturado da seguinte forma: a próxima seção apresenta alguns aspectos relativos aos testes de protocolos, em especial os testes de conformidade e a arquitetura *ferry*, e os testes por injeção de falhas. A seção 3 descreve a FSOFIST, apresentando aspectos de projeto e implementação da mesma. A seção 4 apresenta o uso da ferramenta na validação de uma aplicação espacial e os principais resultados obtidos. A seção 5 conclui o texto, mostrando direções para trabalhos futuros.

2. TESTES DE PROTOCOLOS

2.1 Testes de conformidade

Testes de conformidade têm por objetivo determinar se uma dada implementação de uma entidade de protocolo satisfaz a sua especificação. No contexto do Modelo de Interconexão para Sistemas Abertos (ou OSI, de *Open Systems Interconnection*), foi desenvolvido um padrão para os testes de conformidade de protocolos do modelo OSI. Trata-se do padrão IS 9646: "OSI *Conformance Testing Methodology and Framework*" [11]. O padrão define a metodologia, a estruturação e especificação de seqüências de testes, além de procedimentos a serem seguidos durante os testes. O intuito é permitir que os resultados de testes realizados por diferentes grupos sejam comparáveis e reprodutíveis, evitando assim a duplicação de esforços. O padrão não estabelece como os testes devem ser gerados, mas define um arcabouço para a estruturação dos testes, bem como a forma de especificá-los. O padrão define também as arquiteturas de apoio à execução dos testes.

Segundo o padrão as arquiteturas (também chamadas de métodos) de testes devem basear-se numa metodologia abstrata de testes, cujo principal objetivo é a especificação dos pontos de controle e observação das entradas e saídas da implementação em testes. Sendo assim, a arquitetura conceitual de testes pode ser descrita conforme a figura 2.1 [19].

Dado que existem situações nas quais a implementação sob testes (IST) e os testadores podem residir no mesmo ambiente físico e outras em que a IST reside em um ambiente remoto, foram propostas diversas arquiteturas: local, distribuída e remota.

Na arquitetura local a IST interage tanto com o Testador Superior (TS) quanto com o Testador Inferior (TI), interceptando as Primitivas de Serviço Abstrato (PSA) nas interfaces superior e inferior da IST, estas constituindo, portanto, os Pontos de Controle e Observação (PCO) usados para os testes.

As demais arquiteturas não são locais, e se caracterizam pelo fato de que o TI só tem acesso remotamente à interface inferior da IST. O único PCO é portanto a interface superior. O TI reside no Sistema de Testes, remoto, e a interação com a IST se faz através dos serviços da camada N-1. Na arquitetura distribuída o TS está localizado fisicamente junto à IST, tendo acesso direto a sua interface superior. O Procedimento de Coordenação dos Testes (PCT) deve portanto implementar um protocolo para a comunicação entre o TS e o TI. Na arquitetura remota o TS localiza-se no Sistema de Testes, e não tem acesso direto à interface superior da IST. Suas funções podem ser exercidas à distância, ou embutidas na IST ou ainda podem ser exercidas por um operador.

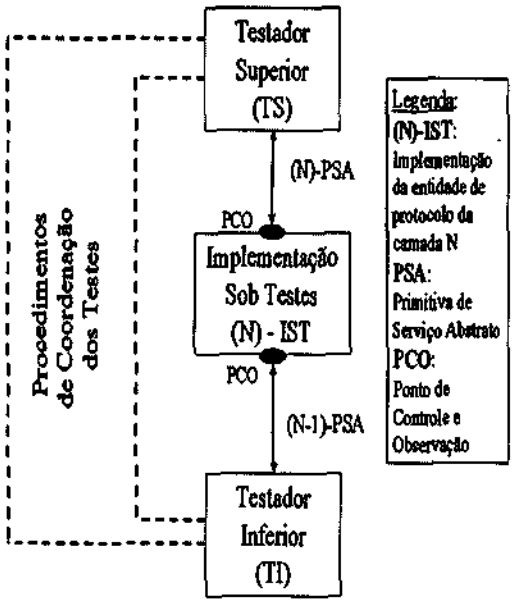


Figura 2.1. Arquitetura conceitual para os testes de conformidade.

2.2 Testes por injeção de falhas

A injeção de falhas consiste na introdução deliberada de falhas ou erros em um sistema com o intuito de observar o seu comportamento. Essa característica faz com que esta seja uma técnica muito útil para a validação de mecanismos de recuperação de erros, de mecanismos de tratamento de exceções ou ainda para determinar se um sistema (ou componente do mesmo) apresenta comportamento de risco em presença de falhas.

Existem diversas abordagens para injeção de falhas [2, 7, 10] entre elas a injeção por software. Nesta abordagem são feitas alterações de forma estática, mudando-se o código fonte, ou dinâmica, alterando-se o estado do sistema alvo. A injeção dinâmica necessita de um código extra, denominado injetor de falhas.

No caso específico da FSOFIST, o código do injetor é inserido em uma camada intermediária entre a camada alvo e a camada inferior para emular conseqüências de falhas de comunicação [8, 9, 16, 17, 18]. Tomando como base a figura 2.1, o injetor de falhas seria inserido entre a (N)-IST e a camada inferior, (N-1)-PSA. O injetor de falhas é responsável pela alteração, perda, duplicação ou retardo de mensagens transferidas pelo canal de comunicação.

3. A FERRAMENTA FSOFIST

O objetivo foi construir uma ferramenta que apoiasse tanto os testes de conformidade quanto os testes por injeção de falhas de protocolos de comunicação. Outros requisitos da ferramenta foram: a facilidade de portar para diferentes plataformas, facilidade de adaptar a diferentes implementações de diferentes camadas de protocolos, e facilidade de acréscimo de novas funcionalidades. Por essa razão foi escolhida a arquitetura *ferry*, pois ela apresenta essas características, o que nos permitiu acrescentar os módulos que implementam as funcionalidades para injeção de falhas.

3.1 Arquitetura "ferry-clip"

A arquitetura *ferry* foi proposta em [24, 25] com a finalidade de dar suporte à validação de protocolos e aos métodos de teste definidos no padrão ISO9646.

Esta arquitetura possui uma estrutura altamente modular na qual ambos os testadores residem junto ao Sistema de Teste, o que simplifica a sincronização entre o Testador Superior e o Testador Inferior. Desta forma minimiza-se a quantidade de software residente no Sistema sob Teste.

Os principais componentes da arquitetura ferry são dois *ferry-clips*, denominados Ferry Ativo, e o Ferry Passivo, o primeiro residindo no Sistema de Testes e o segundo, no Sistema Sob Testes, junto à IST [5, 6, 24,]. O AF é responsável por iniciar a conexão com o PF e por transferir os dados dos testadores para o Sistema Sob Testes. O PF é quem efetivamente transfere os dados enviados pelos testadores para a IST. A comunicação entre o AF e o PF se dá através de um canal independente, denominado de canal do *ferry*.

O estudo apresentado em [6] mostrou a flexibilidade da arquitetura para a realização de testes de conformidade e de interoperabilidade. A arquitetura descrita pelos autores é estendida, neste trabalho, para também dar apoio aos testes por injeção de falhas, conforme será mostrado no item a seguir.

3.2 A arquitetura utilizada

A figura do 3.1 apresenta a arquitetura utilizada. Os componentes são descritos sucintamente a seguir. Para maiores detalhes, a referência [3] pode ser consultada.

O **Controlador de Testes** (CT) é responsável por aplicar a seqüência de testes à IST, realizando as funções dos testadores superior e inferior, bem como iniciar e terminar uma sessão de testes. Os dados do CT para a IST são transferidos utilizando o canal do *ferry*. No caso de testes locais, os dados são transferidos através dos ferry-clips, pois o *Ferry* Passivo tem acesso tanto à interface superior quanto à inferior da IST. No caso de testes distribuídos, o *Ferry* Passivo só tem acesso à interface superior; os dados para a interface inferior são transmitidos através do canal de comunicação em testes.

Os *Ferry* Ativo e Passivo possuem estruturas similares, sendo que o segundo é uma versão simplificada do primeiro para reduzir o impacto no Sistema sob Testes. Além da transferência de dados, estes componentes são responsáveis pela conversão dos dados de teste para um formato aceito pela IST. Dessa forma, o CT não precisa ser modificado a cada nova IST, bastando para isso alterar os módulos dos *ferries* que realizam a conversão.

O código do injetor de falhas é dividido em duas partes: o **Controlador de Injeção de Falhas** (CIF) e o **Módulo Injetor de Falhas** (MIF). O primeiro reside no sistema de testes e implementa as funções de injeção que são independentes da IST. Suas funções incluem determinar o momento em que uma falha deve ser injetada, e por quanto tempo, com base nas informações fornecidas no *script* de testes.

O MIF reside no sistema em teste implementado como uma camada intermediária entre a IST e a camada inferior. O MIF injeta as falhas determinadas pelo CIF, sendo responsável por três atividades: filtragem (interceptação de mensagens), injeção (injeção da falha em si) e entrega (devolução da mensagem, alterada ou não, ao sistema).

A ferramenta possui uma arquitetura distribuída, a qual foi implementada da seguinte forma: o Sistema de Teste foi inicialmente implementado em plataforma Solaris 2.5, e atualmente reside em plataforma Linux. O *Ferry* Passivo foi inicialmente implementado em plataforma Solaris 2.5 e deve ser migrado para a plataforma em que reside o sistema em teste. O *Ferry* Ativo foi desenvolvido em C++ e o *Ferry* Passivo, por ser o módulo que é alterado com mais freqüência, foi desenvolvido em Pearl. A comunicação entre eles é feita utilizando biblioteca de *sockets*. Além do aspecto portabilidade (os ambientes utilizados até então dispõem de tais bibliotecas), o uso de comunicação via *sockets* também facilita a extensão da ferramenta para os testes com mais de uma IST: o CT pode manter conexões com mais de um *Ferry* Ativo (um para cada IST) sem necessidade de grandes alterações na FSOFIST.

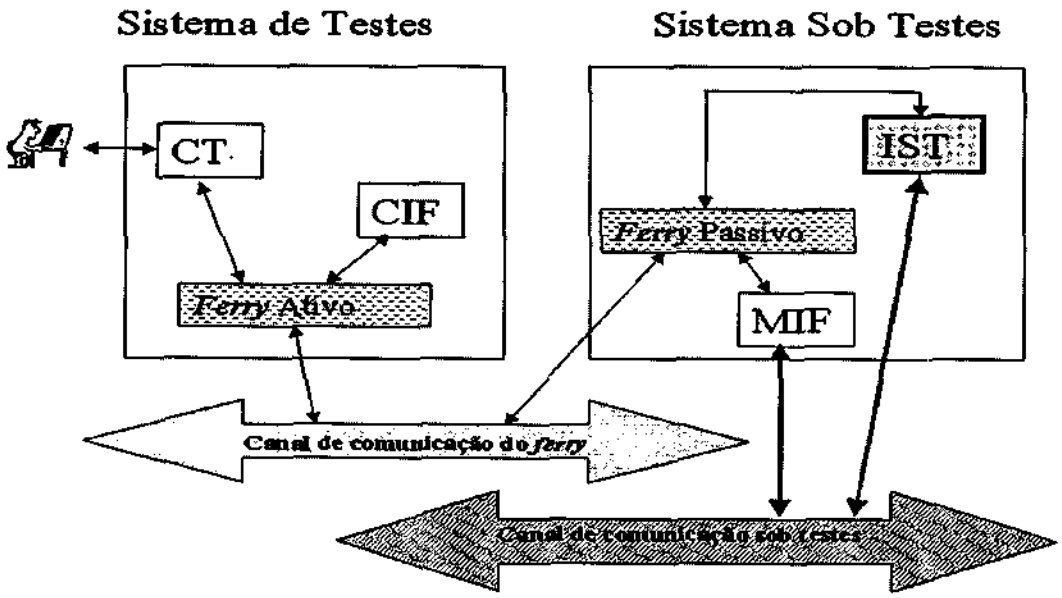


Figura 3.1. Arquitetura da fSofist.

4. USO DA FSOFIST

Esta seção apresenta o uso da FSOFIST na validação de uma pequena aplicação espacial servindo também como um estudo de caso real para avaliação da ferramenta.

4.1 Descrição da aplicaçao

A aplicação utilizada é o Software de Recepção e Armazenamento em Solo dos Dados de Telemetria do Telescópio MASCO - MAScara Codificada. Este software, denominado TMSTATION [1, 15, 20-23], implementa a entidade solo do protocolo de comunicação solo-bordo para recepção em tempo real dos dados de imageamento do céu em raios X realizado pelo telescópio MASCO durante aproximadamente 30 horas de operação do experimento, num vôo em balão. O experimento foi desenvolvido pela Divisão de Astrofísica do INPE – Instituto Nacional de Pesquisas Espacias, na área de pesquisa de Alta-Energia, com lançamento previsto para início de 2003. Os dados adquiridos pelo hardware detector (HD) são organizados em *frames*, armazenados em arquivos e transmitidos pelo computador de bordo em tempo real para a Estação Solo onde são recebidos pelos equipamentos de rádio freqüência e transferidos via comunicação serial para o TMSTATION. A figura 4.1 apresenta a arquitetura desta aplicação espacial.

O TMSTATION é uma aplicação implementada em ambiente LabWindows/CVI [12] que recebe os *frames* de telemetria em tempo real transmitidos pelo Computador de Bordo do MASCO e os armazena separadamente em arquivos (um *frame* por arquivo). Um *frame* contém 937 sub-*frames* de 146 palavras onde cada sub-*frame* é inicializado pela palavra de sincronismo hexadecimal EB90 seguido do identificador do frame. O TMSTATION recebe os dados de telemetria via interface serial RS-232, identifica cada *frame* e o armazena em arquivo nomeado pela data corrente coletada do próprio relógio do PC da estação de recepção onde o software reside. O nome do arquivo é da forma DDMMAAHHNNSS.dat onde: DD – são dígitos de dia; MM – são dígitos de mês; AA – são os dois últimos dígitos do ano; HH – dígitos de hora (0 até 23); NN – são dígitos de minutos (0 até 59); SS – dígitos do segundo (0 até 59).

O TMSTATION implementa as funções correspondentes aos protocolos dos níveis de transporte e aplicação; as funções básicas da recepção de telemetria, correspondentes aos níveis de Decodificação e Físico, são feitas por bibliotecas disponíveis no LabWindows/CVI.

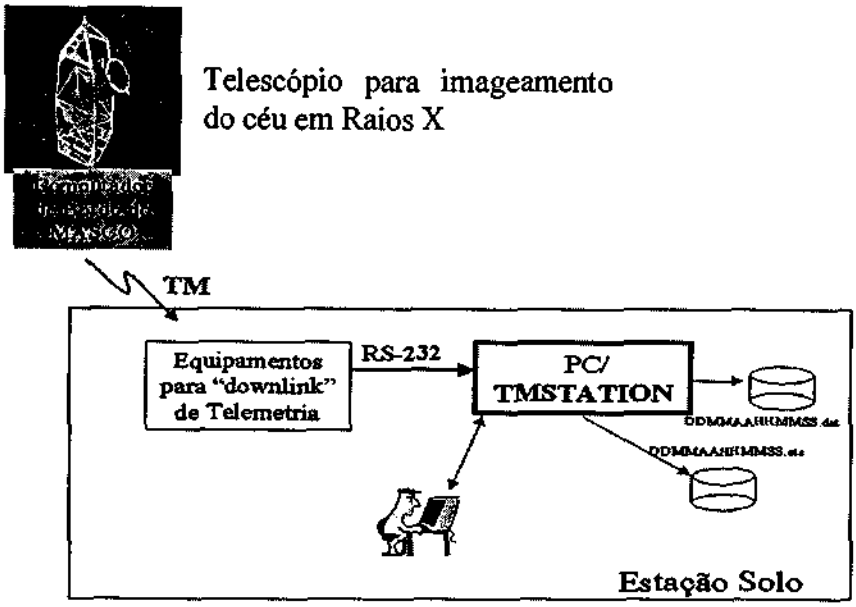


Figura 4.1 Recepção de dados em solo.

O protocolo solo-bordo implementado pelo TMSTATION é muito simples. Basicamente consiste em identificar o padrão de sincronismo EB90 e a palavra seguinte que contém o identificador do *frame* presente em todo *sub-frame*. Associado a cada *sub-frame* recebido, o TMSTATION registra a qualidade da recepção em um arquivo de mesmo nome (DDMMAAHHNNSS.dat) porém com extensão .sts. Se o *sub-frame* recebido for do tamanho esperado (146 palavras) será considerado VÁLIDO e qualificado como "V" no vetor de status. Caso o *sub-frame* contenha menos do que 146 palavras, ele será qualificado como "T", TRUNCADO. E se ultrapassar o valor esperado, assume-se que ocorreu uma FUSÃO de *sub-frames* atribuindo-se o status "F". O programa termina quando o botão de "fim" for acionado manualmente pelo operador.

4.2 Testes de Conformidade

Os testes de conformidade da implementação TMSTATION estão sendo realizados em duas etapas, de forma incremental. A primeira objetiva validar o comportamento do TMSTATION independente do canal de comunicação real. Para este passo foi desenvolvido um *script* executado pelo Controlador de Testes – CT (citado em 3.2) que lê os *frames* de um arquivo (TELEMETRIA .dat) e os fornece para a TMSTATION, conforme mostra a figura 4.2. Tanto o *script* quanto o arquivo TELEMETRIA. dat foram criados manualmente. O arquivo de telemetria baseia-se em 4 padrões de dados:

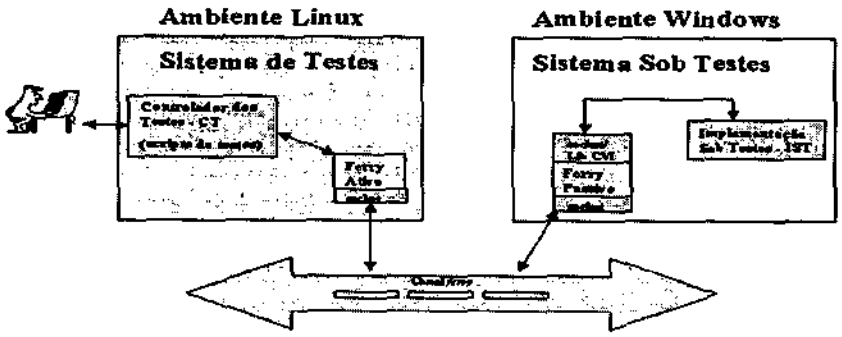
- EB90;
- EB90 + identificador do *frame* + 145 (EA10) – simula um sub-frame VÁLIDO;
- EB90 + identificador do *frame* + 270 (EA10) – simula FUSÃO de 2 sub-*frames*;
- EB90 + identificador do *frame* + 75 (EA10) – simula sub-*frame* TRUNCADO.

O arquivo TELEMETRIA.dat contém casos de testes formados por diferentes combinações destes padrões de dados. Desta forma pode-se simular os dois tipos de perdas de dados decorrentes das possíveis falhas na transmissão bordo-solo previstas na especificação do software:

- Perda do padrão EB90, gerando o que chamamos de FUSÃO (campo de dados de 2 ou mais sub-frames encadeados);
- Perda dos dados de um sub-frame, gerando o que chamamos de sub-frame TRUNCADO (campo de dados menor que as 146 words esperadas)

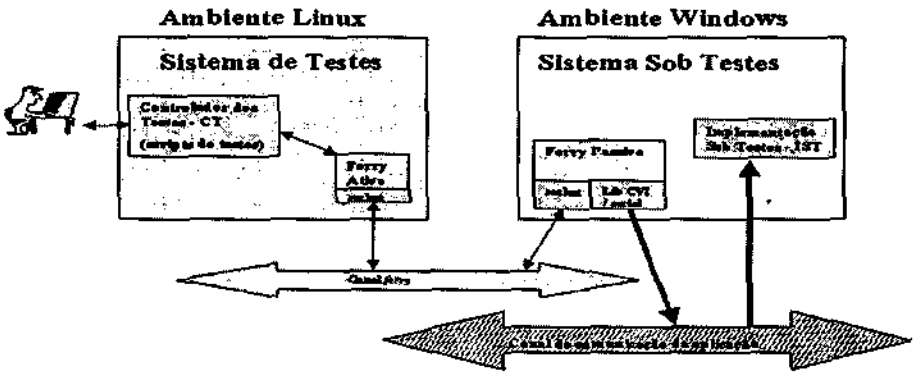
A arquitetura da FSOFIST utilizada para esta etapa de teste é mostrada na figura 4.2a. Ela foi configurada para apoiar os testes locais isto é, sem o uso do canal de comunicação em testes. Neste caso o *Ferry* Passivo se comunica com a IST (TMSTATION) através de *sockets* com uso de bibliotecas CVI.

Configuração da FSOFIST para testar o TMSTATION
Etapa 1



(a)

Configuração da FSOFIST para testar o TMSTATION
Etapa 2



(b)

Figura 4.2 Configurações da FSOFIST usadas nos testes.

A segunda etapa de testes objetiva validar a versão final da IST isto é, através da comunicação pela linha serial. Para apoiar estes testes a arquitetura da FSOFIST foi configurada como apresentado na figura 4.2b.

Para os testes de conformidade os componentes: Controlador de Injeção de Falhas – CIF - e o Módulo de Injeção de Falhas – MIF - não foram incorporados à arquitetura.

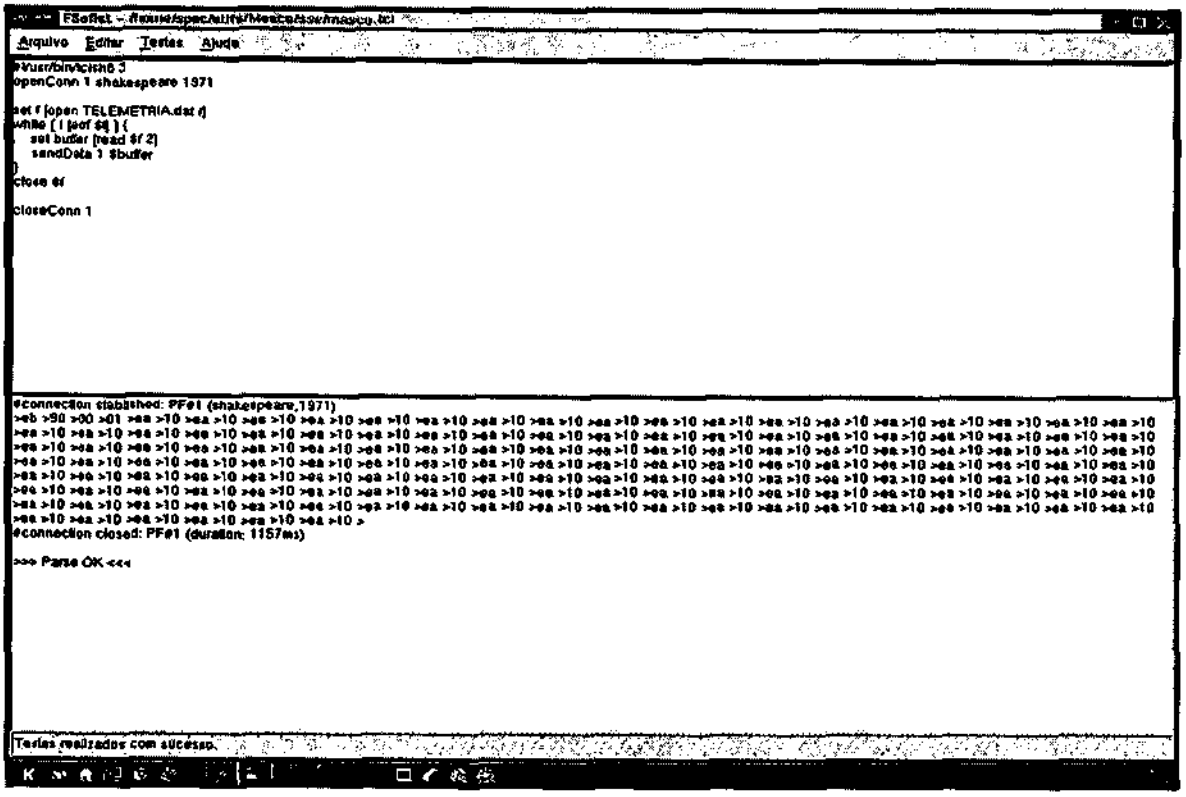


Figura 4.3. Interface da fSofist.

4.3 Execução dos testes e resultados

A figura 4.3 apresenta a imagem de tela de execução da FSOFIST. Na parte superior pode-se ver o *script* de testes utilizado nas duas etapas dos testes de conformidade. As primitivas *openconn / closeconn* fazem parte do protocolo de transferência de dados utilizado pelo Sistema de Testes para comunicação entre seus componentes e o *Ferry* Ativo. Elas servem para abrir / fechar conexão entre o CT (que executa o script) e o *Ferry* Ativo. A primitiva *senddata* é usada para o envio de dados para a IST.

A parte inferior (área de respostas) mostra os dados enviados para a IST durante os testes. Caso a IST envie alguma resposta (o que não é o caso aqui), estas também são exibidas nesta área.

Como resultado da primeira etapa de teste pode-se detectar um erro de especificação do TMSTATION. Ao se aplicar o caso de teste que consistia de uma seqüência de *frames* válidos, a implementação considerou o primeiro *sub-frame* de cada *frame* como válidos e os demais como truncados. Isto ocorreu devido a omissão, na especificação, de uma ação associada à contagem de bytes do *sub-frames*.

Na segunda etapa de testes não foram detectados erros de conformidade.

Com relação ao uso da ferramenta FSOFIST nestas duas etapas, pode-se observar que as alterações de configuração foram mínimas. A única alteração consistiu em adaptar o *Ferry* Passivo para comunicação via linha serial. O Sistema de Testes permaneceu inalterado.

4.4 Próximos passos

A realização dos testes de conformidade evidenciou a necessidade dos testes por injeção de falhas para validar o comportamento do TMSTATION em situações não previstas nos casos de testes aplicados com os padrões de dados descritos em 4.2. Por exemplo, ao invés de provocarmos erros em momentos pré-estabelecidos, a inserção deles se faria de forma aleatória através da perda ou corrupção de um ou mais bytes do *frame*.

Por esta razão, apesar de não terem sido previstos inicialmente, os testes por injeção de falhas serão aplicados no TMSTATION utilizando a arquitetura completa da FSOFIST.

Depois disso, finalmente, serão realizados os testes de interoperabilidade com o computador de bordo do MASCO. Estes testes serão aplicados em ambiente de operação e permitirão validar possíveis incompatibilidades entre a implementação que monta os *frames* no computador de bordo e a que os recebe em solo.

5. CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou os testes de conformidade de uma pequena aplicação espacial utilizando a ferramenta FSOFIST. A

aplicação utilizada é o Software de Recepção e Armazenamento em Solo dos Dados de Telemetria do Telescópio MASCO - MAScara Codificada. Este software recebe, em tempo real, os dados de imageamento do céu em raios X realizado pelo telescópio MASCO durante aproximadamente 30 horas de operação do experimento, num vôo em balão. O experimento foi desenvolvido pela Divisão de Astrofísica do INPE – Instituto Nacional de Pesquisas Espacias, na área de pesquisa de Alta-Energia, com lançamento previsto para início de 2003.

Os testes de conformidade permitiram detectar uma falha na especificação do software, que foi corrigida. Estes testes iniciais mostraram a necessidade de mais testes, em especial, por injeção de falhas, para exercitar melhor os mecanismos de detecção de erros implementados. Dado que os casos de testes foram gerados manualmente, poucas situações de erros puderam ser geradas.

Por outro lado esses testes permitiram avaliar o uso da ferramenta FSOFIST, desenvolvida em meio acadêmico, na validação de uma aplicação real. Embora a aplicação seja bem simples, esse uso foi muito promissor, pois permitiu aos desenvolvedores, tanto da ferramenta quanto da aplicação, compreender melhor o potencial oferecido por uma ferramenta desse tipo. Os testes eram realizados manualmente e exigiam um grande esforço a cada nova configuração, esforço esse que muitas vezes era perdido, pois a infra-estrutura desenvolvida para os testes dificilmente era reutilizada nos testes de novas aplicações. A ferramenta já oferece essa infra-estrutura, suficientemente flexível de modo que poucas modificações sejam necessárias para configurá-la para o uso em novos projetos. Foi possível constatar a facilidade de adaptação da FSOFIST para novas plataformas. As modificações para isso afetam unicamente os componentes da ferramenta que residem no Sistema sob Teste.

Uma outra vantagem do ponto de vista prático é que a ferramenta não requer alterações no processo de desenvolvimento, pois tudo o que ela necessita é do código executável da aplicação. Conseguiu-se portanto um ganho em eficiência na realização dos testes, sem maiores impactos na forma como o software é desenvolvido. A ferramenta é simples de usar, não sendo necessário um grande esforço para treinamento no seu uso. Os *scripts* são escritos em TCL, que tem uma estrutura sintática muito similar a da linguagem C, utilizada pelos desenvolvedores da aplicação.

Certamente são necessários mais estudos de caso utilizando aplicações reais de maior porte para fazermos uma avaliação melhor da aplicabilidade da ferramenta, e também para identificarmos melhorias a serem feitas para tornar seu uso mais adequado às necessidades dos usuários. Outras aplicações estão sendo previstas para esse fim.

Uma outra atividade futura consistirá em estender a ferramenta para dar apoio a outros tipos de testes: de interoperabilidade e de desempenho. Este último, em especial, é de extrema importância para esse tipo de aplicação. Já está em andamento um estudo visando melhorias à FSOFIST para permitir a realização de testes de desempenho.

No mais esse estudo mostrou que a colaboração entre o meio acadêmico e o produtores de software pode ser extremamente profícua para ambas as partes, pois a resolução de problemas enfrentados por desenvolvedores, na prática, podem resultar em assuntos de pesquisa bastante interessantes.

REFERÊNCIAS

[1] Alves, A.M. R.; Rinke, E.; da Silva, E. R.; Fernandes, J.O.; Correa, R. V.; Villela, T.; Braga, J.; D'Amico, F. O Sistema de Aquisição de Dados do Projeto MASCO. Boletim da Sociedade Astronômica Brasileira, 17(1), 68, 1997.

[2] Arlat, J.; Crouzet, Y.; Laprie, J.-C.. Fault injection for dependability validation of fault tolerant computing systems. Proc. Fault Tolerant Computing Symposium (FTCS)-19, Chicago, EUA, jun/1989.

[3] Araújo, M. R. R. fSofist- Uma ferramenta para teste de protocolos tolerantes a falhas. Dissertação de mestrado Instituto de Computação – Unicamp, out/200

[4] Bochmann G.v., Petrenko A. 1994. Protocol Testing: Review of Methods and Relevance for Software Testing. Proc. Intern. Symposium on Software Testing and Analysis (ISSTA), pp109-124.

[5] Chanson, S. T., Lee, B. P., Parakh, N. J., Zeng, H. X., Design and Implementation of a Feery clip Test System, Proc. 9th IFIP Symposium on Protocol Specification Testing & Verification, Ensechede, Holanda (Junho 1989).

[6] Chanson, S. T., Vuong, S., Dany, H., Multi-party and interoperability testing using the ferry clip approach , Computer Communications, vol 15, n0 3 (Abril 1992).

[7] Clark, J. A. & Pradhan, D. K, Fault Injection: A Method for Validating Computer System Dependability. *IEEE Computer*, jun. 1995, pp. 47-56.

[8] Dawson, S.; F. Jahanian,; T. Mitton. ORCHESTRA: a Fault Injection Environment for Distributed Systems. Available on site: http://www.ecs.umich.edu.

[9] Echtle, K.; M. Leu. The EFA Fault Injector for Fault-Tolerant Distributed System Testing. Proc. Workshop on Fault-Tolerant Parallel and Distributed Systems, Amherst, USA, 1992.

[10] Hsueh, M; Tsai, T. K; Iyer, R. K, Fault Injection Techniques and Tools. IEEE Computer, abr/97, pp 52-75.

[11] ISO TC97/SC21, IS 9646. OSI Conformance Testing Methodology and Framework , ISO/1991.

[12] LabWINDOWS/CVI-C for Virtual Instrumentation-User Manual, *National Instruments Corporation Technical Publications*, 1996.

[13] Laprie, Jean-Claude. Dependability – Its Attributes, Impairments and Means. Predictability Dependable Computing Systems (B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood, eds). Springer, Berlin, Germany, 1995, pp 3-18.

[14] Leite J.C.B., Orlando G. Loques F°, Software. II SCTF, cap. 4 do mini-curso intitulado: Introdução à Tolerância a Falhas, Campinas, SP, 1987.

[15] MattielloFrancisco, M.F. "Software Requirements Specification for MASCO Telemetry Ground Reception". Relatório interno MASCO-SRS-001, INPE, 05/1999.

[16] Rosenberg, H.A.; Shin, K.G.. Software Fault Injection and its Application in Distributed Systems. In *Proc. FTCS-23*, Toulouse, França, 1993.

[17] Segall, Z.; D.Vrsalovic; D.P.Siewiorek; D.Yaskin; J.Kownacki; J.Barton; R.Dancey; A.Robinson; T.Lin. FIAT-Fault Injection Based Automated Testing Environment. In *Proc. FTCS-18*, Tokyo, Japan, 1988, pp102-107.

[18] Sotoma, I., Weber, T. S., AFIDS – Arquitetura para Injeção de Falhas em Sistemas Distribuídos, Anais do XV Simpósio Brasileiro de Redes de Computadores, São Carlos/SP (1997).

[19] Stiubiner, S. Testes de Conformidade de Protocolos. Anais do 7° SBRC, 1989.

[20] Villela, T.; Braga, J.; D'Amico, F.; Neri, J. A. "A Balloon-Borne Imaging Telescope for High-Energy Astrophysics", *Revista Mexicana de Astronomia y Astrofisica*, vol.26, 135, 1993.

[21] Villela, T.; Braga, J.; Mejá, J.; D'Amico, F.; Alves, A.; Silva, E.; Rinke, E.; Fernandes, J.; Corrêa, R. "Preflight Tests of the MASCO Telescope", *Advances in Space Research*, vol.26(9)}, pp.1411--1414, 2000.

[22] Villela, T.; Braga, J.; Mejá, J.; D'Amico, F.; Alves, A.; Silva, E.; Rinke, E.; Fernandes, J.; Corrêa, R. "Preflight Tests of the MASCO Telescope", *Advances in Space Research*, vol.26(9)}, pp.1411--1414, 2000.

[23] Villela, T.; Braga, J.; Fonseca, R.; Mej\la, J.; Rinke, E.; D'Amico, F. "An Overview of the MASCO Ballon-Borne Gamma-Ray Experiment", *Advances in Space Research*, aceito, 2001.

[24] Zeng, H.X.; Rayner, D. The Impact of the Ferry Concept on Protocol Testing. Anais do V Protocol Specification, Testing and Verification, 1986, pp533-544.

[25] Zeng, H.X.; Li, Q.; Du, X.F.; He, C.S. New Advances in Ferry Testing Approaches. Computer Networks and ISDN Systems, 15, 1988, pp47-54.

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

AUTORIZAÇÃO PARA PUBLICAÇÃO

Número
EEI-169/2002

Título -1079

Uso da Ferramenta de Testes FSOFIST na Validação de uma Aplicação Espacial

Autor

Eliane MARTINS, Maria de Fátima MATTIELLO-FRANCISCO; Anderson Nunes Paiva MORAIS

Tradutor

Editor

INPE-9554-PRE/5222

Origem	Projeto	Série	No. de Páginas	No. de Fotos	No. de Mapas
ETE/DSS			9		

Tipo

☐ RPQ ☒ PRE ☐ NTC ☐ PRP ☐ MAN ☐ PUD ☐ TAE

Divulgação

☒ Externa ☐ Interna ☐ Reservada ☐ Lista de Distribuição Anexa

Periódico / Evento

Anais da 2a. Jornada Ibero-Americana de Engenharia de Software e Engenharia de Conhecimento
30 Jul a 1 Nov 2002 / Salvador, Bahia, Brasil

Convênio

Autorização Preliminar

__/__/__
Data

Edenilso F.E. Orlandi
Titular da Unidade
Chefe da Divisão de
Desenv. de Sistemas de Solo

Revisão Técnica

☐ Solicitada ☒ Dispensada

Recebida __/__/__ Devolvida __/__/__

Leonel Fernando Perondi
Coordenador Geral
Engenharia e Tecnologia Espacial
Titular de Nível "A"

Assinatura do Revisor

Revisão de Linguagem

☐ Solicitada ☒ Dispensada

Recebida __/__/__ Devolvida __/__/__

Titular de Nível "A"

Assinatura do Revisor

Autorização Final

__/__/__
Data

Edenilso F.E. Orlandi
Titular da Unidade
Chefe da Divisão de
Desenv. de Sistemas de Solo

Palavras Chave

Testes de conformidade de protocolos - injeção de falhas - arquitetura de testes - o conceito de ferry clip

(continua na folha 2/2)

Secretaria

__/__/__ Recebida __/__/__ Devolvida __/__/__

Data

_____ _____

Encaminhado Por Devolvido Por

Controle e Divulgação

__/__/__ Recebido Por: _____ Devolvido Para: _____

Data

Pronto Para Publicação em: __/__/__ ___/___/___

Data

No. _____ Quant._____ _____

Assinatura

Observações